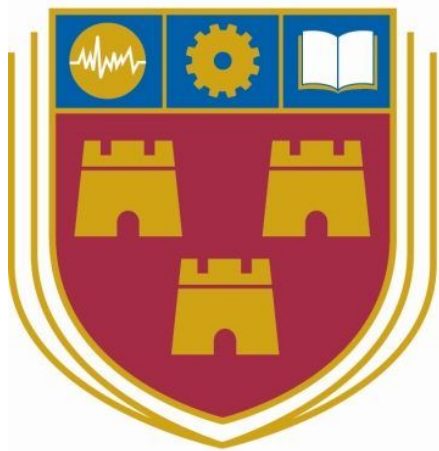


Clinic Management System

Final Report



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

Learning Elixir	3
Learning Elixir: YouTube	3
Learning Elixir - Basic Concepts	4
Learning Elixir: Books	8
Elixir Learning: Elixir Forum & Developer Communication	10
Developer Communication Continued	13
Customer Feedback	14
First customer feedback - Preparation	14
First customer feedback - Issues	15
Continued Feedback	17
UX design	17
Challenges	19
Form data loss	19
Data encryption	21
Security considerations	23
Expansion by Design	23
Future work	24
Overall Experience	25
Acknowledgements	26
References	27

Learning Elixir

Learning Elixir: YouTube

When my tutor first told me about Elixir, I was intrigued. It was a young new programming language that was making waves across the programming community. I had a couple of glaring problems from the start however:

1. I had never heard of Elixir.
2. I had no inclination of what it looked like.
3. I had no idea of the capabilities of the language

Beginning this project, I had no prior knowledge of Elixir or its related technologies, and no knowledge of functional programming. The first place I started, was to watch keynote talks, conferences and tutorials based around Elixir, as a foundation for further learning. Some of the initial (and most informative) video include:

- Elixir Tutorial by Derek Banas **[1]**
- GOTO 2017 • Elixir: The only Sane Choice in an Insane World • Brian Cardarella **[2]**
- José Valim | ElixirConf EU 2017 Keynote **[3]**
- ElixirConf 2017 - Thinking In Ecto - Darin Wilson **[4]**
- Phoenix: an Intro to Elixir's Web Framework - Sonny Scroggin **[5]**
- 18 months of Elixir in production at 2nd largest sport website in the world **[6]**

Through these videos and others like them, I began getting a greater sense of what the language was and what it is in use for. The last link mentioned talks about how Elixir helped Bleacher Report improve their performance of their servers. Previously, Bleacher Report had used Ruby-on-Rails to handle it's 1.5 billion views, 250,00 users and 3 billion push notifications each month. The site reached a point where they could not longer scale it, and began to look into alternatives. After researching many possible options, Elixir was chosen because of its excellent concurrency and scaling abilities, it's fault tolerance and its similar syntax to the pre-existing Ruby installation. Elixir was implemented, and proved so successful, that testing the limits of their service became the largest difficulty faced. Bleacher Report managed to test as much as an 8x increase in average traffic, before things began to fail. Even at that point, it was the database that began to bottleneck and not Elixir. As a result of implementing Elixir, they were able to

reduce the number of servers they had from 150 down to 5, a huge increase in efficiency with a corresponding large increase in performance.

These talks, while not giving me any technical knowledge, did give me some more knowledge on the current uses of Elixir, and some of the features and capabilities it has.

Learning Elixir - Basic Concepts

The project was laid out in two parts, although not explicit, a clear divide was in place. The later part of the project, was to develop a clinic management system using Elixir and its associated technologies, however a sizeable portion of the project time, and a key component of the overall project was to learn to use Elixir and surrounding libraries.

There was a large learning curve involved with learning Elixir, as most of the concepts and “way of mind” of developing using it was different to any other I’d learned previously. Up to the point of starting the project, most languages I’ve used have been Object Oriented, as opposed to a Functional language such as Elixir.

Object-oriented programming is a design based on “objects” - data structures contain attributes of data code in the form of methods (or procedures).

Functional programming “treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data” [7].

Functional programming holds some basic advantages over Object-Oriented Programming (OOP).

- Code can have fewer bugs, as it does not support state, meaning “side-effect” results are minimized and code free of errors can be written.
- Parallel programming is easier, as functional languages have no mutable state, meaning there are no issues involving state changing, supporting easy reusability and testability.
- Functional programs can be more efficient, as they consist of independent units that can run concurrently.

A good way to demonstrate the differences between OOP and functional programming is to give some code examples. The first is the standard “new language” practice of demonstrating a fibonacci sequence using recursion.

Fibonacci sequence in Java:

```
static int n1=0,n2=1,n3=0;

static void printFibonacci(int count){

    if(count>0){

        n3 = n1 + n2;

        n1 = n2;

        n2 = n3;

        System.out.print(" "+n3);

        printFibonacci(count-1);

    }

}
```

[8]

Fibonacci sequence in Elixir:

```
defmodule Fibonacci do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n), do: fib(n-1) + fib(n-2)
end
```

[9]

As we can see, Elixir (representing functional programming) seems to create a shorter overall piece of code than Java (representing OOP). The syntax for Java tends to be more verbose than Elixir, and as such once a grasp of Elixir has been gained it can allow for shorter, quicker writing of code with less potential for bugs or errors.

Although simpler than the Java example, the Elixir example can require some prior knowledge of the syntax for defining modules (the equivalent of classes in Java). These syntax differences are relatively easy to pick up, as it is a similar process to learning other new programming languages. Where the languages differ however, is that Java only allows for one declaration of a given function, in this case *printFibonacci*. However, Elixir has multiple declarations for the function *fib*, with each declaration handling a different input.

This is one of the first challenges I faced when learning Elixir: knowing a function can be declared multiple times.

Another item from Elixir that I found difficult to grasp, but indispensable when it came to building the actual system, was the atom variable. Atoms are constants, whose name is their own value (they may be called symbols in other languages). When I first was introduced to these variables, I could not understand their purpose, however I now know their main uses. Atoms can be used in the following contexts:

- To call functions - Atoms can be used to store a function, meaning a function can take standard variables as arguments, but also a function and then return a function, in the form of an atom.
- To declare data types - When designing the model to be used with the model-view-controller design paradigm, in order to declare the datatype of a field, for example: `:string`, `:binary`, `:date`.
- To handle errors - When an operation is carried out on the database, many languages will return a status code. In Elixir, errors are returned via a tuple, containing an atom (eg. `:ok`, `:error`) and a list of errors

Once I understood the power and uses of these variables, they allowed functionality in the system that would not been as elegant.

The main usage, was when inserting data into the database from a form. The data is passed from the form, to the controller, which attempts to insert the data into the database. In order to do this, it is passed through the model, where any required fields, generation of primary keys etc. are validated. If any of these fail, the data is not inserted, and an error is generated, and an `:error` atom returned, along with the errors that occurred. This is then relayed back to the user, who can act accordingly.

The way Elixir handles the insertion results, is via a concept known as pattern matching, which was the concept I had the most difficulty understanding, however once it was understood, it was a powerful tool that was used in many aspects of the system.

In most OOP languages, the “=” symbol assigns variables. In Java, $x = 2$ could be written in English as ‘*give the variable x the value of 2*’. This logic is learning at the early stages of beginning to write code, however this is undone in the case of Elixir (and other functional programming languages).

In Elixir, the “=” symbol is called the match operator. If we write the same code using Elixir ($x = 2$), it is interpreted as “match the value of the left hand sign of the match operator, to the right hand side of the operator”. If the expression is invalid, a match error occurs.

The above example confused me, and I could not understand the difference until I viewed some other examples, at which point it began to become clearer. Here’s a working example:

```
{a, b, c} = {:hello, "world", 42}
```

If we tried to use to logic baked in from OOP, we would say this would not work in any language (a tuple of items of varying types cannot be assigned to one another). However with Elixirs pattern matching fundamentals, it makes perfect sense.

- The compiler checks both sides of the match symbol, and attempts to match them up.
- The compiler observes the left (a tuple of three items) and the right (a tuple of three items) matching, and so assigns the variables.

After running this code, we could print ‘a’ and get ‘:hello: or ‘b’ and get “world”.

It was only when I tested out inputs in a sandbox environment that I truly began understanding the concept. Here is some code containing an error:

```
{a, b, c} = {:hello, "world", 42, "Ryan"}
```

Again, the compiler attempts to match both sides of the operator, however in this case it cannot match, as the left side contains 3 items, and the right side 4.

Up to this point, all learning and research I had carried was via the Elixir learning guides [10]. Only once I had grasped the rest of the differences between OOP and Elixir (of which there were many more), could I move on to learning how to take these concepts and turn them into working applications.

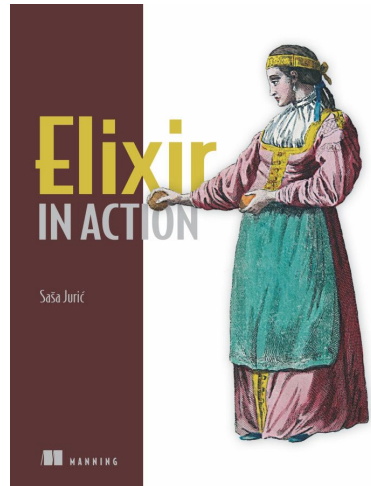
Learning Elixir: Books

Once I had gained a better understanding of the core functionalities of Elixir, I began looking towards creating basic applications. Although discovering a healthy community for Elixir while searching online for “how to’s”, I found the language for these guides to be slightly more advanced than was necessary for me to follow. My tutor recommended a book website for developers - The Pragmatic Bookshelf [11] - which may have some books that might help me get started and start gaining more development experience with Elixir. I supplemented this website with other similar websites and read some excerpts from the books and decided to purchase a number of options (some of which were written or contributed to by the creator of Elixir, José Valim) and also a book dedicated to Elixir's web framework, Phoenix.

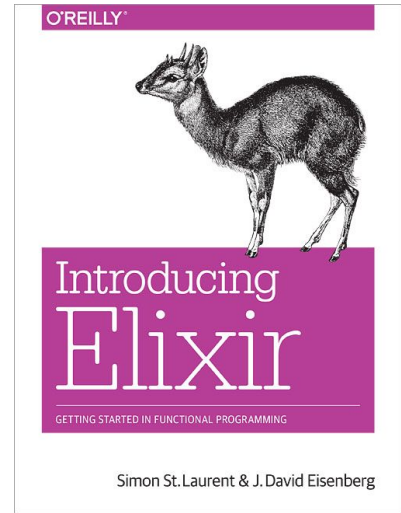
I began reading these, just reading at first without following any of the code tutorials. I found that not only did all of these take adopt a “painting by numbers” approach to teaching, but they also explained every concept and rationale behind decisions being made.



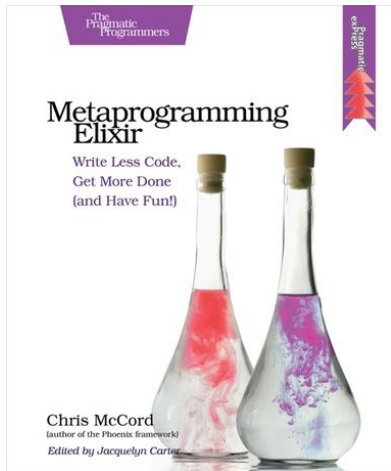
1



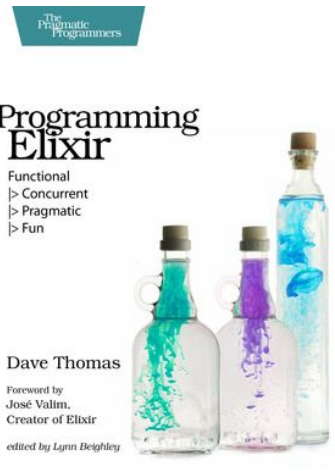
2



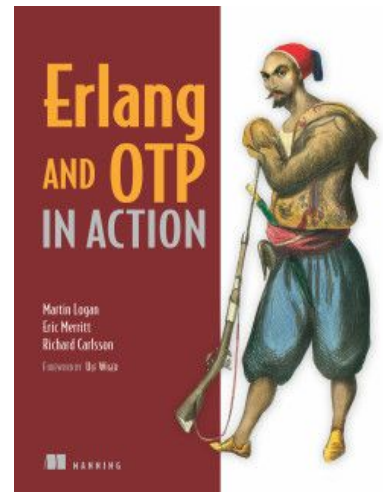
3



4



5



6

1. "Programming Phoenix" by Chris McCord, Bruce Tate and José Valim [12]
2. "Elixir In Action" by Saša Jurić [13]
3. "Introducing Elixir" by Simon St. Laurent, J.Eisenberg [14]
4. "Metaprogramming Elixir" by Chris McCord [15]
5. "Programming Elixir" by Dave Thomas [16]
6. "Erlang and OTP In Action" by Ulf Wiger [17]

These books were one of the best learning sources I encountered over the course the project, and although some of the books had similar content, they were explained slightly differently each time. This really reinforced some of the key concepts and allowed me to progress further. Programming Phoenix was indispensable when it came to learning about Phoenix, and quickly became a source I referred to throughout the duration of the project.

Elixir Learning: Elixir Forum & Developer Communication

The screenshot shows the Elixir Forum user profile for Ryan Donoghue. At the top, the forum logo is on the left and search, menu, and user icons are on the right. Below the header is a dark purple banner with the user's profile picture (a purple circle with 'R'), the username 'r_donoghue', and the name 'Ryan Donoghue'. To the right of the name is an 'Expand' button. Below the banner are navigation tabs: 'Summary' (selected), 'Activity', 'Messages', 'Badges', and 'Preferences'. A 'STATS' section follows, displaying: 34 days visited, 3h read time, 2h recent read time, 50 topics viewed, 242 posts read, 8 hearts given, 1 bookmark, and 5 topics created. Below this are 15 posts created and 3 hearts received. Two columns of 'TOP REPLIES' and 'TOP TOPICS' are shown. The 'TOP REPLIES' column lists several posts about Ecto foreign keys/associations and one about refreshing a dropdown list. The 'TOP TOPICS' column lists topics about refreshing a dropdown list, Ecto foreign keys/associations, and EEX templates. A 'More Replies' link is at the bottom left.

[18]

The Elixir Forum played a crucial role to the success of the project. It allowed me to speak with other users of Elixir and developers of modules in order to troubleshoot issues, contribute and open discussions and even contribute towards the development of Elixir libraries.

Here are some statistics from my activity on the forum:

- 34 days visited
- 3 hours read time
- 50 topics viewed
- 242 posts read
- 5 topics created
- 15 posts created

For the most part, I used the forum when errors occurred in my code that I could not find solutions for. Other times, it was used to ask for suggestions for how to approach a functionality

that I could not figure out how to implement, mostly down to lacking missing knowledge on how to use or the existence of particular libraries.


Every major problem I had with development was solved in part by the community. When a post was made on the forum, within days there were hundreds of views and many replies, offering different solutions each with little to no overlap.

I gained an opportunity to give something back to the community by finding a bug in a developers library - Drab by Tomek "Grych" Gryzkiewicz. This module, was used to perform live updating of fields within a form, without losing data already existing on the form. I began to implement the solution, using an implementation specific to the system I was developing, however ran in to some issues. I decided to contact the developer and maintainer of the library in the attempt to remedy the issue. While Grych was able to find and suggest a fix for the error, it spawned another error, later identified as a bug.



grych

Feb 23

 r_donoghue:



This seems to have stopped the error anyway, however the assigns is not being updated.

set_prop will not update the assign. It is just an equivalence of JS

```
document.querySelector(selector).property = ...
```

I assume that nothing happens on the page, when you do the set_prop I've gave you? Please notice it was just an example, I guessed the ID of the node element you want to update and was probably wrong 😞 Please provide the correct CSS selector of this element and all should work correctly.



r_donoghue

Feb 24

Your guess of the ID wasn't far wrong 😊 the page is rendering the select with an ID of patient_pharm_id, and I used this value, however still no update. One thing that's worth noting, when using some of the socket query functions in iex, I can access the select field with no issues, and run the command manually, returning a tuple of {:ok, 1}, but still no update to the field 😞



grych

Feb 24

Oh. It should update the option list. You must found the bug...
What version of drab do you use? You can check it with `mix deps`.



After identifying I was using the correct version of the library, it was identified as a bug. Grych proceeded to fix the bug, include it in his v0.7 update and thank me for finding the bug.

Developer Communication Continued

After helping Grych identify a bug needing to be fixed, I also contacted the developer of the library I chose to implement the encryption of data, Cloak - Daniel Berkompas. I had some issues getting up and running with the library, and while relatively detailed, the documentation for the library differed between sources. The documentation was located on Github and also HexDocs, with different levels of complexity, however there were a number of holes in both regarding the setup process. I emailed Daniel with a description of the problem I was having, and he advised to create an issue on the Github repo for the library so other community members and users of the library could see.

The screenshot shows the GitHub interface for the repository 'danielberkompas / cloak'. The top navigation bar includes 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the repository name, there are buttons for 'Unwatch', 'Unstar', and 'Fork'. The 'Issues' tab is selected, showing a search filter 'is:issue is:closed'. A list of 28 closed issues is displayed, with columns for 'Author', 'Labels', 'Projects', 'Milestones', 'Assignee', and 'Sort'. The issues listed are:

Issue Title	Author	Closed	Comments
Is poison optional?	jdewar	25 days ago	1
Clarification on usage of 'default' in documents	jdewar	25 days ago	1
Argument error	monting	Mar 7	1
Use with mnesia?	beardeddeagle	25 days ago	3
Make json library configurable	danielberkompas	Jan 2	
Migrations in multi tenant applications	tomciopp	Jan 6	6
Encryption Argument Error	r-donoghue	Nov 27, 2017	
** (MatchError) no match of right hand side value: false	r-donoghue	Nov 24, 2017	1

[19]

The issue created by me is last in the list included in the screenshot. As a result of this issue, Daniel helped me fix the issue (which was a relatively simple one and mostly a mistake on my behalf) and updated the documentation for the module to cover the areas missed.

Customer Feedback

First customer feedback - Preparation

The first meeting with the customer was treated as a formality, and to retrieve further requirements for the application. When beginning the development of the application, a preliminary list of modules and fields were received from the customer as a basis to start development on (as a POC rather than a finished list of user requirements).

The customer walked me through some of the process' and pain points associated with the current method of data entry.

- The customer currently uses a pen and paper system for storing data. This includes forms, paper documents, printed spreadsheets and written notes.
- Any information needed on a patient requires searching through a sizable portion of files held in filing cabinets, security being lock and key.
- Similarly, updating a patients' records (if they change their phone number for example) requires a manual search through documents to find the correct documents and update the record.
- Analysis is incredibly difficult. As part of the day to day running of the clinic, the clinic is required to submit statistics to the HSE for quality assurance purposes. Again, this means trawling through hundreds of records to find, for example, the number of patients that are female and over 40 years of age.
- All patient details are logged into a secondary system run by the HSE that can offer some statistics, but none the clinic is legally allowed to use. This system is purely used for the HSE to enact auditing on the clinics.
- The clinic has had difficulties with migrating to technology based solutions. They only recently received laptop computers to handle day to day running, and there is no funding available to create a system for the clinic.

The customer then ran through some of the process' in more details, and expressed some of the requirements for the system they would like:

- Auto generated fields such as age, based on the the date of birth of the patient
- The ability to enter and track a Clinic Doctor/General Practitioner training level (this is required to ensure they can legally administer prescriptions)
- General analytics. For example, the ability to find out how many patients in the clinic that are from a specific area, or a specific age bracket

Before bringing the first iteration of the application to the customer, the project was brought to my project tutor for UX validation and suggestions. Some of the items that were pointed out included

First customer feedback - Issues

The first iteration of the application was completed 15/02/2018. There was a number of challenges associated with getting the application into the hands of the end user, mainly from a security. The original plan, was to install Elixir, Phoenix and the other technologies associated with the management system on the customer's work laptop, and then arrange a meeting sometime thereafter (for example, one week). This would allow the the customer to use the system prior to the meeting and a more productive talk can be achieved if the discussion is based on prior knowledge rather than a walkthrough of the system (where bugs and UX issues may not be caught).

An issue with this solution arose however when attempting to install the system. As the clinic is owned by the HSE, security measures are put in place to prevent unauthorised software, and software requiring administrator privileges to be installed. Thus the application and it's environment could not be installed. A new solution was required.

The solution decided between my tutor, Paul Barry and I, was to host the website on a server owned by the college (Institute of Technology Carlow) called Glasnost. This server is running an older version of Linux Ubuntu Server, and as such further knowledge on the installation of the project to this platform (which was developed locally on a Windows 10 machine) was required.

The corresponding version of Ubuntu Server was installed via VMWare Workstation 14 on my local machine, and configured to match the Glasnost server:

Technology	Version
Elixir	1.6
PostgreSQL	9.5

The some secondary packages were required including NodeJs, NPM (Node Package Manager), Git, Hex and Brunch.io. Depends on NPM (by default)

Brunch.io : Used to compile static assets such as JavaScript, CSS, uses NPM by default to install dependencies.

NPM : Used to download and installed open-source building block packages of code from developers. Required by Brunch.io. Depends on NodeJs

NodeJs : JavaScript runtime built on Chrome's V8 JavaScript engine. Required by NPM.

Once these dependencies had been installed, the application can be cloned from a GitHub repository to the server and the final configuration can be done. The application cloned from GitHub is a Phoenix Framework and as such, does not require additional dependencies to be installed prior to running the application, as all necessary dependencies required by the code are retrieved, compiled and installed at build time automatically. The database however does not yet exist, however due to the integration of Ecto into Phoenix, this is made relatively simple.

Phoenix includes commands such as the following:

- *mix ecto.create*
- *mix ecto.migrate*

mix ecto.create :

Searches the Phoenix project source code for an environment configuration file. The environment configuration file holds the username, password and location of the PostgreSQL database. This then creates the database specified within the config file.

mix ecto.migrate :

This command is responsible for setting up and making changes to database tables and fields. When a new portion of the project is created (via *mix phoenix.gen.html*), a corresponding migration is created that holds the new table name, fields, field types and relationships to other tables. When a change to a database table, field or field type is required, a migration must be created. When the command *mix ecto.migrate* is run, these migrations manipulate the database in the desired way.

The advantage of using these commands from a deployment point of view, is that this allows for more seamless updates to the system, as the database can be created the first time the system is run, and even if the entire system source code is deleted, the database will remain. This is ideal from a deployment perspective, as changing the source code will not have a direct impact on the underlying data, which is of high importance especially within a medical setting. The migration aspect also allows built in rollbacks, which means if a new update to a field or type breaks the system in the development environment. It can simply be rolled back to assess the cause of the issue.

When a new update has passed through the development environment, in order to update the system, all that needs to occur is to delete the source code, pull the new code from a remote repository (GitHub in this case), the dependencies mentioned above installed and the database migrated if required.

Development of the local server deployment was now complete and deployment to a live server was ready, with the exception of one technology that was installed on the Glasnost server: NGINX.

NGINX is used to server applications on a server, so when a user navigates to the address of the server (ardu.itcarlow.ie) a specified port is then served to the user (in this case, port 4000 as is default by the Phoenix project). This configuration was created by the administrator of the Glasnost server, as superuser privileges are required. This configuration would potentially need to be completed by the developer/me in future deployment solutions.

Continued Feedback

The client played an important role in the project, and continued feedback was vital to creating the best system possible for the user. Iterations began, I would develop new functionality and implement it in the system and the client would then use the system as they would during a normal working day, adding records, updating records, to evaluate how the system feels for use. I would then meet with the client to get feedback from them, and have a discussion about what they liked, what they disliked and what they wish they were able to do. These observations from the client paved the way for me to decide what needed improvement, and what new features to implement.

The best example of implementing the clients feedback was the reports section of the system. The user had a desire to be able to see an overview of all patients, so they can see who is with which doctor, how many patients do I have under forty etc.. This feedback was supplemented by suggestions from my tutor, who suggested the ability to be able to view a patient's doctors details from the report screen by simply clicking on their name. My tutor also suggested the ability to sort a table by any field (e.g. age, county, name), which proved so successful I implemented it in all screens where a list of records are present. The report screen is a module that was not thought of by me, and whose idea came directly from the needs of the client.

UX design

Over the course of developing the project, there was a number of UI and UX choices that were implemented by me, some that were influenced by the client and some that were directly suggested by the client.

- **Simplistic and easy to read**
 - A decision was made to attempt to make the system as simplistic and easy to read as possible. The user that typically would use this system is generally two or more generations before my own, and this has to play a role in the system. A system that is too complicated would cause the user to become frustrated and not enjoy using the system, whereas having the system too simple may omit base functionalities the user requires.
- **Appropriate error messages**
 - A hallmark of a good system is the ability to inform users of errors in as clear and concise a way as possible. If a user encounters an error, they must be able to see what the issue is, how it was caused (from a high level) and how to fix it.
- **Simple page instructions**
 - Some of the functionality of the system was not clear at first to the user. It was discovered through feedback from the client, that certain operations were carried out without realising the ramifications it would have. This was especially evident when adding a patient whose information requires some additional information be added to the system. The first thing the client would do is press back, navigate to the page that needs to be updated, adding information and attempting to return to the patient form, only to discover all information previously entered had been lost. This was resolved, and a message is given to the user at the beginning of the form, informing them of how to approach this situation and how not to lose information already entered
- **Layout of form fields**
 - Some of the early feedback from the client was based around the layout of the system forms, specifically inputs and their labels. The labels, in their state before feedback, were situated above their corresponding input box. It was suggested by the client to move these labels to the same line as the input boxes. This resulted in an overall more readable system, and allowed for a better differentiation between fields. A small touch that made a noticeable difference to the experience of using the system.
- **Calculation of ages**
 - My tutor, Paul Barry, suggested automatically calculating a patient's age and storing it once the user had entered the patient's date of birth. This again, is a small detail that makes the users life easier, and cuts down on the additional work previously required to calculate this manual.

- **Calculation of future dates**
 - Paul Barry added another suggestion surrounding dates, calculating future dates. When the user is recording a patient vaccination record, they must enter dates for three vaccinations required. These vaccinations are required to occur after certain time intervals (Dose two five months after dose one, dose three one month after dose two), and he suggested it would be more efficient if the system intelligently calculated the future dates of vaccination, which again cuts down on the calculations the user must enact and improve the overall flow of the system.
- **Change of address layout**
 - A small change I made to the system was to split the address from one continuous record, to a series of separate boxes for each element, ie. town, city, county etc. . This allows for easier and clearer entry of information into the system, and also allows for reports and extended functionality to be implemented in the future.

Challenges

Form data loss

One of the largest issues encountered during the development of this system, involved forms within Phoenix. The best way to explain the problem that occurs, is to give a sample process a user of the system.

First, some pre-information. The system was designed with expandability and self-sufficiency in mind for key areas. The mentality behind this, was to create a self-sustainable system, that can be configured to some degree by the system. The fields that allow for this customization include:

1. **Patient Relationships** - These values are used when recording a new or existing patient's next of kin. When adding or updating a patient, the user must select from a dropdown list the relationship the patient has to the designated next of kin. This field, in most cases, would be a direct relation (mother, father, brother, sister), however the next of kin can vary between patients (not all patients are from the same background or family structure). The customization allows a user of the system to add relationships that are not already in the system, for example if a new patient arrived who's next of kin was their aunt. This means the options are not prebaked, can be sustained by the users and dependent on developers.
2. **Patient Genders** - As part of adding a new patient to the system, a user must record what gender the patient is. In modern society, there has been an explosion in the number of genders people choose to identify by, and there must be an allowance for this built into the system. Similar to Patient Relationships, if a database of genders is maintained by the users rather than by the developers, it would result in a more positive UX. If a gender has not yet been entered into the system before, a user can record this detail, and any subsequent adding of patients can leverage this detail.
3. **Patient Inactivity Reasons** - Patients who are registered with the clinic may be defined as inactive or active. If the patient is inactive, a reason for this must be stored. Some example reasons include the patient is deceased, the patient is in hospital or the patient is currently in prison. Similar to the two previous fields, this data would benefit from being maintained by the users rather than the developer.

4. **Vaccination Brands** - When recording a vaccination record for a specific patient, the user must select what brand of vaccination they are / will be receiving. If one of these brands may change in the future or disappear entirely, the system must have the capability to store the new vaccination brands. Similar to the aforementioned fields, there is benefit to be had from these fields being maintained by the users rather than the developers.

These customizable fields are aimed at reducing the amount of maintenance developers must do post installation, in favour of the users of the system adding options for fields where appropriate.

The example of typical system use where this could cause problems in the state the application was in, is this:

- The user wants to add a new patient to the system, or update an existing patient.
- The user navigates to the new / update patient form, and begins filling out details.
- The user realises that the value for a customizable field does not yet exist, and as such must add this new item to the database.
- The in order to capture this new information, the user must either (A). Navigate to the appropriate customization portion of the system, record the new data and navigate back to the patient details form (all within the same window) or (B). Open a new tab/window, navigate to the appropriate customization portion of the system, record the new data, close the window and reconvene entering the patient data.
- There are issues with both of these approaches. In method (A) navigating away from the current page you are on would cause the destruction of the data already on the form, meaning the entire data recording process would have to be restarted. Method (B) does not require navigating away from the page, however even if the patient details are successfully been preserved up to this point, the page and form and page has already been rendered, meaning any options for dropdown fields are baked in and cannot be changed without refreshing the page, which in turn would destroy all data due to Phoenix's handling of Cross Site Request Forgery (CSRF) tokens.

In order to better understand, and potentially remediate this issue, I posted on the Elixir Forum (discussed later in this document). From here, I received suggestions for implementations and workarounds, however one in particular stood out from the rest: the Drab library. What's interesting, is the user who posted the advice turned out to be the lead developer of Drab.

Drab is a library created by developer Tomek “Grych” Gryszkiewicz. The webpage for the library reads:

“Drab is the extension library to Phoenix Framework for providing an access to the browser’s User Interface (DOM objects) from the server side. The main advantage is to eliminate necessity of writing two applications: one for the client-side, and one for the backend. All the UI control may be now done in the backend, eliminating JS and AJAX.

Additionally, because HTTP is a stateless and an one-way protocol, it is not easy to communicate back from the server to the browser. Like, for example, during execution of the long running process, when there is a need to update status (eg. progress bar), or to ask the operator about something (like “would you like to continue?”). Of course there are workarounds for this, like polling the server every few seconds to get a status, or even auto-refreshing the page, but those are just a dirty workarounds. The solution would be to allow an access to the interface directly from the server side.” [20]. To simplify this and relate it to my problem, Drab allows for an interface layer between the User Interface, and the server side. As mentioned by the webpage, there are solutions such as polling (hitting the server with requests every few seconds to get updated data) however this could have a notable effect on the performance of the system, simply to resolve an issue that occurs less than 1% of the use of the system.

Drab was implemented into the code, and allows for server side polling whenever particular buttons are pressed (in this case a refresh button for relevant fields). When the user clicks refresh, the application handles the button click, grabs the required data from the database and updates the field within the form, without refreshing the entire form and losing the data already entered. This solved the problem, and although there were some additional challenges, once the initial troubleshooting was done, the code required for implementing the library with the existing code was relatively simple.

Data encryption

One of the tasks that caused issues and required a workaround was implementing an encryption module that encrypts sensitive patient data being added to the PostgreSQL. The module being used is called Cloak (v 0.3.3), which plugs into Elixir and Phoenix to offer certain encryption capabilities. One of the features of Cloak is the ability to seamlessly integrate with Ecto, by offering extended database types that handle encryption and decryption simply by declaring the field type. For example, if we take the following database schema code:

```

defmodule User do
  use Ecto.Model

  schema "users" do
    field :name, Cloak.EncryptedBinaryField
  end
end

```

In this case, the variable “name” is represented as an encrypted binary field (all strings in Erlang/Elixir are based on binaries). When a new changeset is added to the database, methods are called from the Cloak module that encrypts the data, with a similar process occurring for decryption.

This feature however would not work with this particular project as for unknown reasons Erlang could not detect The developer of the module, Daniel Berkompas, was contacted and no obvious logic defects were found.

A workaround was created, that defines the database field as being binary rather than a Cloak custom field:

```

defmodule User do
  use Ecto.Model

  schema "users" do
    field :name, Cloak.EncryptedBinaryField
  end
end

```

Before a changeset is added to the database, the field is encrypted by calling the encryption functions from Cloak, with the following syntax:

```

Patient.changeset(changeset.data, Map.merge(changeset.changes, %{patient_id: Cloak.encrypt(Ecto.Changeset.get_field(changeset, :patient_id))}))

```

Simplified, this portion of code merges the previous changeset with a new temporary changeset with the name field encrypted, resulting in a new changeset with name encrypted.

This workaround comes with a drawback however. When the data from the database is pulled, to be listed shown individually (for example an individual patient), the encrypted field is displayed in its encrypted format, meaning the data is harder to use. The solution to this was the decrypt the values of the fields once they have reached the html template they are destined for.

Security considerations

As mentioned previously, there was a strong requirement for the data held by the system to be encrypted to approved standards. The data encryption method chosen was Cloak, which allows for AES-256 bit encryption, which is approved by the HSE. One of the issues I noticed while implementing the encryption is the way the key is stored for encryption and decryption. By default, the library stores the key in a config file within the Elixir project. This is not optimal, and storing a key in plain text is not an advised way of storing a key in any situation. The most obvious solution, is an industry standard key storage mechanism or using dedicated hardware to store the keys. While this is the most secure method, it is well beyond the budget of the system in its current state. The next best method, is to store the key in a dedicated file on the server, which is then imported to an environment variable. This means the code for the system can be held within source control (public or private), but the file holding the encryption key is not included in source control. If a user gains access to the application code through source control, the code will only point to the file, which is not accessible. The only way an attacker could gain access to the key, is gaining access to the server it is located on, which would compromise the system regardless of where the key is stored (unless on dedicated hardware).

Another consideration that was made while building the system, is vulnerability to Cross Site Scripting (XSS), SQL injection and Cross Site Request Forgery (CSRF) attacks.

XSS protection is handled straight out the box by Phoenix, by sanitizing all inputs received from the browser before they reach sensitive code or data.

Similarly, SQL injection protection is handled by default by Ecto, with all queries being sanitised before being enacted on the underlying PostgreSQL database.

CSRF is handled natively by Phoenix also, which auto generates a CSRF token which is passed transparently between pages to mitigate against this attack.

Expansion by Design

An early observation (and an important consideration) came from my tutor, Paul Barry, in the form of a single question:

“Currently this system is designed for one particular clinic in Carlow, however what happens if another clinic in for example, Kilkenny wants this system? Do they need another redesigned system?”

This had me thinking about the overall design of the system, and how it could be designed with future expansion in mind. After thinking for some time, I decided on an implementation I thought would be optimal. I decided to create a two new entities, User and Clinic.

When a new clinic is required (i.e a new clinic wants to use the system), a new record is created holding the details of the clinic, as well as a clinic id. The logic behind this, is so now a patient, vaccination record, phlebotomy record etc. can be associated with a particular clinic. Next, users are created, and associated with a clinic. A new nurse for a clinic in Carlow would be associated with the Carlow clinic, and a new nurse for the Kilkenny clinic would be associated with the Kilkenny clinic. This logic means that code can be written so a user within a clinic can only see patient data from that clinic (privacy by design).

This approach gave rise to the need for multiple privilege levels for users. Users can either be standard or admin, with standard users having the ability to manipulate patient records, clinic records and other details for the clinic they are assigned to, with admin users able to add clinics and users, but cannot see any patient details for any clinics existing in the system.

An issue that arose from this approach, is that once the system is up and running in a default state, there is no data existing in the system regarding users. This means that in order to create a user, a user must be able to login, which is not possible if they don't already exist. The solution to this was to create a script that runs when the system is installed that allows for pre-baked users to be added to the system without the need to login. This script will install a default clinic (as an admin still needs to be assigned to some clinic) and a default admin, along with other details the system needs to improve the UX from the start.

Another point worth mentioning which applies to the design of the system is the omission of a “delete” function within the user interface. While this functionality has been coded in and is possible, it is not accessible by any user, as to do so would have ramifications on the security of patient data.

Future work

Although there was a lot of work carried out over the course of the project, there are still areas I would have liked to have worked on more, given more time. Some these include:

- **Transfer protocols**
 - One of the requirements from the HSE in terms of data encryption, is that transfer protocols are encrypted for the moving of data. This is especially important in the case of this system, which is accessible from any web browser in its current configuration. The currently approved transfer protocols include IPSec (IP Security), SSL (Secure Socket Layer), SSH (Secure Shell), TLS (Transport Layer Security), S/MIME (Secure Multipurpose Internet Extension).
- **Additional functionality in the application**
 - There is further functions the system does not currently cover, including some day to day operations of the clinic. However under the scope of this project, the core functionalities plus some extra were implemented. The clinic can always have new functionality added, there's always something that can be added to give more use to the user.
- **Further user feedback**
 - The user feedback received over the course of the project was the best way to make the system as appealing and easy to use. This feedback was gained not only vocally, but also by observing the client as they were using the system, and seeing areas they might have struggled in. The more feedback that can be received from the user, the better the system will become.

Overall Experience

Overall, developing using Elixir was a fun and educational experience. I found Elixir to be a challenging language to learn, however once I had learned enough about it to develop independently, I found it a joy to use. The documentation format for Elixir and its libraries was clear and detailed where necessary, meaning they were a valuable source of information from a syntactic standpoint, but also from an education standpoint. The community behind Elixir is a testament to the language, with many individuals happy to help out and suggest fixes and alternatives.

The development experience while using Elixir was a positive one, and it led me to learn some new things as a result, such as the model-view-controller (MVC) design philosophy. Up to this point, I was aware of the concept, but had never implemented it myself. Now that I have some experience with it, I find it an excellent tool and makes larger projects much more manageable. In particular, using the model portion of MVC gave great flexibility to my coding, and allowed me to implement functions such as encryption with much more ease than previous projects.

Elixir's database interface, Ecto, was a breath of fresh air, and a new approach to interfacing with a backend database. It allows for excellent error checking and debugging, meaning development was a lot more informative than anything I had previously used. On top of this, Ecto allows for greater control over the database, meaning if and when I needed to add a field to a database, or create an association between databases, creating these changes was handled well by Ecto. When a change was created, a record is kept of what changed, and allows for the database structure to be rolled back to a previously working state.

Phoenix was an easy tool to use, and I leveraged my previous experience with Python's Flask web framework to seamlessly integrate with the new web framework. It built upon my pre-existing knowledge of Flask, and added more functionality on top of it.

The experience was challenging, but extremely rewarding. I started the project knowing "standard" OOP languages such as Java and C# and ended the project having built an entire system from a functional language I had never heard of, let alone ever coded in.

Acknowledgements

I would like to take the opportunity to thank some people who helped me greatly during the course of this project, without whom most of this project would not have been a success.

- **Paul Barry**
 - A huge thanks to Paul, who suggested the initial idea (a spin on the project I'd submitted at the beginning of the year) and dedicated a large portion of his time to helping me learn and develop over the course of this project, Without his assistance, help and reassurance this project would not have come to fruition in the fashion that it did. Also, a big thanks for the constant bombardment of "is this okay" or "what do you think" over the course of the last 6 months.
- **The Elixir Forum**
 - A big thank you to the community on the Elixir Forum. From the offset and throughout, they were kind and helpful, no matter how vague or "silly" the question, and played a large role in helping remediate errors and keep my sanity.
- **Ardú HSE Substance Misuse Service, Carlow**
 - Thank you to the clinic in Carlow, for allowing me to access their protocols and processes, to better understand the current configuration of the clinic.
- **Tomek "Grych" Gryzkiewicz**
 - Tomeks help and feedback when using his library helped me understand the problem I was having, and ultimately his patience really helped solve my issues and educate me on the solution.
- **Deirdre, Ardú HSE Substance Misuse Service**
 - Finally, thank you very much Deirdre for taking time out of her day to speak to me on multiple occasions, for giving excellent and detailed feedback, and for being an instrumental factor in the successful outcome of the project.

References

- [1] YouTube. 2018. Elixir Tutorial - YouTube. [ONLINE] Available at: <https://www.youtube.com/watch?v=pBNOavRoNL0>. [Accessed 16 April 2018].
- [2] YouTube. 2018. GOTO 2017 - Elixir: The only Sane Choice in an Insane World - Brian Cardarella - YouTube. [ONLINE] Available at: <https://www.youtube.com/watch?v=gom6nEvtl3U>. [Accessed 16 April 2018].
- [3] YouTube. 2018. José Valim | ElixirConf EU 2017 Keynote - YouTube. [ONLINE] Available at: <https://www.youtube.com/watch?v=IZvpKhA6t8A>. [Accessed 16 April 2018].
- [4] YouTube. 2018. ElixirConf 2017 - Thinking In Ecto - Darin Wilson - YouTube. [ONLINE] Available at: <https://www.youtube.com/watch?v=YQxopjai0CU>. [Accessed 16 April 2018].
- [5] YouTube. 2018. Phoenix: an Intro to Elixir's Web Framework - Sonny Scroggin - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=F-7MX_Az6_4. [Accessed 16 April 2018].
- [6] Erlang Solutions. 2018. Erlang Solutions. [ONLINE] Available at: <https://www.erlang-solutions.com/resources/webinars.html#18-months-of-elixir-in-production-at-2nd-largest-sport-website-in-the-world-26>. [Accessed 16 April 2018].
- [7] Functional programming - Wikipedia. 2018. Functional programming - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Functional_programming. [Accessed 16 April 2018].
- [8] www.javatpoint.com. 2018. Fibonacci Series in Java - Javatpoint. [ONLINE] Available at: <https://www.javatpoint.com/fibonacci-series-in-java>. [Accessed 16 April 2018].
- [9] Karol Słuszniak. 2018. Is Elixir programming really that hard? - Phoenix on Rails blog - Cloudless Studio. [ONLINE] Available at: <http://cloudless.studio/articles/38-is-elixir-programming-really-that-hard>. [Accessed 16 April 2018].
- [10] elixir-lang.github.com. 2018. Introduction - Elixir. [ONLINE] Available at: <http://elixir-lang.github.io/getting-started/introduction.html>. [Accessed 16 April 2018].

- [11]** The Pragmatic Bookshelf . 2018. The Pragmatic Bookshelf . [ONLINE] Available at: <https://pragprog.com/>. [Accessed 16 April 2018].
- [12]** McCord, C., 2016. Programming Phoenix: Productive |> Reliable |> Fast. Pragmatic Bookshelf.
- [13]** Jurić, S., 2015. Elixir in Action. Manning Publications.
- [14]** St., S., 2014. Introducing Elixir: Getting Started in Functional Programming. O'Reilly Media.
- [15]** McCord, C., 2015. Metaprogramming Elixir: Write Less Code, Get More Done (and Have Fun!). Pragmatic Bookshelf.
- [16]** Thomas, D., 2014. Programming Elixir: Functional |> Concurrent |> Pragmatic |> Fun. Pragmatic Bookshelf.
- [17]** Logan, M., 2010. Erlang and OTP in Action. Manning Publications Company.
- [18]** Elixir Forum. 2018. Profile - r_donoghue - Elixir Forum . [ONLINE] Available at: https://elixirforum.com/u/r_donoghue/activity. [Accessed 16 April 2018].
- [19]** GitHub. 2018. Issues · danielberkompas/cloak · GitHub. [ONLINE] Available at: <https://github.com/danielberkompas/cloak/issues?q=is%3Aissue+is%3Aclosed>. [Accessed 16 April 2018].
- [20]** Drab: Server Side User Interface Access. 2018. Drab: Server Side User Interface Access. [ONLINE] Available at: <https://tg.pl/drab>. [Accessed 16 April 2018].