WEB BASED APPLICATION FOR EQUIPMENT MANAGEMENT - OCP
Student Statement:
I have applied ethics to the design process and in the selection of the final proposed design. I also have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable.

_____________________________________________________
Kaoutar Arhlane

Approved by the Supervisor

Dr. Fouad Mohammed ABBOU
Professor
School of Science & Engineering
Al Akhawayn University
E-mail: F. Abbou@au.ma
http://www.au.ma/personal/~F.abbou

_____________________________________________________
Dr. F.M. Abbou
Acknowledgement

I would like to express my sincere gratitude for all the people that have helped me work on this project. First of all, I would like to thank my supervisor Dr. Fouad Mohamed Abbou for the valuable support and advice that he provided me with throughout the semester. Secondly, I would like to thank my family and all of my friends for their continuous encouragement. Finally, I would love to thank every single person that helped me during my working time on this project.
# Table of Content

Acknowledgement .................................................................................................................. 4  
Table of Content ..................................................................................................................... 5  
Table of Figures ....................................................................................................................... 6  
Abstract ..................................................................................................................................... 7  
1. Introduction .......................................................................................................................... 8  
   1.1. Context of the Project ................................................................................................. 8  
   1.2. Project Scope .............................................................................................................. 8  
   1.3. STEEPLE ANALYSIS ............................................................................................... 8  
2. Feasibility study and analysis ............................................................................................... 9  
   2.1. Technical feasibility: ................................................................................................. 9  
   2.2. Economic feasibility: ............................................................................................... 10  
   2.3. Technological feasibility: .......................................................................................... 10  
3. Framework analysis ............................................................................................................ 10  
   3.1. Elixir overview .......................................................................................................... 11  
   3.2. Ecto ......................................................................................................................... 11  
   3.3. Phoenix Framework ................................................................................................. 13  
4. Software Engineering methodology ..................................................................................... 16  
   4.1. Requirements ............................................................................................................ 16  
      4.1.1. Specific Requirements ...................................................................................... 17  
      4.1.2. User/functional Requirements ......................................................................... 17  
      4.1.3. Non-functional Requirements ......................................................................... 19  
      4.1.4. Interface Requirements ................................................................................... 19  
      4.1.5. ER Diagram ...................................................................................................... 20  
      4.1.6. Use-Case Diagram ......................................................................................... 21  
      4.1.7. Class Diagram ................................................................................................... 22  
   4.2. Design and Implementation ......................................................................................... 23  
      4.2.1. Creating The Models: ..................................................................................... 23  
      4.2.1.2 Creating the Controllers: ............................................................................. 23  
      4.2.1.3 Template (HTML code example) ............................................................... 24  
   4.3 Testing ......................................................................................................................... 33  
5 Conclusion ............................................................................................................................. 36  
References .................................................................................................................................. 37
Table of Figures

Figure 1 Schema.................................................................................................................. 12
Figure 2 Structure .................................................................................................................. 15
Figure 3 ERD ............................................................................................................................ 20
Figure 4 Use Case Diagram ..................................................................................................... 21
Figure 5 Class Diagram .......................................................................................................... 22
Figure 6 Model ........................................................................................................................ 23
Figure 7 Controller ................................................................................................................... 24
Figure 8 Template .................................................................................................................... 25
Figure 9 Screenshot .................................................................................................................... 26
Figure 10 Screenshot .................................................................................................................. 27
Figure 11 Screenshot .................................................................................................................. 27
Figure 12 Screenshot .................................................................................................................. 28
Figure 13 Screenshot .................................................................................................................. 29
Figure 14 Screenshot ............................................................................................................... 29
Figure 15 Screenshot .................................................................................................................. 30
Figure 16 Screenshot ............................................................................................................... 31
Figure 17 Screenshot .................................................................................................................. 31
Figure 18 Screenshot .................................................................................................................. 32
Figure 19 Screenshot .................................................................................................................. 32
Figure 20 Testing structure ...................................................................................................... 33
Figure 21 Unit Testing .............................................................................................................. 33
Figure 22 Unit Testing .............................................................................................................. 34
Figure 23 Integration Testing ................................................................................................... 35
Figure 24 Integration Testing ................................................................................................... 35
Abstract

OCP - Office Chérifien des Phosphates - is one of the largest and biggest companies in Morocco. It is in fact the world’s leading producer of phosphate rock and phosphoric acid as well as one of the leading global fertilizer players, with more than 90 years of history serving agriculture. The many departments they have as well as the variety of equipment they use require the usage of many applications and software in order to manage the huge load of work and processing done within the different domains of the company.
1. Introduction

1.1. Context of the Project

The internal administrative work within the OCP is a large network that is composed of many interacting departments using a large load of different equipment and software. The work done need to be continuously tracked, managed, and supervised in order to offer a smooth and efficient result that is free from errors and mistakes since it could affect a big aspect of the whole company income and reputation. One of the tasks needed is the management and supervision of the equipment available and used.

1.2. Project Scope

The purpose of this application is to develop and implement a web application that allows the different users and administrators of the IT department to manage, track, and supervise their stock of equipment, report any incident, make a list of requests in case of a system failure, track the request and who made it at what time, and to view these data in form of statistical information.

1.3. STEEPLE ANALYSIS

The STEEPLE analysis had to be taken care of during the development and implementation of this project:

- **Societal**: This application does not have any specific societal involvement since it is specific to OCP apart from the ease of communication between departments.
• **Technical:** This application can be used by anyone who is working within the company and has knowledge about the different equipment of their own department, the user interface is as friendly as possible and does not require any complicated skills from the user.

• **Environmental:** Other than the source of energy that is used by the devices that run the application, it does not have any impact or relation with the environment.

• **Ethical:** This application does not involve any unethical behavior, it actually makes sense that equipment management and handling is done in the most efficient and transparent way possible.

• **Political & Legal:** This application is legal and does not have any political involvement whatsoever.

• **Economic:** The application does not need any economical resources as it is meant to be used internally by the OCP community.

**2. Feasibility study and analysis**

**2.1. Technical feasibility:**

This application is technically feasible since all the tools to make it are available to download and use for free.
2.2. Economic feasibility:

This application is economically feasible and it only requires a working laptop. No extra expenses are required for it to be implemented and working.

2.3. Technological feasibility:

The application shall be web based and only used internally by the members of the OCP group given the nature of the application.

3. Framework analysis

Phoenix is a web development framework written in Elixir. It is based on the Model View Controller pattern; it processes data in the model, read or write that data through the controller.

**Web Framework Objectives:**

- "Distributed Web Services" framework
- Common tasks should be easy
  - WebSockets
  - Realtime events
  - SOA
- No productivity sacrifices for performance
- No performance sacrifices for productivity
3.1. Elixir overview

Elixir is a programming language that is written and working on top of the Erlang; which is a general purpose programming language; Virtual Machine, and just like Erlang, it is a functional language. Its purpose was to support the distributed (systems where components in different networks on different computers communicate between them via message passing) and fault-tolerant (applications that can keep functioning despite the failure of one or more components) applications. Elixir is also known for its strong pattern matching.

```
iex> {first, second, third} = {[:lions, :tigers, :bears]}
{:lions, :tigers, :bears}
iex> first
: lions
```

3.2. Ecto

Ecto is a domain specific language for writing queries and interacting with databases in Elixir.

Ecto is split into four main components:
WEB BASED APPLICATION FOR EQUIPMENT MANAGEMENT - OCP

- **Ecto.Repo** - repositories are wrappers around the data store. Via the repository, we can create, update, destroy and query existing entries.

- **Ecto.Schema** - schemas are used to map any data source into an Elixir struct.

![Figure 1 Schema](image)

- **Ecto.Changeset** - changesets provide a way for developers to filter and cast external parameters, as well as a mechanism to track and validate changes before they are applied to your data.

- **Ecto.Query** - written in Elixir syntax, queries are used to retrieve information from a given repository.
Ecto Configuration:

```elixir
# Configure your database
config :ticketing, Ticketing.Repo,
adapter: Ecto.Adapters.Postgres,
username: "postgres",
password: "postgres",
database: "ticketing_dev",
hostname: "localhost",
```

3.3. Phoenix Framework

Phoenix as a framework is supported on Linux, OS X and windows, can also run SQL, MySQL and PostgreSQL. It known for being fast and concurrent. An example would be when you have two database fetches; you won’t have to artificially group them together with a stored procedure or a complex query. You don’t have to wait for the combined time for three database requests. Your code will take as long as the single longest database request.
In terms of scalability and extensibility, Phoenix has a good set of abstractions that are helpful. Applications written with Phoenix break down into individual functions using the concept of pipelining; denoted by the |> pipe operator, it takes the value on the left and passes it as the first argument to the function on the right. We call these compositions pipes or pipelines, and we call each individual function a segment or pipe segment. For example “connection |> phoenix” would usually be written as phoenix (connection). The browser would establish a connection with an end user using this big function called “phoenix”; In Phoenix, that connection is the whole universe of things we need to know about a user’s request. It is a struct, which is a map with a known set of fields. The connection comes in with information about the request: whether it’s HTTP or HTTPS, what the URL is, what the parameters look like. Then, each layer of Phoenix makes a little change to that connection. When Phoenix is done, that connection will have the response in it.

When Phoenix generates a new application for us, it builds a top-level directory structure like this:
WEB BASED APPLICATION FOR EQUIPMENT MANAGEMENT - OCP

Most of our work is in the web directory, which looks like this when expanded:

```
|-- build
|-- config
|-- deps
|-- lib
|-- priv
|-- test
|-- web

|-- controllers
| |-- page_controller.ex
|-- models
|-- static
| |-- assets
| | |-- images
| | | |-- phoenix.png
| | |-- favicon.ico
| | |-- robots.txt
| | |-- vendor
|-- templates
| |-- layout
| | |-- app.html.eex
| |-- page
| | |-- index.html.eex
|-- views
| |-- error_helpers.ex
| |-- error_view.ex
| |-- layout_view.ex
| |-- page_view.ex
|-- router.ex
|-- gettext.ex
|-- web.ex
```

Figure 2 Structure

**Phoenix actions**
4. **Software Engineering methodology**

The development of this application implements the agile methodology.

The agile software engineering method, is an adaptive and flexible way that is described by the following characteristics:

- Involvement of the customer and their satisfaction
- Late changes in requirements are taken into consideration
- Frequent delivery of increments of the project
- Simplicity in both the software and its development process

4.1. **Requirements**

Specific requirements of the OCP application are listed in this section.
4.1.1. Specific Requirements

The specifications of the application are as followed:

1. User/Functional requirements
2. Non-Functional requirements

4.1.2. User/functional Requirements

There are three types of users that can operate this application:

I. Admin
II. Member
III. Super admin

- Super admins
  - Super admins shall view all tickets submitted and who submitted them as well as the timestamp of each operation.
  - Super admins shall update a ticket.
  - Super admins shall create a ticket.
  - Super admins shall view users.
  - Super admins shall create a user.
  - Super admins shall update a user.
  - Super admins shall search for tickets.
  - Super admins shall search for users.
  - Super admins shall view interventions.
Super admins shall create an intervention.

Super admins shall view departments.

Super admins shall create a department.

Super admins shall update departments.

Super admins shall view equipment.

Super admins shall create equipment.

Super admins shall update equipment.

- **Admins**
  
  - Admins shall view tickets.
  
  - Admins shall search for tickets.
  
  - Admins shall search for users.
  
  - Admins shall create a ticket.
  
  - Admins shall update a ticket.
  
  - Admins shall view interventions.
  
  - Admins shall create an intervention.
  
  - Admins shall view equipment.
  
  - Admins shall create equipment.
  
  - Admins shall update equipment.

- **Members**
  
  - Members shall create a ticket.
Members shall view tickets.

Members shall view interventions.

Members shall create interventions.

4.1.3. Non-functional Requirements

✓ The application should be fast
✓ The application should be reliable
✓ The application should be scalable
✓ The application should fault tolerant

4.1.4. Interface Requirements

The interface should be intuitive and user friendly
4.1.5. ER Diagram

The ER Diagram shows the different tables and the relationships between them.
4.1.6. Use-Case Diagram

Figure 4 Use Case Diagram
### 4.1.7. Class Diagram

**User**
- id: Integer
- name: String
- password: String
- email: String
- role: Enum
- department: Integer
  + viewUsers()
  + addUser()
  + updateUser()
  + viewIntervention()
  + updateIntervention()
  + addEquipment()
  + viewIntervention()
  + updateIntervention()
  + addUser()
  + viewIntervention()
  + updateIntervention()
  + addDepartment()
  + viewIntervention()

**Departments**
- id: Integer
- name: String
- description: String
- status: Enum
- equipment: Integer
  + equipment()

**Tickets**
- id: Integer
- title: String
- description: String
- status: Enum
- equipment: Integer
  + ticket()

**Equipments**
- id: Integer
- name: String
- manufacturer: String
- purchase_date: Date
- warranty: String
- serial_number: String
- call_number: String
- department: Integer
  + interventions()

**Interventions**
- id: Integer
- title: String
- description: String
- type: String
- ticket: Integer
  + interventions()

---

**Figure 5 Class Diagram**
4.2. Design and Implementation

This section is about the design and implementation phases, it provides the decomposition of the whole and overall system into modules. The implementation and testing go along since the incremental model is adopted.

4.2.1. Implementation and technology used

In this section I will talk about the process of the application implementation process such as the development environment used.

4.2.1.1 Creating The Models:

An example of a model creation (Intervention) where the schema for the database is set, adding the references to other tables in the database

```elixir
defmodule Ticketing.Intervention do
  use Ticketing.Web, :model

  schema "interventions" do
    field :title, :string
    field :description, :string
    field :type, :string
    belongs_to :ticket, Ticketing.Ticket
    belongs_to :user, Ticketing.User
    timestamps()
  end
end
```

Figure 6 Model

4.2.1.2 Creating the Controllers:

Example of the InterventionController:
4.2.1.3 Template (HTML code example)

Where the creation of the table and the form code is
4.2.2. Development environment

- **PC:** MacOS Sierra Version 10.12.3, 6 GB RAM, 2.5GHz Intel Core i5

- **Atom:** Text editor with a simple and good interface.

- **Postico:** Database Client (PostgreSQL client for mac)

- **DBVisualizer:** Database management and analysis tool (ER diagram generator)
4.2.3. Application Screenshots

Login/authentication – Case of wrong input

Logged in SuperAdmin – Users table
Figure 10 Screenshot

Updating a User

Figure 11 Screenshot
Creating a new equipment
WEB BASED APPLICATION FOR EQUIPMENT MANAGEMENT - OCP

Validating/inserting the entry

Figure 13 Screenshot

Figure 14 Screenshot
Creating a new Intervention

Figure 15 Screenshot
Listing the available Interventions made by a user

**Figure 16 Screenshot**

Editing a Ticket already inserted.

**Figure 17 Screenshot**

Logout
WEB BASED APPLICATION FOR EQUIPMENT MANAGEMENT - OCP

Figure 18 Screenshot

Trying to access any page without login in

Figure 19 Screenshot
4.3 Testing

4.3.1 Unit Testing

Elixir has a unit testing framework called ExUnit. The structure of the test folder that contains the unit test under the Models folder:

![Testing structure](image)

Figure 20 Testing structure
each model test has the unit testing for that specific model

Example of User Unit Testing

```elixir
defmodule Ticketing.UserTest do
  use Ticketing.ModelCase
  alias Ticketing.User

  @valid_atrs email: "email@email.com", name: "some content", password: "password", phone: "som", role: "some content"

  test "changeset with valid attributes" do
    changeset = User.changeset(%User{}, @valid_atrs)
    assert changeset.valid?
  end

  test "changeset with invalid attributes" do
    changeset = User.changeset(%User{}, @invalid_atrs)
    refute changeset.valid?
  end
end
```

Figure 21 Unit Testing

Running the Unit Testing using the command line mix test test/models

```
mix test test/models
```

Finished in 0.7 seconds
10 tests, 0 failures

Randomized with seed 0x649

Figure 22 Unit Testing

4.3.2 Integration Testing

Phoenix/Elixir use Hound library for Integration Testing

Test functions are defined under the Controllers folder
Figure 23 Integration Testing

Running the Integration Test

Figure 24 Integration Testing
5 Conclusion

This capstone project was a great opportunity for me to get to know a bunch of technologies and methodologies that I was not really familiar with, especially in the field of web development. I had the opportunity to dive a little bit into the aspects of a new web framework and explore many of its functionalities that do not exist elsewhere in what I have experienced before. Realizing that the world is running fast and evolving on a rather faster paste, fast growing businesses and technologies, new programming languages with new added values and features, making a lot of what seemed a little bit impossible to do kind of easy and intuitive for many programmers and software engineers around the world. I was able to create a web application from scratch with the appropriate design and requirements that have been asked for.

The main requirements that the company wanted have been met; however, there’s a need for a further revision of the whole system that needs to be taken care of before releasing the final version that needs to be deployed and installed on the internet servers within the OCP group.
References

