ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

DESARROLLO DE PROTOTIPO DE APLICACIÓN QUE MUESTRE EL CARNET ESTUDIANTIL EN IOS

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

JORGE LUIS MOLINA MORILLO
jorge.molina02@epn.edu.ec

DIRECTOR: RÁUL DAVID MEJÍA NAVARRETE, M.Sc.
david.mejia@epn.edu.ec

DMQ, Febrero 2022

CERTIFICACIONES

Yo, JORGE LUIS MOLINA MORILLO declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

JORGE LUIS MOLINA MORILLO

Certifico que el presente trabajo de integración curricular fue desarrollado por JORGE LUIS MOLINA MORILLO, bajo mi supervisión.

ING. RÁUL DAVID MEJÍA NAVARRETE, M.Sc.

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

JORGE LUIS MOLINA MORILLO

ING. RÁUL DAVID MEJÍA NAVARRETE, M.Sc.

DEDICATORIA

Esto es para aquellos

interesados en saber

como se desarrolló

la app para presentar carnet

El lenguaje Swift se utilizó

y con métodos POST y GET

Web Scraping se necesitó

así esto se puede ver

en la app prototipo

que yo desarrollé

AGRADECIMIENTO

Agradezco a mi familia por el apoyo incondicional que me han dado durante estos 24 años de vida.

Agradezco a toda persona que ha confiado y creído en mí.

Agradezco a mi tutor, el Ing. David Mejía, quien me planteó la idea que dio origen a este trabajo, así como me otorgó el apoyo necesario para la realización del mismo.

ÍNDICE DE CONTENIDO

RESUMEN

Mediante el presente trabajo se obtuvo un producto final demostrable que funciona en dispositivos móviles con sistema operativo iOS, el prototipo en cuestión, puede presentar y almacenar el Carnet Estudiantil de la EPN obtenido desde el sistema SAEw. Como parte de este trabajo fue necesario conocer y estudiar determinadas herramientas, las cuales fueron requeridas para la creación de la lógica y funcionamiento de la aplicación elaborada.

Este documento está organizado en tres capítulos.

En el primer capítulo se mostrará de forma resumida el contenido teórico necesario para la realización de la aplicación, aquí se discutirá sobre: *web scraping*, una técnica de recopilación de información de un sitio web; Swift, el lenguaje para el desarrollo de aplicaciones en iOS; SwiftUI, una forma moderna para declarar la interfaz de usuario; y la metodología Kanban, necesaria para organizar y desarrollar la aplicación prototipo.

En el segundo capítulo se resume el uso de los conocimientos teóricos presentados en el capítulo 1, para el diseño e implementación del prototipo, mientras que en el capítulo 3 se presentarán los resultados obtenidos.

En el anexo se encuentra el código del prototipo desarrollado en Xcode, el mismo que funciona en un dispositivo con sistema operativo macOS y en un dispositivo móvil con iOS.

PALABRAS CLAVE: swift, carnet, web scraping, SAEw, iOS, macOS, xcode, MVVM

ABSTRACT

Through the present work, a demonstrable final product was obtained that works on mobile devices with the iOS operating system, the prototype in question, can present and store the EPN Student Card obtained from the SAEw system. As part of this work, it was necessary to know and study certain tools, which were required for the creation of the logic and operation of the elaborated application.

This document is organized into three chapters.

In the first chapter, the theoretical content necessary to carry out the application will be shown in a summarized way, here it will be discussed: web scraping, a technique for collecting information from a website; Swift, the language for the development of applications in iOS; SwiftUI, a modern way to declare the user interface; and the Kanban methodology, necessary to organize and develop the prototype application.

The second chapter summarizes the use of the theoretical knowledge presented in chapter 1, for the design and implementation of the prototype, while chapter 3 presents the results obtained.

In the annex is the code of the prototype developed in Xcode, the same one that works on a device with the macOS operating system and on a mobile device with iOS.

KEYWORDS: swift, card, web scraping, SAEw, iOS, macOS, Xcode, MVVM.

1 INTRODUCCIÓN

Hasta la fecha en la que se generó este documento, en la Escuela Politécnica Nacional para poder interactuar con el Carnet Estudiantil se pueden usar dos opciones: la primera, mediante un navegador web que acceda al recurso que se encuentra en el SAEw; la segunda, a través de un archivo en formato PDF que contiene el recurso descargado usando la primera opción. El problema detectado es que el SAEw no está optimizado para dispositivos moviles y esto se evidencia en que el contenido visual del carnet corresponde a aproximadamente el 24% del tamaño de la pantalla del navegador. Por lo indicado, se plantea el desarrollo de un prototipo de aplicación que permita presentar el carnet en un dispositivo móvil con sistema operativo iOS.

La visión con la que se desarrolló el prototipo fue la de mantener lo existente y establecer ciertas características en el prototipo para que no sea necesario modificar el SAEw. Para extraer información del SAEw se usó la técnica *web scraping*, la misma que fue codificada usando el lenguaje de programación Swift.

Para el prototipo se empleó el patrón Modelo Vista VistaModelo disponible en el *framework* SwiftUI.

Para poder desarrollar la lógica del prototipo se realizó un estudio detallado del proceso de comunicación con el SAEw, lo cual sirvió como base para la realización de *web scraping*, hay que destacar que para el *web scraping* no se requirió librerías externas, y únicamente se necesitó la API (*Application Programming Interface*) de Swift conocida como `URLSession`.

Para garantizar el funcionamiento correcto del prototipo se realizaron dos tipos de pruebas: el primero, originado con la identificación de errores por parte del desarrollador; el segundo, originado mediante la retroalimentación obtenida a partir de encuestas que fueron realizadas por parte de estudiantes matriculados en la EPN que usaron la aplicación.

1.1 OBJETIVO GENERAL

Desarrollar un prototipo de aplicación que muestre el carnet estudiantil en iOS.

1.2 OBJETIVOS ESPECÍFICOS

1. Analizar las herramientas necesarias para la generación del prototipo.

2. Diseñar los elementos que conforman el prototipo.
3. Implementar el prototipo basándose en el diseño realizado.
4. Analizar los resultados obtenidos de las pruebas.

1.3 ALCANCE

Se trata de un prototipo que permitirá a los estudiantes de la EPN que tengan un dispositivo iPhone obtener el carnet estudiantil mediante el uso de una aplicación. El prototipo será desarrollado en lenguaje Swift y SwiftUI para el caso de iOS.

El prototipo se comunicará con el SAEw para obtener la información necesaria del carnet, para poder realizar esto, se obtendrá mediante *web scraping* la información que se recupere del SAEw a través de pedidos y respuestas usando el protocolo HTTP.

En la Figura **1.1** se muestra un ejemplo de cómo se realizará la comunicación entre el prototipo y el SAEw.

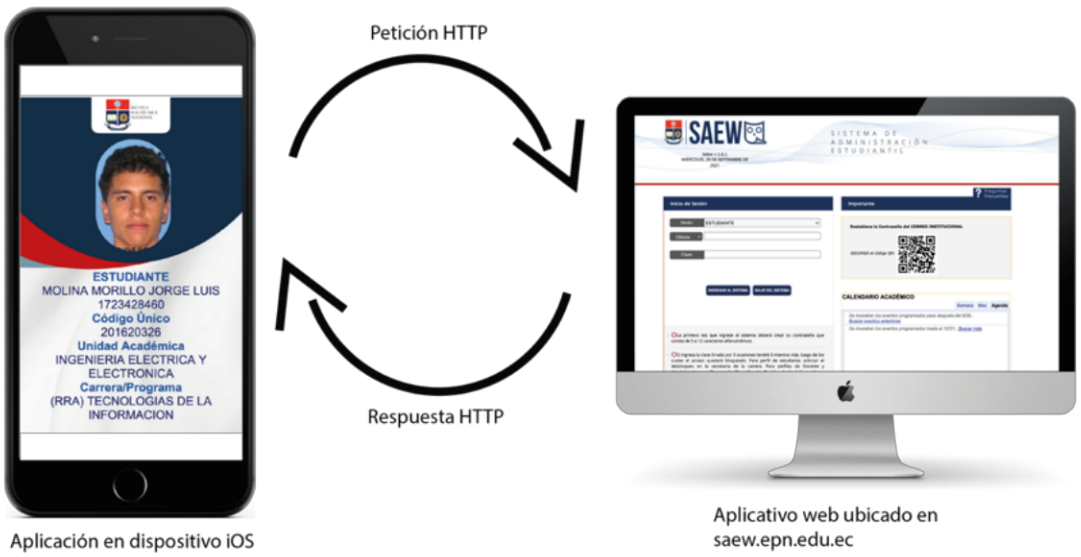


Figura 1.1. Ejemplo de comunicación entre el prototipo y el SAEw

Para el desarrollo del prototipo se empleará el patrón MVVM, el funcionamiento de este se puede evidenciar en la Figura **1.2**.

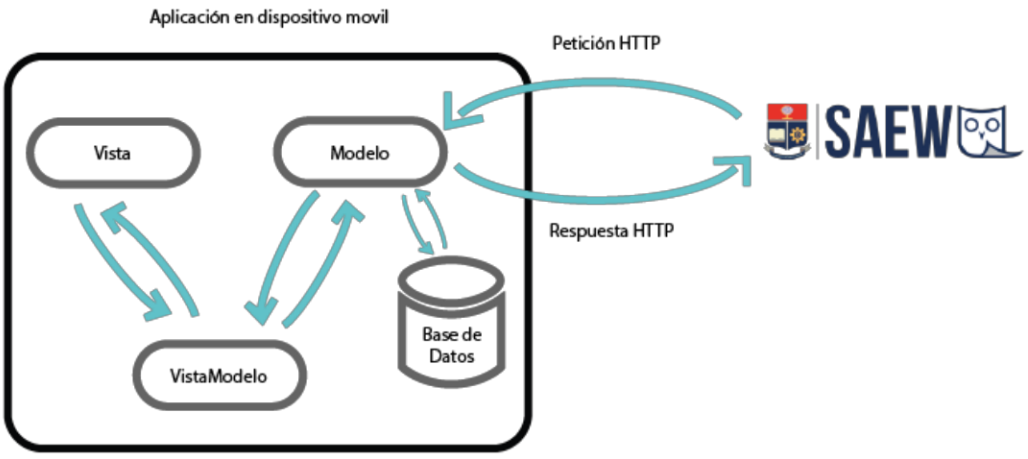


Figura 1.2. Patrón MVVM a implementar en el dispositivo móvil

El prototipo contará con al menos las siguientes vistas: vista de inicio de sesión y vista de presentación de carnet.

- Vista de inicio de sesión: Permitirá que el usuario pueda ingresar mediante las credenciales del SAEw, como la cédula, contraseña y la carrera que se encuentra actualmente cursando.
- Vista de presentación de carnet: En esta se mostrará el carnet estudiantil.

El presente Trabajo de Integración Curricular se desarrollará con base en la metodología Kanban.

Para la validación del prototipo se pedirá que 5 estudiantes de la EPN prueben la aplicación y respondan a una encuesta.

Este trabajo tendrá un producto final demostrable.

1.4 MARCO TEÓRICO

1.4.1 WEB SCRAPING

Existen dos formas para la extracción de datos en la web: la primera, mediante el uso de una API expuesta por el servidor; y la segunda, a través del *web scraping*.

El *web scraping* es una técnica para extracción de datos que se encuentran en la web, los datos obtenidos pueden ser procesados y almacenados para los usos que se deseen. El

web scraping típicamente suele aplicarse a recolección masiva de datos. Se define que existen dos etapas, la primera es la recolección de los recursos web y la segunda es la extracción de la información requerida [1].

El proceso empieza con la generación y envío de la solicitud, que podría enviarse mediante un método de HTTP GET o POST. Al recibir la solicitud el servidor web procesará la misma y responderá con un recurso que podría ser un archivo multimedia, un archivo en formato HTML o en otra extensión como la empleada en servidores de aplicaciones desarrollados con ASP.NET como ASPX. Al obtener los recursos, la aplicación debe procesar la información y la almacenará de acuerdo con sus necesidades [1].

Al realizar peticiones mediante los métodos GET o POST de HTTP se debe tomar en cuenta que estos pedidos pueden requerir el uso de *cookies*, puesto que estas sirven para mantener el estado de sesión, considerando que HTTP es un protocolo sin estado. Las *cookies* se envían como parte del encabezado del protocolo HTTP [2].

Para el *web scraping* es de suma importancia conocer sobre las *cookies* y cómo estas se incluyen en la cabecera del protocolo HTTP. El servidor web crea as *cookies* en respuesta a un pedido inicial originado por un cliente, en la Figura **1.3** se aprecia un diagrama que indica el funcionamiento de las *cookies* considerando las diferentes interacciones entre cliente y servidor [2].

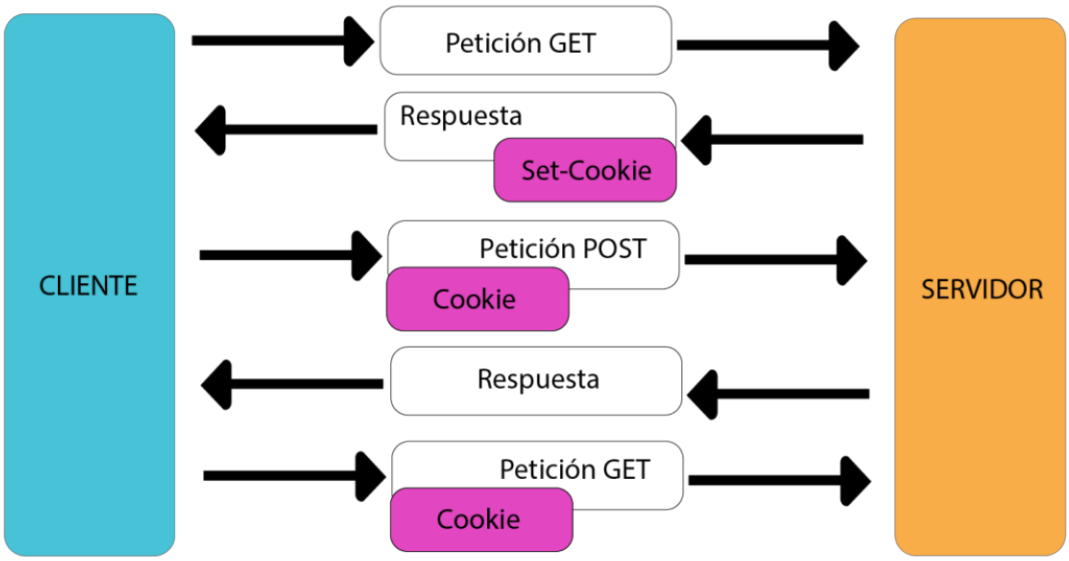


Figura 1.3. Diagrama de funcionamiento de *cookies*

En el protocolo HTTP se tienen nueve métodos de solicitud.

Sin embargo y dado que la mayoría de los servidores web solo emplean dos para la obtención de información, se detallará información sobre los métodos POST y GET [3].

El método GET de HTTP permite solicitar un recurso específico, el cuerpo del protocolo no incluye datos adicionales [4].

El método POST de HTTP se emplea para enviar datos originados a partir de un formulario HTML, tomando en cuenta atributos como `<input>` y `<button>` [5].

Usualmente, cuando se envían datos se debe tomar en cuenta que cada dato está separado mediante el carácter `&`, además el dato está compuesto por un campo y su valor unidos mediante el carácter `=`. A más de esto cada campo y valor debe ser codificado usando códigos definidos de manera porcentual para los caracteres no alfanuméricos. Por ejemplo, en el caso de que se desee enviar dos datos: el primero, con el campo `campo 1` y valor `valor á`; el segundo, con el campo `campo2` y valor `valor$b`, se deberá codificar la cadena así: `campo+1=valor+%C3%A1&campo2=valor%24b` [5].

1.4.2 LENGUAJE SWIFT

Swift es un lenguaje de código abierto propio de Apple, por lo que garantiza mayor compatibilidad frente a otros lenguajes, además dispone de características de seguridad y fluidez. En esta sección se presentará una guía básica del uso de este lenguaje [6].

1.4.2.1 Declaración de constantes o variables

Para declarar variables se usa la palabra reservada `var`, mientras que para constantes se emplea `let` Ambas opciones permiten especificar el tipo de dato que contendrá mediante el uso del caracter `:`, los tipos soportados son: `Int` para enteros, `Double` y `Float` para decimales, `Bool` para booleanos y `String` para cadenas de texto. En la Figura **1.4** se presenta un ejemplo de la declaración de diferentes tipos de variables y constantes [7].

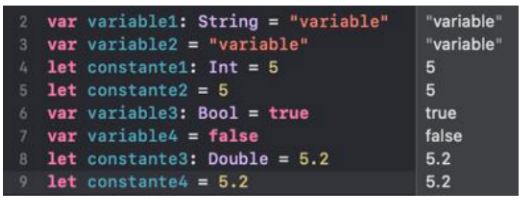


Figura 1.4. Declaración de variables y constantes

1.4.2.2 Arreglos

Un arreglo es un conjunto de valores ordenados mediante índices, los índices son números enteros secuenciales que parten desde cero. En la Figura **1.5** se muestra la declaración de arreglos para tipos de datos enteros y cadenas de textos. Para poder agregar elementos a un aregllo se puede usar el método `append` [8].

Figura 1.5. Declaración de arreglos

1.4.2.3 Diccionarios

Un Diccionario es una asociación de dos elementos, una llave y un valor. Cada valor debe ser asociado a una única llave y no existe un orden especifico de la ubicación de los elementos en el almacenamiento. En la Figura **1.6** se presenta un ejemplo de declaraciones para diccionarios [8].

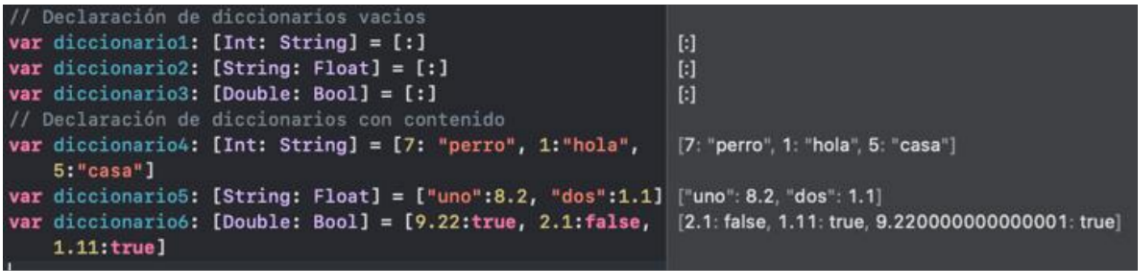


Figura 1.6. Ejemplo de declaración de diccionarios

En la Figura **1.7** se muestra un ejemplo del código que permite agregar valores a un diccionario, en el cual su llave es de tipo `Double` mientras que el valor es de tipo `String`.

Figura 1.7. Ejemplo de agregar valores a un diccionario

1.4.2.4 Funciones

Las funciones en Swift son extremadamente flexibles, para declararlas se emplea la palabra reservada `func`. Las funciones pueden tener como parámetros de entrada múltiples datos, un solo dato o ningún dato, lo mismo sucede con los datos de retorno. En la Figura **1.8** se observa ejemplos debidamente comentados de declaraciones de funciones [9].

```swift
//Nombre: funcion1
//Parametros de entrada: array de enteros
//Parametros de salida: entero
func funcion1(array: [Int]) -> Int {
    return 1
}

//Nombre: funcion2
//Parametros de entrada: cadena de texto y Double
//Parametros de salida: entero y boleano
func funcion2(para1: String, para2: Double) -> (val1:
    Int, val2: Bool) {
    return (1, true)
}

//Nombre: funcion3
//Parametro de entrada: sin parametros
//Parametros de salida: sin parametros
func funcion3(){

}
```

Figura 1.8. Ejemplos de declaraciones de funciones

Para llamar a una función se usa el código mostrado en la Figura **1.9**, en este ejemplo se llama a la `funcion2` presentada en la Figura **1.8**.

```swift
var hola = funcion2(para1: "cadena", para2: 3.3)
hola.val1
hola.val2
```

```
(val1 1, val2 true)
1
true
```

Figura 1.9. Ejemplo de llamado de una función

1.4.2.5 Estructuras y clases

Las estructura y clases son muy similares, pero las clases ofrecen funcionalidades adicionales como es el caso de la herencia. Para definir una clase se usa la palabra reservada `class`, mientras que para estructura se emplea `struct`. En la Figura **1.10** se observa la definición de una clase y una estructura [10].

```swift
class clase1{
    var variable1: Int = 0
    var variable2: String = ""
}

struct estructura1{
    var variable1: Int = 0
    var variable2: String = ""
}
```

Figura 1.10. Ejemplo de definición de clase y estructura

1.4.2.6 API para comunicación en la red

Para la comunicación mediante la red, Swift proporciona una API que funciona de forma asíncrona en la transmisión de datos. Para usar esta API es necesario importar la librería `Foundation` y usar la clase llamada `URLSession`, en el caso de que se necesite compartir información como *cookies* se puede usar un singleton[1] llamado `shared`. Para realizar una tarea de comunicación de datos se necesita llamar a la función `dataTask`, la cual retorna el contenido de un pedido que fue especificado mediante una URL, la cual debe ser construida usando la estructura de tipo `URLRequest` [11].

Para inicializar una estructura de tipo `URLRequest` se usa una estructura de tipo `URL`; también se debe definir el tipo de petición, sea GET o POST mediante la propiedad `httpMethod` y en el caso de POST los datos se incluyen en la propiedad `httpBody`. En la Figura **1.11** se muestran ejemplos del uso de estructuras de tipo `URLRequest` [12].

```
82  import Foundation
83  // Creacion de un URLRequest para GET
84  let url1 = URL(string: "https:/saew.epn.edu.ec/")
85  var pedidoGet = URLRequest(url: url1!)
86  pedidoGet.httpMethod = "GET"
87  // Creacion de un URLRequest para POST
88  let url2 = URL(string: "https:/saew.epn.edu.ec/")
89  var pedidoPost = URLRequest(url: url2!)
90  pedidoPost.httpMethod = "POST"
91  let postString = "field1=value1&field2=value2"
92  pedidoPost.httpBody = postString.data(using:
        String.Encoding.utf8)
```

Figura 1.11. Ejemplos de uso de URLRequest

1.4.3 LENGUAJE SWIFTUI

El lenguaje SwiftUI se basa en dos protocolos, el protocolo App y el protocolo View, en esta sección se describirá de forma breve el uso de estos [13].

Protocolo APP: Permite definir la estructura y el comportamiento de una aplicación. Para crear una aplicación se debe declarar una estructura que cumpla con el protocolo `App`, la estructura requiere de un elemento `body` que establezca el contenido de esta. Para definir el inicio de la aplicación, se usa el atributo `@main`, Adicionalmente para representar los aspectos de la interfaz de usuario se usa el protocolo `Scene`. En la Figura **1.12** se muestra

[1] Singleton: Clase que devuelve la misma instancia, sin importar cuantas veces haya sido llamada.

un ejemplo del uso del protocolo `App` en la que existe una aplicación llamada `MyApp` y esta llama a la vista `ContentView` [14].

```swift
import SwiftUI

@main
struct MyApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Figura 1.12. Ejemplo de uso de protocolo `App`

Protocolo View: Establece las características que un tipo debe cumplir para poder ser una vista. En la Figura **1.13** se muestra un ejemplo del uso de una estructura llamada `ContentView` la cual muestra una instancia de tipo `Text` [15].

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("¡Hola Mundo!")
    }
}
```

Figura 1.13. Ejemplo de uso del protocolo `View`

1.4.3.1 Arquitectura MVVM

MVVM fue desarrollado por John Gossman en 2005, en la Figura **1.14** se aprecia el diagrama de funcionamiento de esta arquitectura, en esta arquitectura la Vista tiene un enlace con VistaModelo donde se encuentran las propiedades y esta obtiene del Modelo los objetos de datos [16].

Figura 1.14. Diagrama de la arquitectura MVVM

1.4.4 METODOLOGÍA KANBAN

La idea esencial de la metodología Kanban es el trabajo en el proceso, debido a esto entre sus características esta la mutabilidad de las tareas a realizar que se muestran mediante tarjetas gráficas. Esta metodología puede ser implementada en el desarrollo de procesos ya existentes. Para poder usar esta metodología es necesario dividir el proceso en etapas las cuales corresponden a las columnas y el uso de las tarjetas gráficas que contienen información esencial de los trabajos que se realizan. En la Figura **1.15** se muestra un ejemplo del uso de un tablero Kanban mediante el software *Airtable*[2] [17].

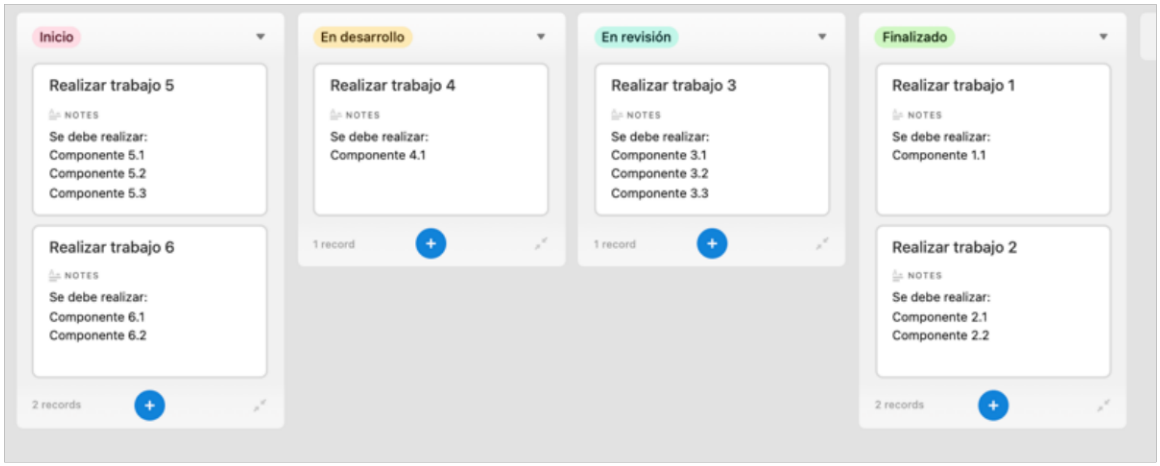


Figura 1.15. Ejemplo de tablero Kanban

[2] *Airtable*: Software desarrollado para almacenar bases de datos.

2 METODOLOGÍA

Para el desarrollo del prototipo se usará la metodología ágil Kanban, una de las ventajas de esta metodología es que permite que el número y el contenido de las tareas varíe, esto es de suma importancia debido a que el prototipo a desarrollar se generará a partir de un trabajo experimental.

Este capítulo se encuentra organizado con dos secciones: la primera de diseño, en la cual se detalla la lógica del prototipo generada a partir de un análisis de los recursos web obtenidos; la segunda de implementación, con base en la información generada en el diseño se desarrolla una aplicación para iOS.

Como parte del diseño se genera el tablero Kanban a partir de tareas preestablecidas y creadas a partir de los requerimientos funcionales originados por el uso de un método inductivo a partir del análisis del SAEw.

También se realizaron pruebas para analizar que la aplicación funcione de forma adecuada.

2.1 DISEÑO

Esta sección presenta un resumen del proceso seguido para la obtención del carnet desde el SAEw, además se plantean los requerimientos funcionales y los no funcionales del prototipo.

2.1.1 PROCESO DE OBTENCIÓN DEL CARNET DEL SAEW

En esta sección se estudiará el funcionamiento del SAEw para presentar el carnet estudiantil. En el ANEXO I se incluye el enlace a un video con el procedimiento realizado en Safari para iOS, con una duración de 1 minuto y 34 segundos.

2.1.1.1 Procedimiento para la obtención de carnet mediante el navegador web

La obtención del carnet estudiantil disponbile en el SAEw, mediante el navegador web, consta de las siguientes etapas, que a su vez se dividen en diferentes pasos:

1. Etapa de inicio de sesión en el SAEw
2. Etapa de navegación a través de los módulos del SAEw

3. Etapa de obtención del carnet

2.1.1.1.1 Etapa de inicio de sesión en el SAEw

Paso 1.— Para poder iniciar sesión en el SAEw, desde un explorador web, se realiza una petición GET a la URL `https://saew.epn.edu.ec/login.aspx`. La respuesta de esta petición se la observa en la Figura **2.1**.

Figura 2.1. Página de inicio de sesión del SAEw

Paso 2. — Para poder ingresar al SAEw, el usuario debe cambiar el modo predeterminado `ADMINISTRATIVO` a `ESTUDIANTE`. Cuando se realiza el cambio indicado, la página realiza un *postback*[3], es decir el navegador web envío una solicitud tipo POST al momento de cambiar el campo Modo de su valor por defecto al de `ESTUDIANTE`. La respuesta de esta petición se presenta en la Figura **2.2**.

Figura 2.2. Respuesta ante cambio de modo

[3] *Postback*: es la realización de un pedido POST al mismo recurso donde se encuentra el formulario.

Paso 3. — Una vez establecido el modo `ESTUDIANTE`, el usuario ingresa el número de cédula en el campo *`Cédula`*: y la Clave en el campo *`Clave`*:, al dar clic en el botón `INGRESAR AL SISTEMA`, el explorador web genera una acción de tipo *submit*[4], es decir envía una solicitud de tipo POST con el contenido del formulario al servidor web. En el caso de que las credenciales de acceso hayan sido correctas, la respuesta de esta petición se presenta en la Figura **2.3**, caso contrario si el usuario no existe se mostrará el mensaje "`No existe usuario`" y si en cambio la clave es incorrecta se mostrara "`Clave no Válida`".

Figura 2.3. Respuesta ante la validación de las credenciales

Paso 4. — Si no existe un mensaje de error, el usuario debe presionar en el botón `ACEPTAR`, con lo cual se genera una acción *submit* para enviar mediante el método POST los datos del formulario de inicio de sesión. En la Figura **2.4** se observa la respuesta en caso de que los datos de inicio de sesión hayan sido validados.

Paso 5. — En el formulario presentado en la Figura **2.4**, existen opciones que se pueden seleccionar, estas son *Carrera:* y *Periodo:*. Se dejan las opciones preseleccionadas debido a que de escogerse otra carrera se generaría un paso adicional debido al *postback* asociado al control; y si se cambia el periodo, el carnet que se obtendrá no será el actual. Si no se realiza ningún cambio, y se da clic en el botón `INGRESAR AL SISTEMA`, el explorador web realizará un *submit* con lo cual se generará una petición POST con los datos establecidos en el formulario. En la Figura **2.5** se encuentra la respuesta ante el ingreso al sistema por parte de un usuario.

[4] *Submit:* Acción de enviar un formulario, mediante el método de POST.

Figura 2.4. Respuesta ante inicio de sesión válido

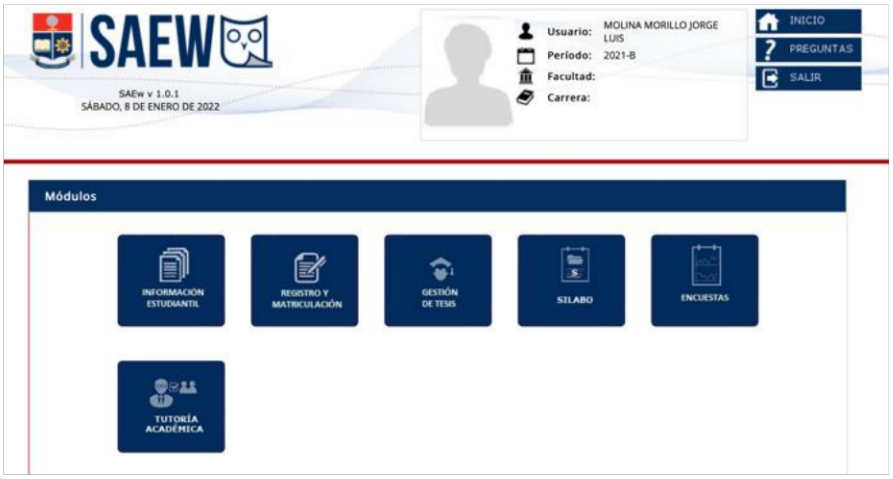


Figura 2.5. Respuesta ante al ingreso al sistema

2.1.1.1.2 Etapa de navegación a través de los módulos del SAEw

Una vez finalizada la etapa de inicio de sesión, en la que el usuario mediante el navegador web interactuó con la página `login.aspx` para poder acceder al recurso `Modulos.aspx`, el usuario debe navegar entre los módulos del SAEW para encontrar el recurso del carnet estudiantil y su URL. El esquema de navegación se presenta en la Figura **2.6**, en la cual se observa que el usuario primero presiona en el módulo Información Estudiantil, después en el menú Información General y por último en la opción Carnet Estudiantil.

Figura 2.6. Diagrama de Navegación

Al finalizar estas tareas, en la barra de navegación del navegador web se puede observar que la URL indica: `https://saew.epn.edu.ec/SAEINF/CarneEstudiantil.aspx`, que corresponde a un formulario web que permite obtener el carnet estudiantil y el recurso con el que se interactúa es `CarnetEstudiantil.aspx`. Si el usuario desea ingresar al recurso del Carnet Estudiantil puede digitar la URL indicada en el navegador web, después de haber iniciado sesión.

2.1.1.1.3 Etapa de obtención del carnet

Paso 6.— Una vez iniciada sesión en el SAEw, en la barra de navegación se ingresa a la URL `https://saew.epn.edu.ec/SAEINF/CarneEstudiantil.aspx` o se navega a través de los módulos. Ambas opciones implican que desde el navegador web se realice una petición GET, con lo que se obtendrá como respuesta un formulario web que solo presenta un cuadro de selección. En la Figura **2.7** se aprecia el formulario web que permite obtener el carnet.

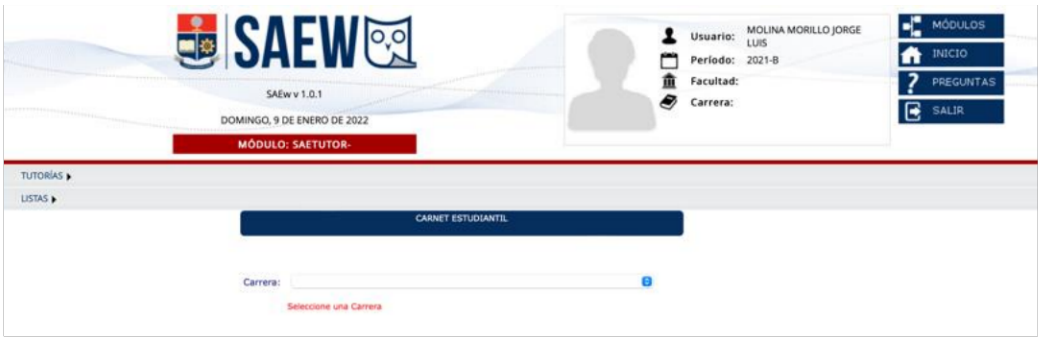


Figura 2.7. Formulario para obtención de carnet

Paso 7. — En el formulario web se debe seleccionar la Carrera en la que el estudiante se encuentra matriculado, con lo cual el navegador web realizará una petición POST, con lo

que se obtendrá como respuesta un formulario web con el carnet estudiantil. En la Figura **2.8** se aprecia el resultado de este paso.

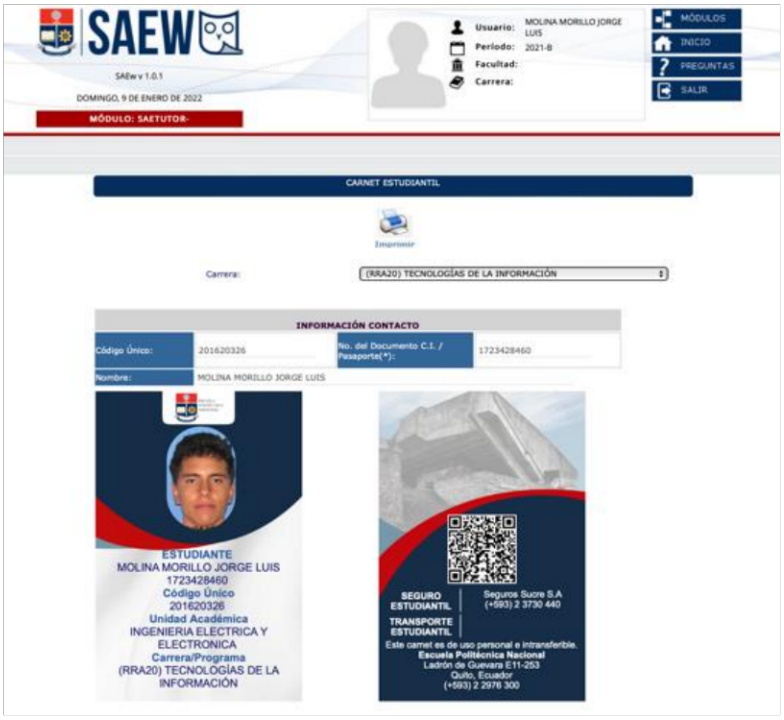


Figura 2.8. Formulario web con el Carnet

2.1.1.2 Análisis técnico de los pasos

Los pasos de la sección 2.1.1.1 serán referenciados en esta sección, con los nombres que se enumeran a continuación:

1. Iniciar comunicación con el SAEw
2. Cambiar a modo de estudiante
3. Validar datos de usuario
4. Aceptar validación
5. Ingresar al sistema
6. Ingresar al aplicativo Carnet Estudiantil
7. Mostrar Carnet Estudiantil

Para que el análisis sea adecuado, el navegador web no debe tener datos almacenados del SAEw, además se requiere de una herramienta que permita inspeccionar la página web obtenida.

Mediante la navegación privada y el *Web Inspector* del navegador Safari se describen los resultados obtenidos del análisis.

2.1.1.2.1 Iniciar comunicación con el SAEw

Se verifica que el navegador web solo haya realizado una petición GET al recurso `Login.aspx`. En la Figura **2.9** se presenta el resultado del *Web Inspector,* en el cual, se seleccionan las opciones *Network*, *Document* y `Login.aspx`. se aprecia en la sección *Request* el pedido GET remitido, y en *Response*, la respuesta obtenida.

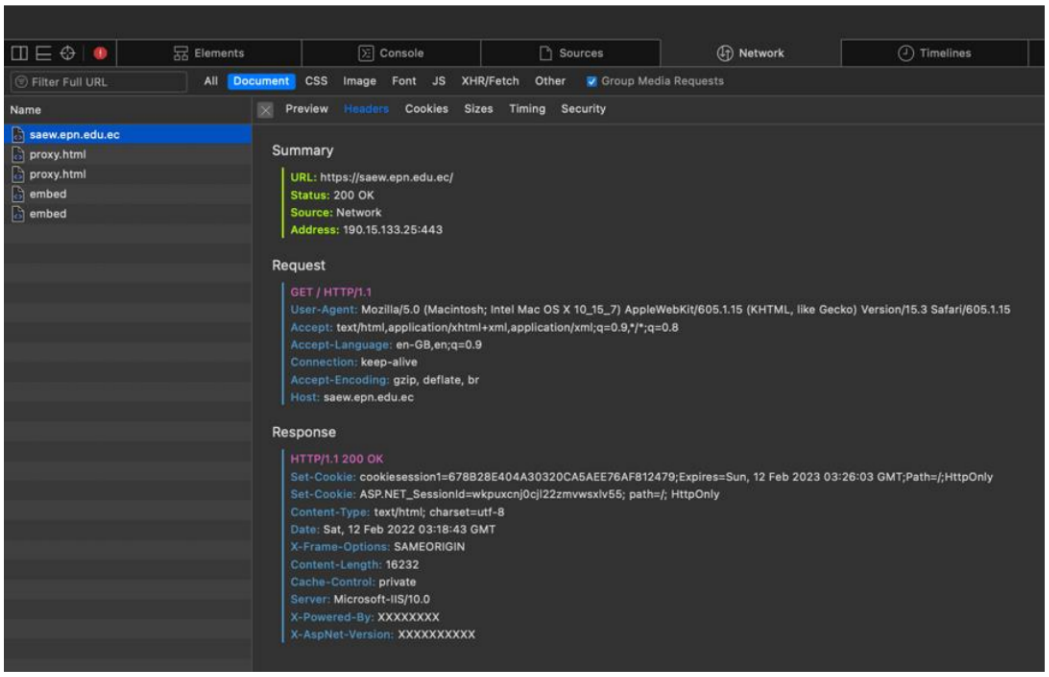


Figura 2.9. Inspector Web del paso 1, iniciar comunicación con el SAEw

2.1.1.2.2 Cambiar a modo de estudiante

Este paso se produce debido al cambio de modo, lo cual provoca que se realice un *postback*. Al realizar el análisis mediante *Inspector Web* e inspeccionar el control que provoca este *postback*, se aprecia que, existe un fragmento de código HTML que indica que existe código JavaScript asociado a este control, el mismo que se invoca cuando se produce el evento `onChange`, es decir cuando se escoge otra opción diferente a la que presenta el control, como se aprecia en la Figura **2.10**, en la cual se ha marcado en azul el código del control que permite cambiar el modo.

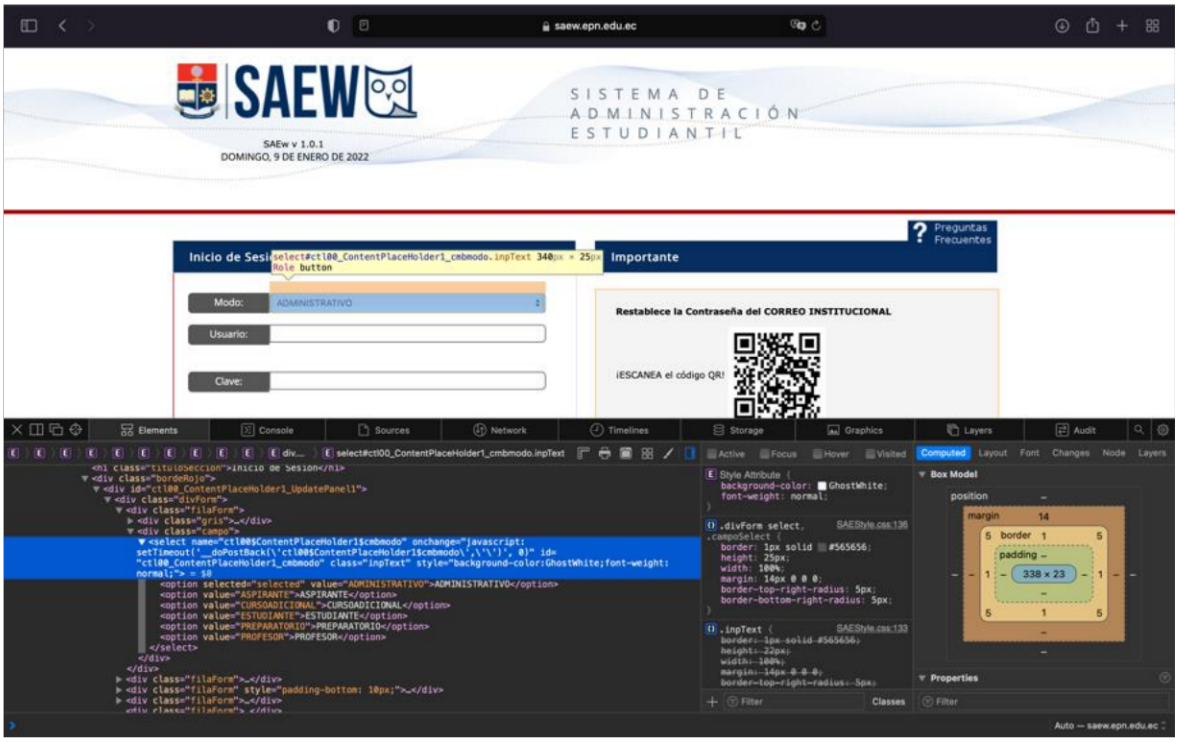


Figura 2.10. Código HTML asociado al control que permite cambiar el modo en el SAEw

En la Figura **2.10** se observa que la función denominada `__doPostBack()` tiene 2 parámetros de entrada, el uno con valor `ctl100$ContentPlaceHolder1$cmbmodo` y el otro vacío. Es necesario entender el funcionamiento de `__doPostBack()` para saber los datos que se enviarán en la solicitud POST. En la Figura **2.11** se aprecia la función `__doPostBack()` visualizada en el inspector web de Safari, mediante esta herramienta se puede analizar que el objetivo de la función es asegurar que cuando se ejecute el evento `OnSubmit` se actualicen dos valores del formulario: `__EVENTTARGET` y `__EVENTARGUMENT`, para posteriormente hacer un *submit*, con lo cual se enviará en la solicitud POST el formulario web, incluyendo estos dos valores indicados.

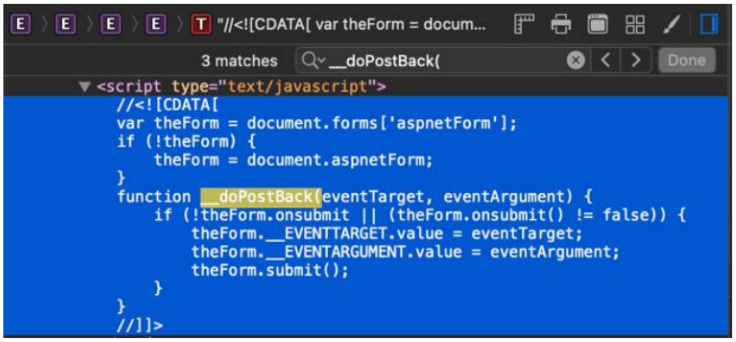


Figura 2.11. Contenido de la función `__doPostBack()`

Al analizar los valores `input` y dado que entre estos están `__EVENTTARGET` y `__EVENTARGUMENT` se procede a conocer datos adicionales que se envían en la solicitud, para esto se realiza el paso 2, es decir en el explorador web se cambia el a modo a estudiante, y con el inspector web se observan los datos enviados en *Request Data*, como se evidencia en la Figura **2.12**.

Figura 2.12. *Request Data* del paso 2, cambiar a modo estudiante

En la Tabla **2.1** se aprecian las variables enviadas en la petición POST con la cual se realizar el cambio de modo.

En la primera columna de la Tabla **2.1** se muestra el nombre de las variables, en la segunda se incluye una descripción de las mismas y en la tercera se clasifica a la variable en uno de los siguientes tres tipos:

a) De libre ingreso, el contenido de estas depende totalmente del usuario;

b) De selección, el usuario solo podrá escoger el valor de la variable entre una serie de opciones; y;

c) Ocultas, cuyos valores son designados por el servidor.

Tabla 2.1. Datos del método POST en el paso 2 al cambiar a modo estudiante

Variable	Descripción	Tipo
__EVENTTARGET	Nombre de la variable que se modificó, en este caso ctl00$ContentPlaceHolder1$cmbmodo.	De selección
__VIEWSTATE	El __VIEWSTATE se obtiene como respuesta del servidor al paso 1, al iniciar la comunicación con el SAEw, es input y hidden.	Oculta
__VIEWSTATEGENERATOR	El __VIEWSTATEGENERATOR se obtiene como respuesta del servidor al paso 1, al iniciar comunicación con el SAEw, es input y hidden.	Oculta
__EVENTVALIDATION	El __EVENTVALIDATION se obtiene como respuesta del servidor al paso 1, al iniciar comunicación con el SAEw, es input y hidden.	Oculta
ctl00$ContentPlaceHolder1$cmbmodo	Es el modo escogido, en este caso es ESTUDIANTE	De selección

2.1.1.2.3 Validar datos de usuario

Para realizar este paso se debe completar los campo de *Cédula:*, *Clave:*, y dar clic en el botón INGRESAR AL SISTEMA, el resultado se visualiza en *Request Data* presentado en la Figura **2.13**. En la Tabla **2.2** se aprecian las variables empleadas en el paso 3.

Figura 2.13. *Request Data* del paso 3, validar datos de usuario

2.1.1.2.4 Aceptar validación

Luego de dar clic en el botón `ACEPTAR` se obtiene la información del paso 4, cuyo resultado se presenta en la Figura **2.14**, en la cual se visualiza el *Request Data* y en la **Tabla 2.3** se aprecian las variables empleadas en este paso.

Tabla 2.2. Datos del método POST en el paso 3, validar datos de usuario

Variable	Descripción	Tipo
`__VIEWSTATE`	El `__VIEWSTATE` es obtiene como respuesta del servidor al paso 2, al cambiar a modo de estudiante, es `input` y `hidden`.	Oculta
`__VIEWSTATEGENERATOR`	El `__VIEWSTATEGENERATOR` se obtiene como respuesta del servidor al paso 2, al cambiar a modo de estudiante, es `input` y `hidden`.	Oculta
`__EVENTVALIDATION`	El `__EVENTVALIDATION` se obtiene como respuesta del servidor al paso 2, al cambiar a modo de estudiante, es `input` y `hidden`.	Oculta
`ctl00$ContentPlaceHolder1$cmbmodo`	Es el modo escogido, en este caso es `ESTUDIANTE`.	De selección
`ctl00$ContentPlaceHolder1$cmbDocumento`	Es el documento de identificacion escogido, en este caso es `Cédula:`.	De selección
`ctl00$ContentPlaceHolder1$UserName`	Es la cédula digitada por el usuario, en este caso es `1723428460`.	De libre ingreso
`ctl00$ContentPlaceHolder1$Password`	Es la clave digitada por el usuario , en este caso es `Password123`.	De libre ingreso
`ctl00$ContentPlaceHolder1$LoginButton`	Es el contenido del texto del botón `INGRESAR AL SISTEMA`, y en este caso es `Ingresar al Sistema`	Oculta

Figura 2.14. *Request Data* del paso 4, aceptar validación

2.1.1.2.5 Ingresar al sistema

Al dar clic en el botón INGRESAR AL SISTEMA que se muestra en la Figura **2.4**, se realiza el análisis mediante el *Inspector Web* y se visualiza el *Request Data* con lo indicado en la Figura **2.15**. Es importante notar que la forma en la que el request data se muestra es diferente, esto es debido a que la repuesta fue redireccionada al recurso `Modulos.aspx`, al suceder esto el inspector web muestra el *Request Data* de forma codificada porcentual usando el carácter `&` para separar datos y el carácter `=` para asignar un valor al campo

Tabla 2.3. Datos del método POST en el paso 4, aceptar validación

Variable	Descripción	Tipo
`__VIEWSTATE`	El `__VIEWSTATE` se obtiene como respuesta del servidor al paso 3, al validar datos de usuario, es `input` y `hidden`.	Oculta
`__VIEWSTATEGENERATOR`	El `__VIEWSTATEGENERATOR` se obtiene como respuesta del servidor al paso 3, al validar datos de usuario, es `input` y `hidden`.	Oculta
`__EVENTVALIDATION`	El `__EVENTVALIDATION` se obtiene como respuesta del servidor al paso 3, al validar datos de usuario, es `input` y `hidden`.	Oculta
`ctl00$ContentPlaceHolder1$ cmbmodo`	Es el modo escogido, en este caso es `ESTUDIANTE`.	De selección
`ctl00$ContentPlaceHolder1$ cmbDocumento`	Es el documento de identificación escogido, en este caso es `Cédula:`.	De selección
`ctl00$ContentPlaceHolder1$ UserName`	Es la cédula digitada por el usuario, en este caso es `1723428460`.	De libre ingreso
`ctl00$ContentPlaceHolder1$ Password`	Es la clave digitada por el usuario , en este caso es `Password123`.	De libre ingreso
`ctl00$ContentPlaceHolder1$ btnValidar`	Es el contenido del texto del botón `ACEPTAR`.	Oculta

El *Request Data* codificado mostrada en la Figura **2.15** necesita ser decodificado, para asi conocer que variables del recuso `Login.aspx` que se estan enviando junto a su correspondiente valor. En la **Tabla 2.4** se muestran los datos después de haber sido decodificados.

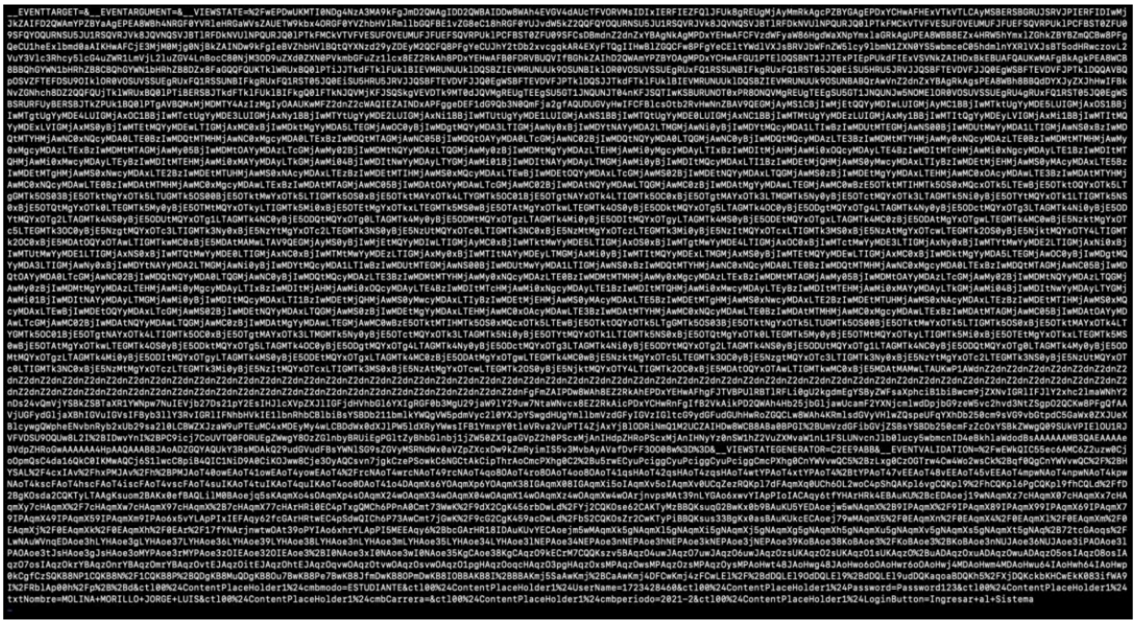


Figura 2.15. *Inspector Web* en el paso 5, ingresar al sistema

Tabla 2.4. Datos del método POST en el paso 5, ingresar al sistema

Variable	Descripción	Tipo
`__VIEWSTATE`	El `__VIEWSTATE` se obtiene como respuesta del servidor al paso 4, al aceptar la validación, es `input` y `hidden`.	Oculta
`__VIEWSTATEGENERATOR`	El `__VIEWSTATEGENERATOR` se obtiene como respuesta del servidor al paso 4, al aceptar la validación, es `input` y `hidden`.	Oculta
`__EVENTVALIDATION`	El `__EVENTVALIDATION` se obtiene como respuesta del servidor al paso 4, al aceptar la validación, es `input` y `hidden`.	Oculta
`ctl00$ContentPlaceHolder1$cmbmodo`	Es el modo escogido, en este caso es `ESTUDIANTE`.	De selección
`ctl00$ContentPlaceHolder1$UserName`	Es la cédula digitada por el usuario, en este caso es `1723428460`.	De libre ingreso
`ctl00$ContentPlaceHolder1$Password`	Es la clave digitada por el usuario, en este caso es `Password123`.	De libre ingreso
`ctl00$ContentPlaceHolder1$txtNombre`	Es el nombre completo del usuario, en este caso es `MOLINA MORILLO JORGE LUIS`.	Oculta
`ctl00$ContentPlaceHolder1$cmbCarrera`	Presenta una carrera de todas las que el servidor informa que el usuario ha cursado, en este caso es vacío “”.	De selección
`ctl00$ContentPlaceHolder1$cmbperiodo`	Presenta el periodo académico de una lista proporcionada por el servidor, en este caso es `2021-2`.	De selección
`ctl00$ContentPlaceHolder1$LoginButton`	Es el contenido de del texto del botón `INGRESAR AL SISTEMA`.	Oculta

2.1.1.2.6 *Ingresar al aplicativo Carnet Estudiantil*

Una vez iniciada la sesión en el SAEw, se realiza una petición a la URL `https://saew.epn.edu.ec/SAEINF/CarneEstudiantil.aspx`, en el *Inspector Web* se observa que se realiza una petición GET, como se evidencia en la Figura **2.16**. Al comparar el contenido de `cookie` con el contenido de `set-cookie` presentado en la Figura **2.9**, se evidencia que son iguales.

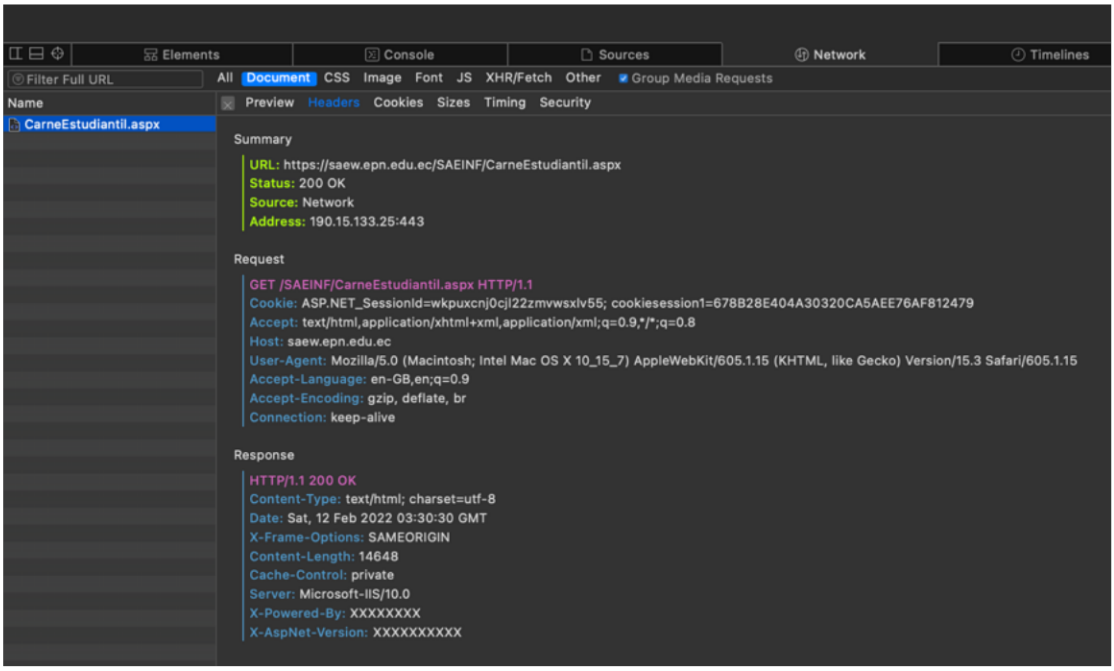


Figura 2.16. *Inspector Web* en el paso 6, ingresar al aplicativo Carnet Estudiantil

2.1.1.2.7 *Mostrar Carnet Estudiantil*

Para finalizar el procedimiento se escoge la carrera en la que se encuentra el estudiante matriculado, con el *Inspector Web* se obtiene el *Request Data* presentado en la Figura **2.17**.

Figura 2.17. *Request Data* del paso 7, mostrar Carnet Estudiantil

El contenido de los datos del método POST de la Figura **2.17** se lo resume en la Tabla **2.5**.

Tabla 2.5. Datos del método POST en el paso 7, mostrar Carnet Estudiantil

Variable	Descripción	Tipo
__EVENTTARGET	Nombre de la variable modificada, en este caso ctl00$ContentPlaceHolder1 $cmbCarrera.	De selección
__VIEWSTATE	El __VIEWSTATE se obtiene como respuesta del servidor al paso 6, al ingresar al Carnet Estudiantil, es input y hidden.	Oculta
__VIEWSTATEGENERATOR	El __VIEWSTATEGENERATOR se obtiene como respuesta del servidor al paso 6, al ingresar al Carnet Estudiantil, es input y hidden.	Oculta
__EVENTVALIDATION	El __EVENTVALIDATION se obtiene como respuesta del servidor al paso 6, al ingresar al Carnet Estudianti, es input y hidden.	Oculta
ctl00$ContentPlaceHolder1$ cmbCarrera	Es el id de la carrera escogida, en este caso es (RRA) TECNOLOGIAS DE LA INFORMACION con ID 232.	De selección

Mediante el Web Inspector se puede obtener el id de las variables HTML que son necesarias para el carnet, estas dependen del usuario que haya iniciado sesión. La información de estas variables se la puede observar en la Tabla **2.6**.

Tabla 2.6. Variables del Carnet Estudiantil

Variable Id	Descripción
ctl00_ContentPlace Holder1_imgFoto	Imagen en base 64 con formato JPEG que incluye la foto de la cedula del usuario que inició sesión.
ctl00_ContentPlace Holder1_lblNombreComp	Nombre completo del usuario que inició sesión.
ctl00_ContentPlace Holder1_lblCedulaEst	Número de cédula del usuario que inició sesión.
ctl00_ContentPlace Holder1_lblCodest	Código único del usuario.
ctl00_ContentPlace Holder1_lblFacultaEst	Facultad de la carrera seleccionada por el usuario.
ctl00_ContentPlace Holder1_lblCarreraEst	Carrera seleccionada por el usuario.
ctl00_ContentPlace Holder1_imgQRCode	Imagen en base 64 con formato JPEG de un código QR que contiene información del usuario que inició sesión.

2.1.2 REQUERIMIENTOS FUNCIONALES

Se encontró que para el funcionamiento del prototipo es necesario que existan los siguientes requerimientos funcionales:

1. Iniciar sesión en el prototipo
2. Almacenar datos en el dispositivo
3. Visualizar el carnet
4. Cerrar sesión

Mediante un dispositivo iOS con acceso a Internet, un estudiante puede acceder a los datos del carnet almacenado en el SAEw con las credenciales otorgadas por la EPN, para esto el prototipo deberá realizar los 7 pasos mostrados en la sección 2.1.1.2.

Una vez que el usuario haya iniciado sesión, el prototipo puede funcionar sin acceso a Internet, debido a que los datos del carnet se guardarán en la base de datos propia del dispositivo. Los datos que se almacenarán son los presentados en la Tabla **2.6**.

La interfaz de usuario debe ser similar a la parte anterior y posterior del carnet estudiantil que se muestra en la Figura **2.8**.

Al cerrar sesión, se borrarán los datos del carnet que se encuentran almacenados en el dispositivo.

2.1.3 REQUERIMIENTOS NO FUNCIONALES

Se establecieron los siguientes requerimientos no funcionales, debe ser construido con Swift y SwifttUI, debe basarse en el patrón MVVM, y debe controlar los errores ya sean por falla en la comunicación o debido al ingreso de credenciales erroneas.

2.1.4 TABLERO KANBAN

Mediante el uso de la versión gratuita del software Airtable, se construyó el tablero Kanban el cual consta de 4 etapas: inicio, desarrollo, revisión y finalizado. En la Figura **2.18** se muestra el tablero Kanban con las etapas finalizada hasta la elaboración de esta sección. En el ANEXO II se incluya el tablero Kanban con toda la información y se incluye la URL del tablero Kanban.

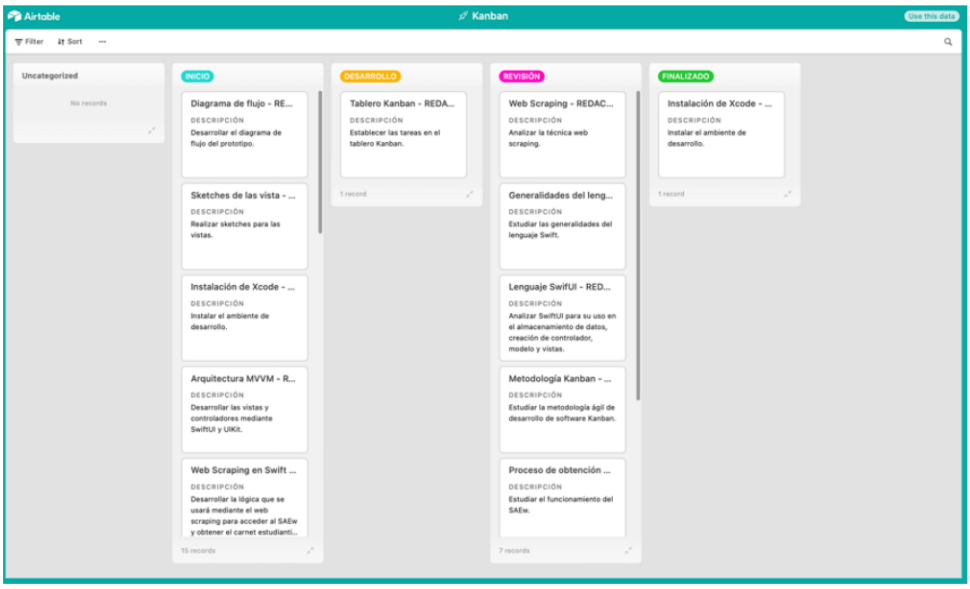


Figura 2.18. Tablero Kanban desarrollado en Airtable

2.1.5 DIAGRAMA DE FLUJO

Para el desarrollo del prototipo se generó el diagrama de flujo que se muestra en la Figura **2.19**.

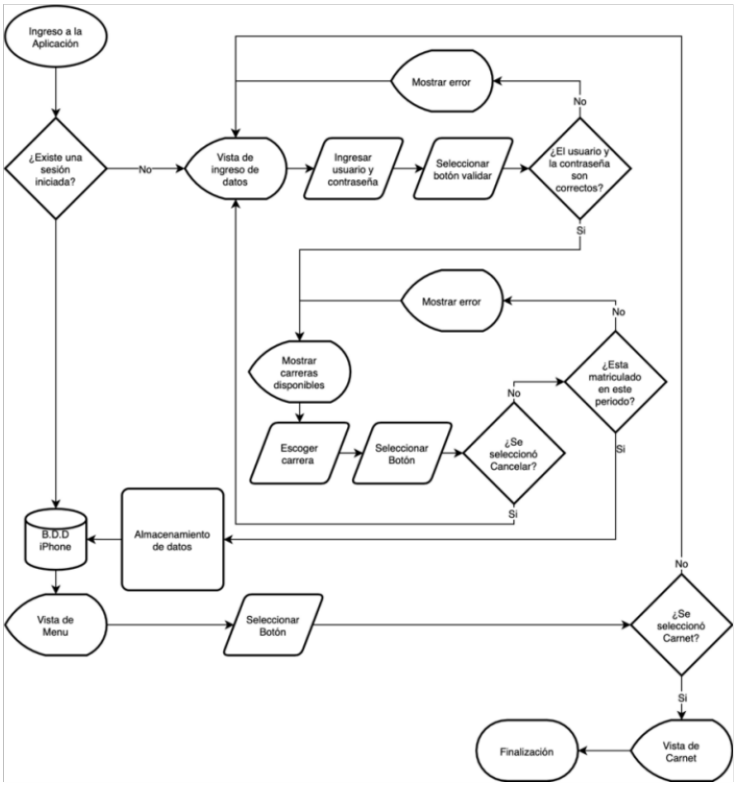


Figura 2.19. Diagrama de flujo de la aplicación

2.1.6 SKETCHES DE LAS VISTAS

En la Figura **2.20** se presenta el *sketch* que corresponde a la vista de ingreso de datos.

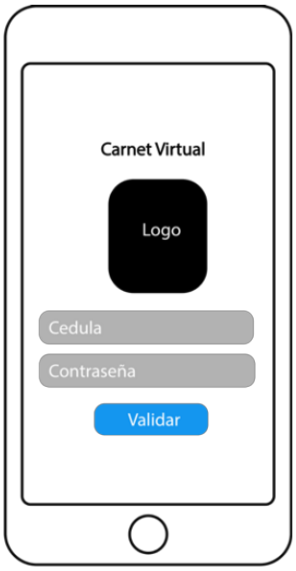


Figura 2.20. *Sketch* Ingreso de Datos 1

En la Figura **2.21** se presenta el *sketch* que corresponde a la vista de ingreso de datos con la opción de carrera y los botones *Ingresar* y *Cancelar*.

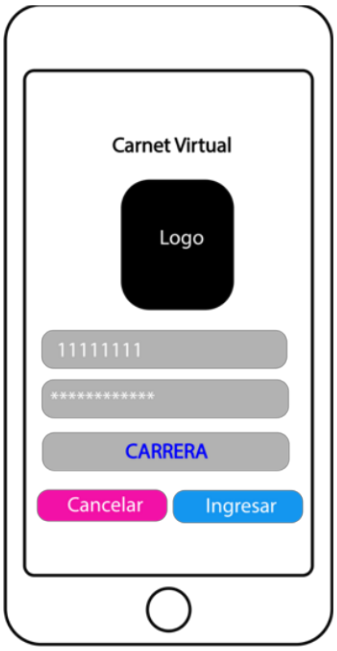


Figura 2.21. *Sketch* Ingreso de Datos 2

En la Figura **2.22** se presenta el *sketch* que corresponde a la vista de menú, mediante el cual el usuario puede ver el Carnet o Salir.

Figura 2.22. Sketch Menú

En la Figura **2.23** se presenta el *sketch* que corresponde a la vista frontal del carnet, con un diseño similar al que usa el SAEw.

Figura 2.23. *Sketch* vista frontal del carnet

En la Figura **2.24** se presenta el *sketch* que corresponde a la vista trasera del carnet, el cual es similar en diseño al que se encuentra en el SAEw.

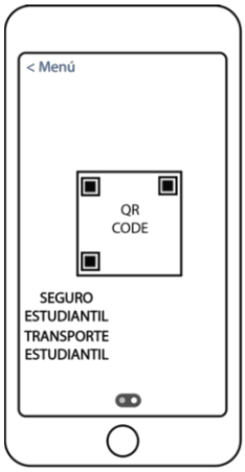


Figura 2.24. *Sketch* vista trasera del carnet

2.2 IMPLEMENTACIÓN

2.2.1 INSTALACIÓN DE XCODE

Para poder realizar una aplicación es necesario disponer de un dispositivo con el sistema operativo macOS, en el cual se instale Xcode. Xcode está disponible en la App *Store*[5]

En Xcode se creó un proyecto para desarrollar el prototipo, para esto se creó una cuenta de iCloud con las credenciales del correo electrónico institucional, como se observa en la pantalla de configuración presentada en la Figura **2.25**.

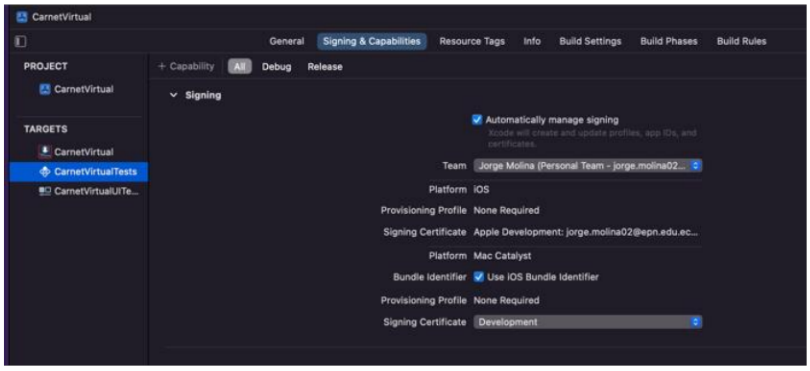


Figura 2.25. Creación de proyecto con credenciales institucionales

[5] *App Store:* Tienda oficial de aplicaciones para dispositvos Apple

2.2.2 WEB SCRAPING

El *web scraping* permitirá obtener los datos que fueron analizados en la sección 2.1.1.2. Tambien se debe tomar en cuenta que la comunicación entre el prototipo y el SAEw debe tomar en consideración las peticiones GET y POST adecuadas.

2.2.2.1 Datos Constantes

Los datos constantes cumplen con el siguiente criterio: el valor debe ser el mismo sin importar de qué estudiante se trate. En la Tabla **2.7** se aprecian las constantes del prototipo.

Tabla 2.7. Constantes

Constante	**Descripción**	**Valor**
`__EVENTTARGET`	El `__EVENTTARGET` del paso 2, al cambiar a modo de estudiante, requiere de la variable `ctl00$ContentPlaceHolder1$cmbmodo`.	`ctl00$ContentPlaceHolder1$cmbmodo`
`ctl00$ContentPlaceHolder1$cmbmodo`	Debido a que se ingresa como estudiante el valor debe ser `ESTUDIANTE`.	`ESTUDIANTE`
`ctl00$ContentPlaceHolder1$cmbDocumento`	Para ingresar la credencial es la cédula .	`Cédula:`
`ctl00$ContentPlaceHolder1$LoginButton`	El texto del botón debe ser: paso 3, validar datos de usuario, y paso 5, ingresar al sistema.	`Ingresar al Sistema`
`ctl00$ContentPlaceHolder1$btnValidar`	El texto del botón en el paso 4 es aceptar validación.	`ACEPTAR`
`__EVENTTARGET`	El `__EVENTTARGET` del paso 7 requiere de la variable `ctl00$ContentPlaceHolder1$cmbCarrera`.	`ctl00$ContentPlaceHolder1$cmbCarrera`

2.2.2.2 Datos Variables

Los datos variables son definidos por el usuario sin que el servidor imponga una restricción considerando un determinado número de opciones. En la Tabla **2.8** se observan los datos de las variables, su descripción y el nombre de la variable del código.

Tabla 2.8. Datos Variables

Dato variable HTML	Descripción	Nombre
ctl00$ContentPlace Holder1$UserName	Es la cédula del usuario	UserName
ctl00$ContentPlace Holder1$Password	Es la clave del usuario	Password

2.2.2.3 Datos Variables Ocultos y de Selección

El valor de estos datos es dado por el servidor y en el caso de ser de selección, si bien el usuario deberá escoger uno, su valor será uno de una lista específica de opciones. Para obtener el dato variable oculto o de selección se requiere de funciones que puedan leer una cadena de texto que contiene el código escrito en HTML y retornen el valor deseado. En la Tabla **2.9** se presenta la variable, la función para obtener dicha variable y una descripción de la función.

Tabla 2.9. Datos Variables ocultos y de selección

Dato Variable HTML	Nombre de la función	Descripción
__VIEWSTATE	getViewState()	Toma como argumento una cadena con el contenido HTML y retorna una cadena con el valor de la variable __VIEWSTATE
__VIEWSTATEGENERATOR	getViewState Generator()	Toma como argumento una cadena con el contenido HTML y retorna una cadena con el valor de la variable __VIEWSTATEGENERATOR
__EVENTVALIDATION	getEventVali dation()	Toma como argumento una cadena con el contenido HTML y retorna una cadena con el valor de la variable __EVENTVALIDATION
ctl00$ContentPlace Holder1$txtNombre	getTxtNombre()	Toma como argumento una cadena con el contenido HTML y retorna una cadena con el valor contenido en ctl00$Content PlaceHolder1$txtNombre
ctl00$ContentPlace Holder1$cmbperiodo	getCmbPeriod o()	Toma como argumento una cadena con el contenido HTML y retorna una cadena con el valor del periodo seleccionado.
ctl00$ContentPlace Holder1$cmbCarrera	getCmbCarrer a()	Toma como argumento de entrada una cadena con el contenido HTML y retorna un Diccionario, el cual contiene los ID y nombres de las carreras disponibles para el usuario.

2.2.2.4 Variables de respuesta

Estas variables necesitan ser procesadas, en la Tabla **2.10** se presenta un resumen del ID de la variable en el código HTML, la función a usar y su descripción.

Tabla 2.10. Variables del Carnet Estudiantil

Variable ID HTML	Función	Descripción
`ctl00_ContentPlaceHolder1_imgFoto`	`getImgFoto()`	A partir del código HTML se obtiene la cadena de texto de imagen en base 64 y formato JPEG de la foto de la cedula, lo cual se procesa para retornar un elemento tipo `Data`.
`ctl00_ContentPlaceHolder1_lblNombreComp`	`getLblNombreComp()`	A partir del código HTML se retorna un `String` que contiene el nombre completo del usuario.
`ctl00_ContentPlaceHolder1_lblCedulaEst`	`getLblCedulaEst()`	A partir del código HTML se retorna un `String` que contiene el número de cédula del usuario.
`ctl00_ContentPlaceHolder1_lblCodest`	`getLblCodest()`	A partir del código HTML se retorna un `String` que contiene el código único del usuario.
`ctl00_ContentPlaceHolder1_lblFacultaEst`	`getLblFacultaEst()`	A partir del código HTML se retorna un `String` que contiene la facultad de la carrera seleccionada por el usuario.
`ctl00_ContentPlaceHolder1_lblCarreraEst`	`getLblCarreraEst()`	A partir del código HTML se retorna un `String` que contiene la carrera seleccionada por el usuario.
`ctl00_ContentPlaceHolder1_imgQRCode`	`getImgQRCode()`	A partir del código HTML se obtiene la cadena de texto de imagen en base 64 con formato JPEGN del código QR, lo cual se procesa para retornar un elemento tipo `Data`.

2.2.2.5 Funciones para comunicación

En la etapa de inicio de sesión en el SAEw se interactúa con el recurso `Login.aspx` y en la de obtención del carnet se interactúa con `CarneEstudiantil.aspx`; para cada etapa

se desarrollaron dos funciones `realizarEtpSesion()` y `realizarEtpCarnet()`. Para compartir la *cookie* de la sesión se emplea un objeto singleton de tipo `URLSession.shared`.

En el caso de la función `realizarEtpSesion()` se abarcarán los pasos desde el 1 al 5, en la Tabla **2.11** se puede apreciar los pasos asociados con la función en la que se realizará ese paso y su descripción.

Tabla 2.11. Funciones perteneciente a `realizarEtpSesion()`

# Paso	Paso a realizar	Nombre de la función	Descripción
1	Iniciar comunicación con el SAEw	`iniciarComunicacion()`	Petición GET, retorna la página HTML en un `String`.
2	Cambiar a modo de estudiante	`cambiarModoEstudiante()`	Petición POST, envía los datos de la Tabla **2.1** y retorna la página HTML en un `String`.
3	Validar datos de usuario	`validarDatos()`	Petición POST, envía los datos de la Tabla **2.2** y retorna la página HTML en un `String`.
4	Aceptar validación	`aceptarValidacion()`	Petición POST, envía los datos de la Tabla **2.3**, y retorna la página HTML en un `String`.
5	Ingresar al sistema	`ingresarSistema()`	Petición POST, envía los datos de la Tabla **2.4** y retorna la página HTML en un `String`.

La función `realizarEtpCarnet()` abarca los pasos 6 y 7; en la Tabla **2.12** se pueden apreciar los pasos asociados a la función respectiva y su descripción.

Tabla 2.12. Funciones perteneciente a `realizarEtpCarnet()`

# Paso	Paso a realizar	Nombre de la función	Descripción
6	Ingresar al Carnet Estudiantil	`ingresarCarnet()`	Petición GET, retorna la pagina HTML en un `String`.
7	Mostrar Carnet Estudiantil	`obtenerCarnet()`	Petición POST, envía los datos de la Tabla **2.5** y retorna la página HTML en un `String`.

2.2.3 ARQUITECTURA MVVM

En esta sección se describen las capas correspondientes al *Model*, *View* y *ViewModel*. En la Figura **2.26** se observan los archivos ubicados en la carpeta que corresponde a cada capa.

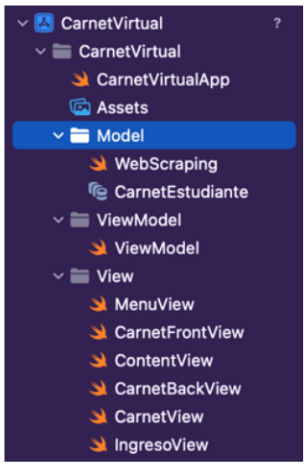


Figura 2.26. Archivos distribuidos en las capas de la arquitectura MVVM

2.2.3.1 Model

En la capa del modelo se abarca la lógica de negocios y de datos, aquí se encuentran los archivos `WebScraping.swift` y `CarnetEstudiante.xcdatamodeld`.

`WebScraping.swift`: contiene todas las funciones detalladas en la sección 2.2.2 las cuales se usan desde la capa ViewModel.

`CarnetEstudiante.xcdatamodeld`: Este archivo representa la base de datos que se almacenará en el dispositivo iOS. La base de datos se observa en la Figura **2.27**.

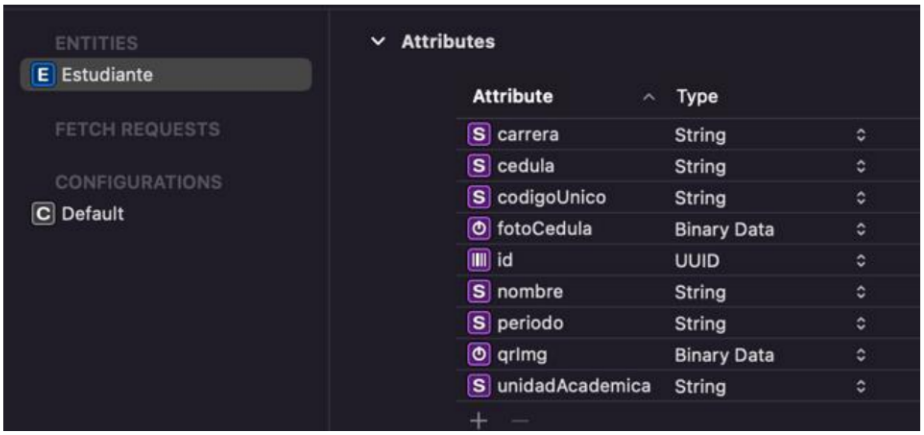


Figura 2.27. Contenido del archivo `CarnetEstudiante.xcdatamodeld`

En la Figura **2.27** se observa la existencia de una entidad denominada Estudiante, la cual tiene los atributos: carrera, cedula, codigoUnico, fotoCedula, id, nombre, periodo, qrImg y unidadAcademica.

2.2.3.2 View

Los archivos que componen la capa *View* se generaron a partir de los *sketches* realizados en la sección 2.1.6, los archivos que conforman esta capa son: `ContentView.swift`, `IngresoView.swift`, `MenuView.swift`, `CarnetView.swift`, `CarnetFrontView.swift` y `CarnetBackView.swift`.

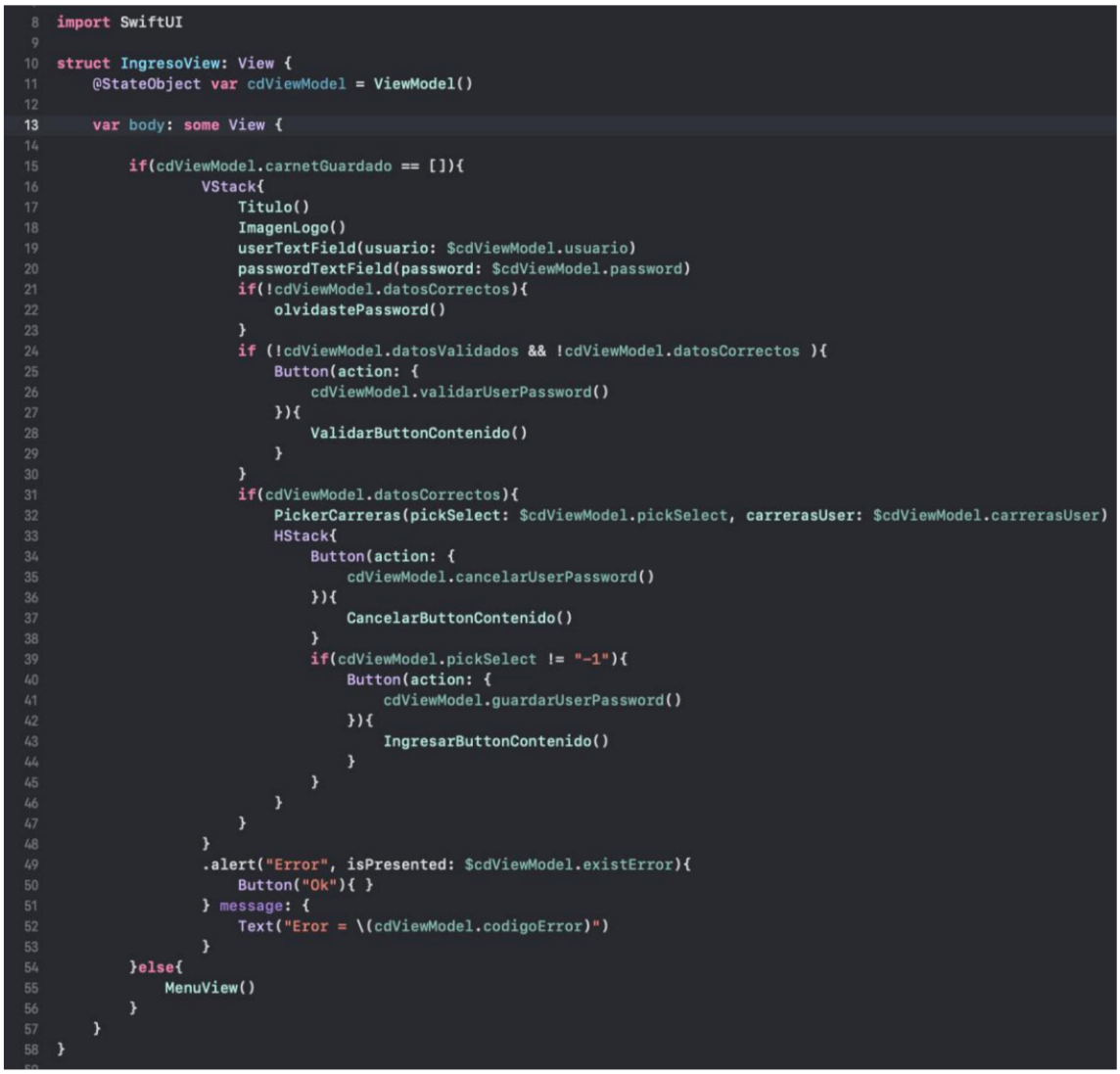
`ContentView.swift`: Es el archivo principal del protocolo `View`, se ejecuta al abrir la aplicación gracias al protocolo App que se encuentra en el archivo `CarnetVirtualApp.swift`, y sirve para redireccionar al archivo `IngresoView.swift` en caso de que no exista información sobre el carnet guardado en la base de datos, caso contrario sirve para redireccionar al archivo `MenuView.swift`. En la Figura **2.28** se encuentra el código y se observa el uso del condicional `if` con la variable carnetGuardado correspondiente a la clase `ViewModel`. En la estructura `PreviewProvider` se expresa que la orientación del dispositivo debe ser vertical.

```swift
import SwiftUI


struct ContentView: View {
    @StateObject var cdViewModel = ViewModel()

    var body: some View {
        if(cdViewModel.carnetGuardado == []){
            IngresoView()
        }else{
            MenuView()
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        Group {
            ContentView()
                .previewInterfaceOrientation(.portrait)
        }
    }
}
```

Figura 2.28. Contenido del archivo `ContentView.swift`

`IngresoView.swift`: Se modifica de acuerdo con las acciones que realice el usuario para iniciar sesión. En la Figura **2.29** se aprecia un fragmento del código de este archivo,

el cual contiene la distribución y agrupación de las diferentes estructuras mediante el uso de los métodos Vstack, para separar estructuras verticalmente, y Hstack, para separar estructuras horizontalmente.



```swift
import SwiftUI

struct IngresoView: View {
    @StateObject var cdViewModel = ViewModel()

    var body: some View {

        if(cdViewModel.carnetGuardado == []){
                VStack{
                    Titulo()
                    ImagenLogo()
                    userTextField(usuario: $cdViewModel.usuario)
                    passwordTextField(password: $cdViewModel.password)
                    if(!cdViewModel.datosCorrectos){
                        olvidastePassword()
                    }
                    if (!cdViewModel.datosValidados && !cdViewModel.datosCorrectos ){
                        Button(action: {
                            cdViewModel.validarUserPassword()
                        }){
                            ValidarButtonContenido()
                        }
                    }
                    if(cdViewModel.datosCorrectos){
                        PickerCarreras(pickSelect: $cdViewModel.pickSelect, carrerasUser: $cdViewModel.carrerasUser)
                        HStack{
                            Button(action: {
                                cdViewModel.cancelarUserPassword()
                            }){
                                CancelarButtonContenido()
                            }
                            if(cdViewModel.pickSelect != "-1"){
                                Button(action: {
                                    cdViewModel.guardarUserPassword()
                                }){
                                    IngresarButtonContenido()
                                }
                            }
                        }
                    }
                }
                .alert("Error", isPresented: $cdViewModel.existError){
                    Button("Ok"){ }
                } message: {
                    Text("Eror = \(cdViewModel.codigoError)")
                }
        }else{
            MenuView()
        }
    }
}
```

Figura 2.29. Fragmento de contenido del archivo `IngresoView.swift`

La vista `IngresoView` está compuesta por diferentes estructuras con un diseño personalizado, estas estructuras se detallan en la Tabla **2.13**.

`MenuView.swift`: Usa `NavigationView{}`, con lo cual se permite la navegación entre las diferentes vistas.

`CarnetView.swift`: Este archivo es un intermediario, que permite la navegación mediante un `TabView`, entre las vistas definidas en los archivos `CarnetFrontView.swift` y `CarnetBackView.swift`.

`CarnetFrontView.swift`: Muestra los datos de la parte frontal del carnet Estudiantil.

`CarnetBackView.swift`: Muestra los datos de la parte trasera del carnet Estudiantil.

Tabla 2.13. Estructuras pertenecientes a la vista `IngresoView`

Nombre de la estructura	Descripción
`Titulo`	Compuesta de una estrutura `Text`, muestra el texto “Carnet Virtual".
`ImagenLogo`	Compuesta de una estructura `Image`, muestra la imagen logo de la EPN.
`userTextField`	Compuesta de una estrutura `TextField`, permite el ingreso de la Cédula.
`passwordTextField`	Compuesta de una estrutura `SecureField`, permite el ingreso de la Contraseña.
`olvidastePassword`	Compuesta de una estrutura `Text`, muestra un enlace para recuperar contraseña.
`ValidarButtonContenido`	Diseño y contenido del botón validar.
`PickerCarreras`	Compuesto de una estructura `Picker` permite seleccionar una carrera entre múltiples carreras.
`CancelarButtonContenido`	Diseño y contenido del botón cancelar.
`IngresarButtonContenido`	Diseño y contenido del botón ingresar.

2.2.3.3 ViewModel

El archivo `ViewModel.swift` contiene la clase `ViewModel` la cual se detalla como un `ObservableObject`[6] propiedad necesaria para poder transmitir información con las vistas. Entre las funciones de esta clase están: la comunicación con la base de datos y con el servidor SAEw. Esta clase usa los recursos establecidos en la capa de modelo o en la base de datos.

En esta clase solo existe una variable constante denominada `contenedor` y que es de tipo `NSPersistentCloudKitContainer`, mediante la cual se llama a la base de datos; el resto de la variables son marcadas con el atributo `@Published` con lo que se garantiza que puedan ser accedidas desde la capa Vista, En la Tabla **2.14** se aprecian las variables con su descripción.

[6] `ObservableObject`: Permite que las variables sean accedidas desde las vistas, así puede realizar una actualización de la vista ante un cambio de la variable.

Tabla 2.14. Variables pertenecientes a la clase `ViewModel`

Nombre de la variable	Tipo	Valor inicial	Descripción
`carnetGuardado`	`Estudiante`	`[]`	Sirve para almacenar la información que se encuentra en el `contenedor`.
`usuario`	`String`	`""`	Se usa para almacenar la información de entrada de texto en el campo Cédula.
`password`	`String`	`""`	Se usa para almacenar la información de entrada de texto en el campo Contraseña.
`pickSelect`	`String`	`"-1"`	Se usa para almacenar el valor de la carrera seleccionada, el valor `-1` significa que esta seleccionada la opción `SELECCIONA TU ACTUAL CARRERA`.
`periodo`	`String`	`""`	Almacena el último periodo disponible en el SAEw.
`existError`	`Bool`	`False`	Almacena el estado en caso de existir algún error en la comunicación o debido a que las credenciales son incorrectas. En caso de error su valor será `True`.
`datosCorrectos`	`Bool`	`False`	Almacena el estado en caso de que los datos sean correctos, su estado se modifica a `True` cuando se verifica la información.
`codigoError`	`Int`	`0`	Indica el código del error.
`carrerasUser`	`Carreras`	`[]`	Almacena la carreras en una estructura denominada `Carreras`, que contiene dos elementos, uno para almacenar el nombre de la carrera y el otro para su ID.

Cuando se inicializa un objeto de esta clase, se crea un contenedor que será el encargado de manejar la base de datos propia de iOS. En la Figura **2.30** se observa el código para la inicialización de esta clase, ahí se referencia al archivo que se encuentra en la capa modelo llamado `CarnetEstudiante.xcdatamodeld`.

```
init() {
    // A la variable contenedor se le inicializa con el Data Model que tiene de nombre "CarnetEstudiante"
    contenedor = NSPersistentCloudKitContainer(name: "CarnetEstudiante")
    // Indica al contenedor que cargue todos los datos necesarios, en caso de error informa
    contenedor.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            fatalError("Error 1 en \(error), \(error.userInfo)")
        }
    })
    // llama a la función getCarnet
    getCarnet()
}
```

Figura 2.30. Inicialización de la clase `ViewModel`

En Tabla **2.15** se aprecian las funciones desarrolladas en la clase `ViewModel` con su respectiva descripción.

Tabla 2.15. Funciones pertenecientes a la clase `ViewModel`

Nombre de la función	**Descripción**
`getCarnet()`	Obtiene los valores de la entidad `Estudiante` y la almacena en `carnetGuardado`
`guardarCambios()`	Guarda los cambios efectuados y llama a la función `getCarnet()`.
`agregarInformacion()`	Cera un nuevo registro en la entidad `Estudiante`, llama a la función `guardarCambios()`.
`validarUserPswd()`	Se usa en la etapa 1, inicio de sesión en el SAEw, y actualiza las variables dependiendo de si existió o no algún error.
`cancelarUserPswd()`	Inicializa las variables.
`guardarUserPswd()`	Se usa en la etapa 2, obtención del carnet, y actualizar las variables dependiendo de si existió o no algún error, llama a la función `agregarInformacion()`
`deleteCarnetData()`	Elimina todos los registros de la entidad `Estudiante` de la base de datos.
`deleteCarnet()`	Inicializar las variables y llama a la función `deleteCarnetData()`.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 RESULTADOS

En esta sección se presentan los resultados de las pruebas del prototipo, y en caso de existir errores se presenta su corrección. En el ANEXO IX se adjunta la última versión del código del prototipo.

3.1.1 PRUEBAS DE FUNCIONALIDAD

El prototipo funciona correctamente, en la Figura **3.1** se aprecia la prueba realizada en un iPhone 7 habilitado el modo *Dark*[7], mientras que en la Figura **3.2** se aprecia la prueba realizada con otro usuario, pero con el modo *Light*[8] habilitado.

Existe un error cuando se encuentra el usuario en la pantalla número 3 que se presenta en la Figura **3.1** y en la Figura **3.2**, el error es que se permite modificar la Cédula y contraseña despues de que estos datos hayan sido validado, la modificación de estos valores no afecta la funcionalidad de la aplicación.

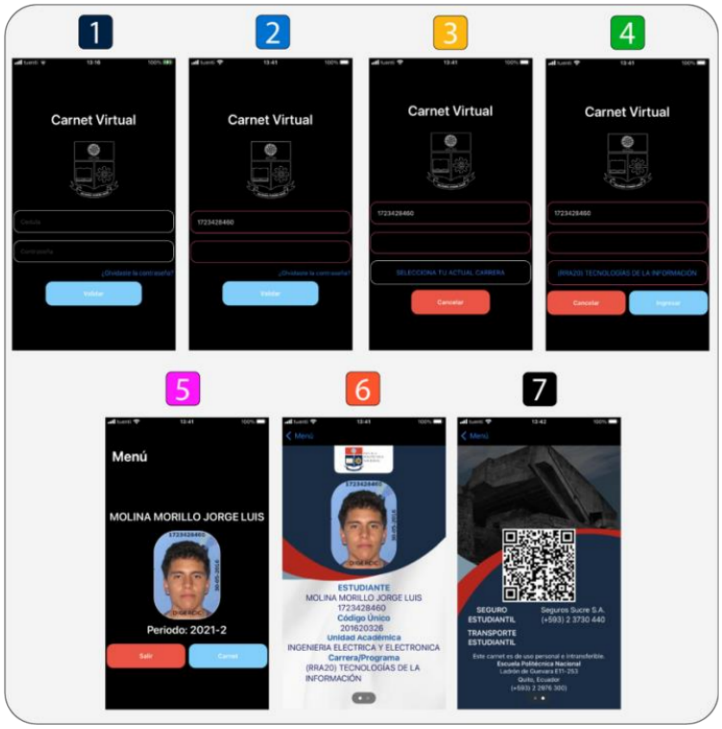


Figura 3.1. Pruebas de funcionalidad en el modo *Dark*

[7] Es un entorno que modifica la apariencia de las aplicaciones, su óptimo uso se produce en ambientes con poca luz.

[8] Es el entorno común, se usa en ambientes con luz.

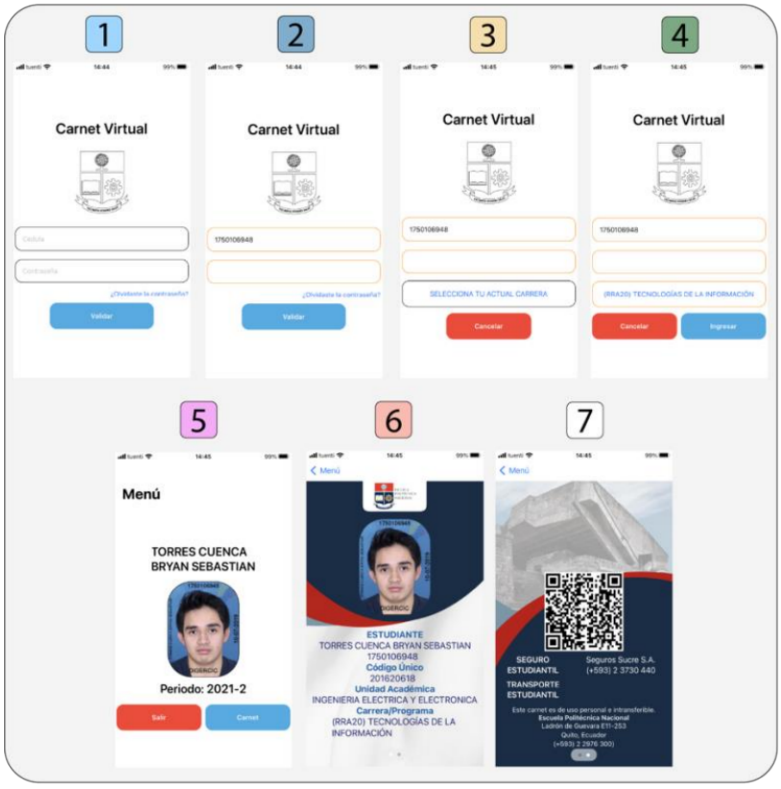


Figura 3.2. Pruebas de funcionalidad en el modo *Light*

En la Figura **3.3** se presenta la vista con un error, en este caso es el error -4 que significa que el usuario no existe. Debido a que el usuario necesita saber que la cédula que digitó es incorrecta se procede a corregir la vista para que muestre en el cuadro de dialogo “No existe usuario”. En el apartado 3.1.2 se indicará como se corrigió este error.

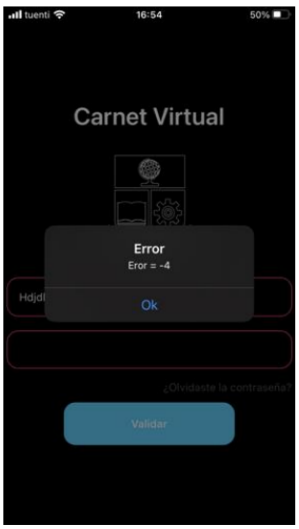


Figura 3.3. Vista que muestra un error

3.1.2 CORRECCIÓN DE ERRORES

Para que se presenten los errores se creó un diccionario en el cual, `key` es el número del error, mientras que `value` es la descripción del error. En la Tabla **3.1** se presentan los errores con su descripción. El diccionario se encuentra en la clase `ViewModel` y se usa desde la capa Vista.

Tabla 3.1. Error y descripción

# Error	Descripción técnica	Descripción Visual
-1	Error al realizar el paso 1, existió error al realizar la petición GET	No se pudo conectar con el SAEw
-2	Error de comunicación al realizar el paso 2, al cambiar el modo a estudiante.	Fallo en la comunicación con el SAEw
-3	Error de comunicación al realizar el paso 3.	Fallo en la comunicación con el SAEw
-4	Error por usuario incorrecto en el paso 3.	No existe usuario
-5	Error por clave incorrecta en el paso 3.	Clave no válida
-6	Error de comunicación al realizar el paso 4.	Fallo en la comunicación con el SAEw
-7	Error de comunicación al realizar el paso 5.	Fallo en la comunicación con el SAEw
-8	Error de comunicación al realizar el paso 6.	Fallo en la comunicación con el SAEw
-9	Error de comunicación al realizar el paso 7.	Fallo en la comunicación con el SAEw
-10	Erro en el paso 7 al no estar matriculado en la carrera seleccionada en el periodo actual.	No existe un carnet para la carrera seleccionada.

En la **Figura 3.4** se muestra el código de la alerta de error corregida, que se encuentra en la la vista `IngresoView`, aquí se observa el uso del diccionario `errorRespuestas` que como valor de la `key` es el `codigoError`.

```
.alert("Error \(cdViewModel.codigoError)", isPresented:
    $cdViewModel.existError){
    Button("Ok"){ }
} message: {
    Text("\(cdViewModel.errorRespuestas
        [cdViewModel.codigoError] ?? "Error")")
}
```

Figura 3.4. Alerta de error corregida

Para corregir el error de modificación de Cédula y Contraseña después de ser validados, se usó el estado de la variable `datosCorrectos`, en el caso de que los datos sean correctos se desabilita la opcion de que se puedan modificar, caso contrario los datos si se pueden modificar. En la Figura **3.5** se aprecia el fragmento de código incluido en el archivo `IngresoView.swift` para poder activar o desactivar los campos de Cédula y Contraseña.

```
if(!cdViewModel.datosCorrectos){
    usuarioyPassword(usuario: $cdViewModel.usuario,
        password: $cdViewModel.password)
    olvidastePassword()
}else{
    usuarioyPassword(usuario: $cdViewModel.usuario,
        password: $cdViewModel.password)
        .disabled(true)
}
```

Figura 3.5. Código que modifica estado de los campos Cédula y Contraseña

3.1.3 ENCUESTA DE VALIDACIÓN

Como requisito para realizar la encuesta se solicitó a colegas de la Institución que primero se familiaricen con el prototipo, para lo cual se les capacitó mediante Microsoft Teams y un emulador de iPhone 11. 5 estudiantes de la Institución recibieron la capacitación y probaron el prototipo. En el ANEXO IV se incluyen enlaces a los videos sobre el uso del prototipo.

El formato de la encuesta de validación se encuentra en el ANEXO V, la encuesta se realizó en Microsoft Forms para limitar el acceso a estudiantes de la EPN. En la encuesta, los cuatros primeros literales corresponden a la opinión acerca del diseño del prototipo, en cambio los tres restantes están relacionados con su funcionamiento.

En el ANEXO VI se encuentran las respuestas obtenidas a la encuesta.

En cuestión al diseño del carnet, la percepción del público objetivo es que el diseño es similar al presentado en el SAEw, con una puntuación promedio de 8.33, como se puede observar en la Figura **3.6**. Como observación se indica que la imagen de fondo de la parte posterior del carnet se cambie.

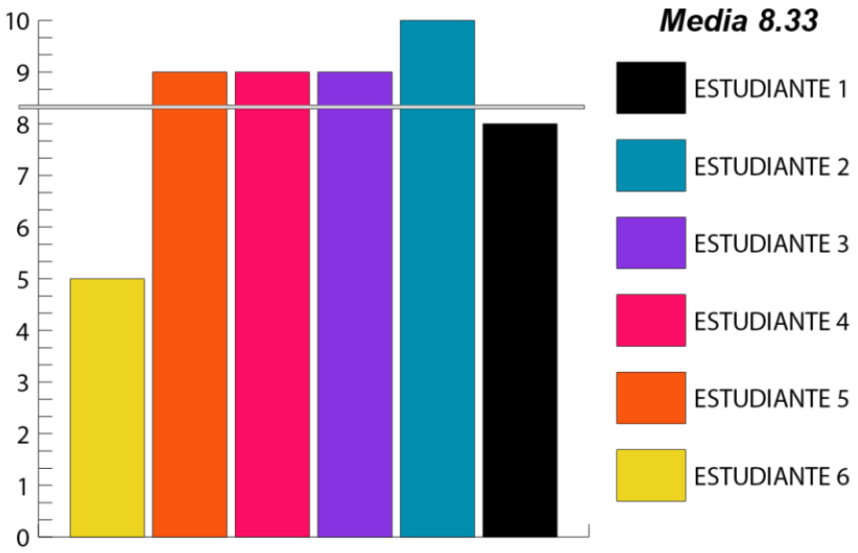


Figura 3.6. Calificación de similitud con el diseño del Carnet del SAEw

En el diseño se preguntó sobre el modo *Light* o *Dark* del prototipo, esto se observa en la Figura **3.7**. A 2 de las 6 personas les atrajo el modo *Dark* y al resto el modo *Light*, esto refuerza el argumento del diseño de dos modos de vistas ya que permite que la aplicación sea personalizada por el usuario.

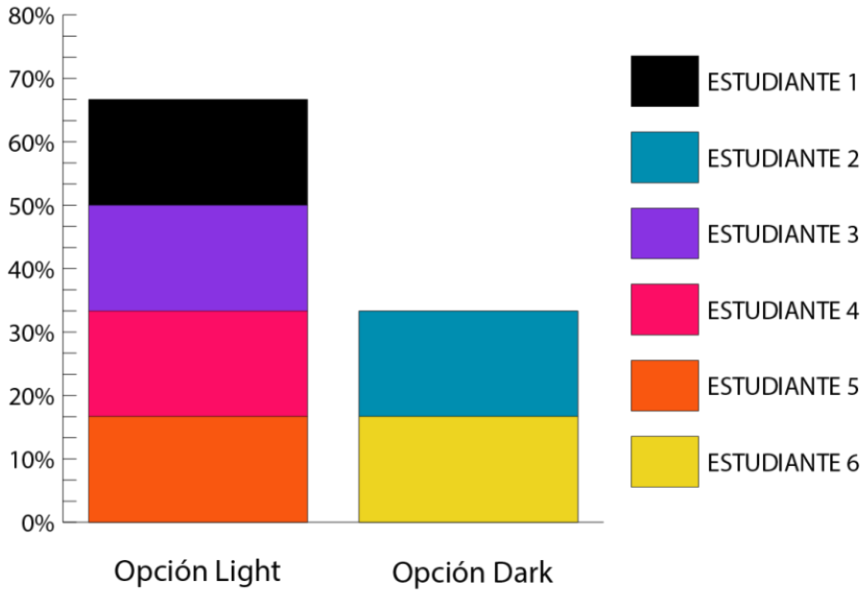


Figura 3.7. Preferencia sobre el modo de la aplicación

En la pregunta de si el prototipo cumple sus expectativas en cuestión al funcionamiento, los encuestados dan una calificación promedio de 9.67 sobre 10, esto se observa en la Figura **3.8**, una de las posibles causas de que el puntaje no haya sido 10 es que se probó el prototipo mediante un emulador y con la opción de uso remoto de Microsoft Teams.

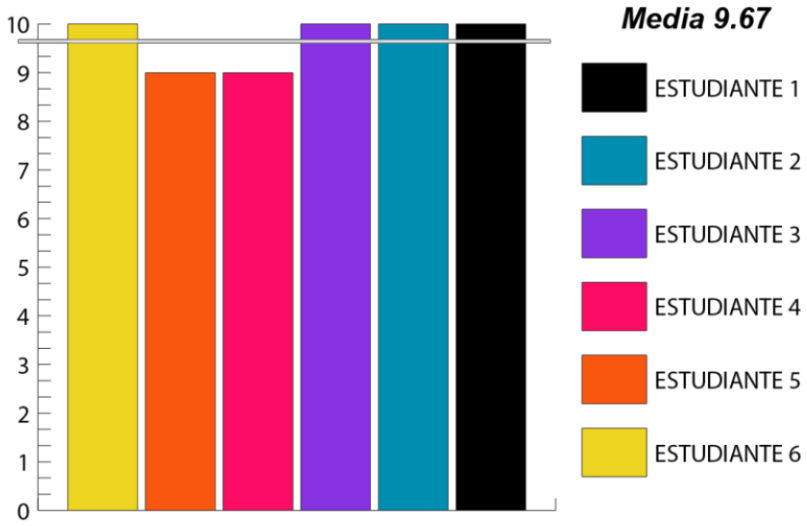


Figura 3.8. Calificación de funcionamiento del prototipo

La última pregunta en cuestión al funcionamiento permite evaluar la necesidad de incluir que muestre u oculte la contraseña. La totalidad de encuestados expreso su opinión favorable a que se habilite esa opción, como se observa en la Figura **3.9**.

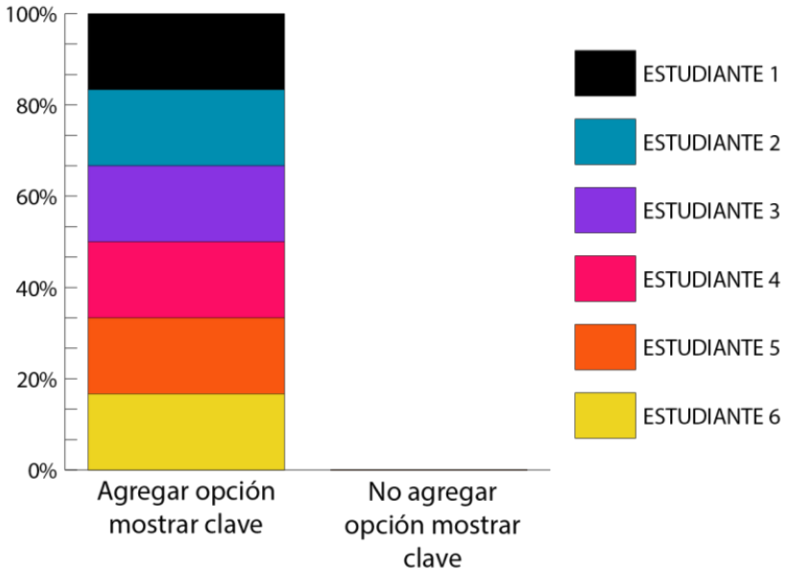


Figura 3.9. Implementación de opción mostrar clave

3.1.4 CORRECCIONES CON BASE EN LOS RESULTADOS DE LA ENCUESTA

Se estableció que se requería corregir la imagen de fondo de la parte posterior del carnet que se encuentra en la vista `CarnetBackView` y también se debe agregar el campo de contraseña visible en la vista `IngresoView`.

En el ANEXO VIII se incluye un enlace a un video en el que se usó el prototipo de forma rápida, se procedió a tomar el tiempo y se determinó que se requerían 29 segundos para acceder y almacenar el carnet.

3.1.4.1 Modificación de la vista CarnetBackView

Para realizar esta modificación solo fue necesario realizar el cambio de imagen con una en la que el fondo no sea semitransparente. En la Figura **3.10** se aprecia el resultado.

Figura 3.10. Antes y después del cambio en la vista `CarnetBackView`

3.1.4.2 Modificación de la vista IngresoView

Para crear el botón que muestra la clave, se define una variable de tipo `bool` llamada `showPassword` en la clase `ViewModel`, mediante esta variable y la generación de condiciones necesarias en la vista `IngresoView`, se obtiene el resultado que se muestra en la Figura **3.11**.

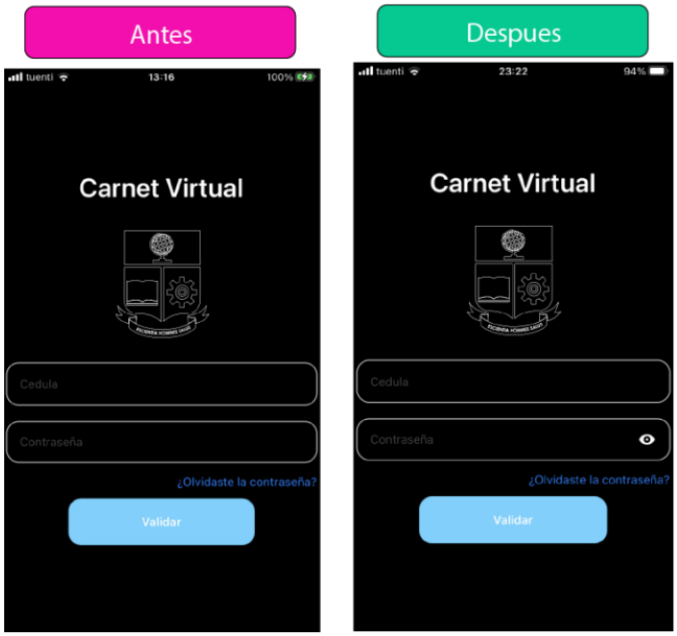


Figura 3.11. Antes y después del cambio en la vista `IngresoView`

3.2 CONCLUSIONES

Al concluir este Trabajo de Integración Curricular se dispone de un prototipo de aplicación que obtiene el carnet estudiantil desde el SAEw y lo presenta en un dispositivo con sistema operativo iOS.

Dado que el SAEw no dispone de una API, fue necesario emplear *web scraping* como solución para extraer datos desde la página web, la desventaja que esto conlleva es que el código desarrollado se debe actualizar si el sitio web se modifica.

El prototipo funcionará correctamente, mientras no se realicen actualizaciones a los recursos `Login.aspx` y `CarneEstudiantil.aspx`, cabe recalcar que haber usado el patrón Model-View-ViewModel es una ventaja ya que, en caso de modificación de algún recurso, disminuiría la complejidad en la actualización del código.

El prototipo realizado es más eficiente que emplear el navegador web móvil Safari, debido a que toma aproximadamente 30% el tiempo que se requiere en acceder y almacenar el carnet con el prototipo en comparación con la forma tradicional.

Al hacer uso de la arquitectura MVVM permitió que la corrección de errores sea más rápida, al solo tener que enfocarse en la capa donde se encuentra el error y hacer mínimas correcciones de ser necesario en las demás capas.

3.3 RECOMENDACIONES

Se recomienda usar el presente trabajo como base para la realización de una aplicación para dispositivos Android que muestre el Carnet Estudiantil, para esto se podría usar Jetpack Compose con el cual se desarrollan interfaces de usuario de manera similar que SwiftUI.

Se podría usar el prototipo como base para la realización de una aplicación iOS que interactúe con el SAEw para obtener otro tipo de recursos, por ejemplo, el horario del estudiante o las notas de las materias en las que se encuentra matriculado.

Se recomienda realizar pruebas del prototipo para estudiantes que hayan sido registrados con pasaporte, esto no se realizó debido a que la población estudiantil que usa pasaporte es muy pequeña.

En el caso de que se desee realizar una aplicación similar y se desee usar JavaScript, se podría hacer uso de WKWebView, una clase de Swift que es compatible con ese lenguaje.

Se sugiere que cuando se realicen las vistas, estas sean simples, debido a que una saturación de estructuras podría generar un efecto contraproducente en los usuarios.

Se aconseja que para desarrollar aplicaciones para iOS se utilice un equipo con almacenamiento mayor a 128 GB, debido a que en el desarrollo de este prototipo las herramientas necesarias ocuparon aproximadamente el 95% de la capacidad del disco duro.

Para usar la metodología Kanban se sugiere el uso de software de terceros que permite el almacenamiento online, ya que esto genera dinamismo en la realización del prototipo.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] B. Zhao, Web Scraping, L. A. Schintler and C. L. McNeely, Eds., Springer International Publishing, 2017, pp. 1-3.

[2] MDN contributors, "Using HTTP cookies," 20 January 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. [Accessed 15 February 2022].

[3] MDN contributors, "HTTP request methods," 3 October 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. [Accessed 15 February 2022].

[4] MDN contributors, "GET," 13 August 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET. [Accessed 15 February 2022].

[5] MDN contributors, "POST," 13 August 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST. [Accessed 15 February 2022].

[6] Apple Inc, «About Swift,» 2022. [En línea]. Available: https://www.swift.org/about/. [Último acceso: 15 February 2022].

[7] Apple Inc, "The Basics," 2022. [Online]. Available: https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html. [Accessed 15 February 2022].

[8] Apple Inc, "Collection Types," 2022. [Online]. Available: https://docs.swift.org/swift-book/LanguageGuide/CollectionTypes.html. [Accessed 15 February 2022].

[9] Apple Inc, "Functions," 2022. [Online]. Available: https://docs.swift.org/swift-book/LanguageGuide/Functions.html. [Accessed 15 February 2022].

[10] Apple Inc, "Structures and Classes," 2022. [Online]. Available: https://docs.swift.org/swift-book/LanguageGuide/ClassesAndStructures.html. [Accessed 15 February 2022].

[11] Apple Inc, "URLSession," 2022. [Online]. Available: https://developer.apple.com/documentation/foundation/urlsession. [Accessed 15 February 2022].

[12] Apple Inc, "URLRequest," 2022. [Online]. Available: https://developer.apple.com/documentation/foundation/urlrequest. [Accessed 15 February 2022].

[13] Apple Inc, "SwiftUI," 2022. [Online]. Available: https://developer.apple.com/documentation/swiftui. [Accessed 15 February 2022].

[14] Apple Inc, "App," 2022. [Online]. Available: https://developer.apple.com/documentation/swiftui/app. [Accessed 15 February 2022].

[15] Apple Inc, "View," 2022. [Online]. Available: https://developer.apple.com/documentation/swiftui/view. [Accessed 15 February 2022].

[16] J. Smith, «Patterns - WPF Apps With The Model-View-ViewModel Design Pattern,» February 2009. [En línea]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern. [Último acceso: 17 February 2022].

[17] H. Kniberg y M. Skarin, Kanban and Scrum making the most of both, lulu.com; null edition, 2010.

5 ANEXOS

ANEXO I. Obtención del carnet en navegador web móvil Safari

ANEXO II. Tablero Kanban en estado inicial

ANEXO III. Tablero Kanban

ANEXO IV. Enlaces de videos de 5 estudiantes probando el prototipo en un simulador

ANEXO V. Formato de Encuesta

ANEXO VI. Resultados de la Encuesta

ANEXO VII. Enlaces de videos de uso de la versión final del prototipo en un dispositivo real

ANEXO VIII. Obtención del carnet de forma rápida mediante el prototipo

ANEXO IX. Código del prototipo

ANEXO I

El video de la obtención del Carnet Estudiantil mediante el navegador web móvil Safari se encuentra en la siguiente URL:

https://youtu.be/5LtBZflQ4EQ

ANEXO II

El tablero Kanban detallado es el siguiente:

Kanban

#	NOMBRE	DESCRIPCIÓN
ESTADO : INICIO		
1	Diagrama de flujo - REDACCIÓN	Desarrollar el diagrama de flujo del prototipo.
2	Sketches de las vista - REDACCIÓN	Realizar sketches para las vistas.
3	Instalación de Xcode - REDACCIÓN	Instalar el ambiente de desarrollo.
4	Arquitectura MVVM - REDACCIÓN	Desarrollar las vistas y controladores mediante SwiftUI y UIKit.
5	Web Scraping en Swift - REDACCIÓN	Desarrollar la lógica que se usará mediante el web scraping para acceder al SAEw y obtener el carnet estudiantil e implementarla mediante Swift.
6	Pruebas de funcionalidad - REDACCIÓN	Realizar pruebas de funcionalidad en dispositivos móviles.
7	Corrección de errores - REDACCIÓN	Corregir errores detectados en las pruebas de funcionalidad.
8	Corrección de errores - PRÁCTICO	Definir la encuesta de validación.
9	Encuesta de validación - REDACCIÓN	Aplicar las encuestas de validación a al menos 5 estudiantes de la EPN.
10	Corrección en base a la encuesta - REDACCIÓN	Realizar corrección del prototipo, de ser necesario.
11	Arquitectura MVVM - PRÁCTICO	Desarrollar las vistas y controladores mediante SwiftUI y UIKit.
12	Web Scraping en Swift - PRÁCTICO	Desarrollar la lógica que se usará mediante el web scraping para acceder al SAEw y obtener el carnet estudiantil e implementarla mediante Swift.
13	Pruebas de funcionalidad - PRÁCTICO	Realizar pruebas de funcionalidad en dispositivos móviles.
14	Encuesta de validación - PRÁCTICO	Aplicar las encuestas de validación a al menos 5 estudiantes de la EPN.
15	Corrección en base a la encuesta - PRÁCTICO	Realizar corrección del prototipo, de ser necesario.
ESTADO : DESARROLLO		
16	Tablero Kanban - REDACCIÓN	Establecer las tareas en el tablero Kanban.
ESTADO : REVISIÓN		
17	Web Scraping - REDACCIÓN	Analizar la técnica web scraping.
18	Generalidades del lenguaje Swift - REDACCIÓN	Estudiar las generalidades del lenguaje Swift.
19	Lenguaje SwifUI - REDACCIÓN	Analizar SwiftUI para su uso en el almacenamiento de datos, creación de controlador, modelo y vistas.
20	Metodología Kanban - REDACCIÓN	Estudiar la metodología ágil de desarrollo de software Kanban.
21	Proceso de obtención del carnet del SAEw - REDACCIÓN	Estudiar el funcionamiento del SAEw.
22	Requerimientos funcionales - REDACCIÓN	Generar requerimientos funcionales.
23	Requerimientos no funcionales - REDACCIÓN	Generar requerimientos no funcionales.
ESTADO : FINALIZADO		
24	Instalación de Xcode - PRÁCTICO	Instalar el ambiente de desarrollo.

ANEXO III

El tablero Kanban desarrollado mediante Airtable se encuentra en la siguiente URL:

https://airtable.com/shryFtwyVzEo12XEE/tblrpXVcpJhfFeCKc

ANEXO IV

1. Enlace con video del prototipo en simulador de iPhone 11 por estudiante 1

URL: https://youtu.be/sYGeXT7kMV0

2. Enlace con video del prototipo en simulador de iPhone 11 por estudiante 2

URL: https://youtu.be/FK38VcQG-6w

3. Enlace con video del prototipo en simulador de iPhone 11 por estudiante 3

URL: https://youtu.be/3ccRrq04AII

4. Enlace con video del prototipo en simulador de iPhone 11 por estudiante 4

URL: https://youtu.be/3A5LxiCrUnw

5. Enlace con video del prototipo en simulador de iPhone 11 por estudiante 5

URL: https://youtu.be/hh3YRJ6Od30

ANEXO V

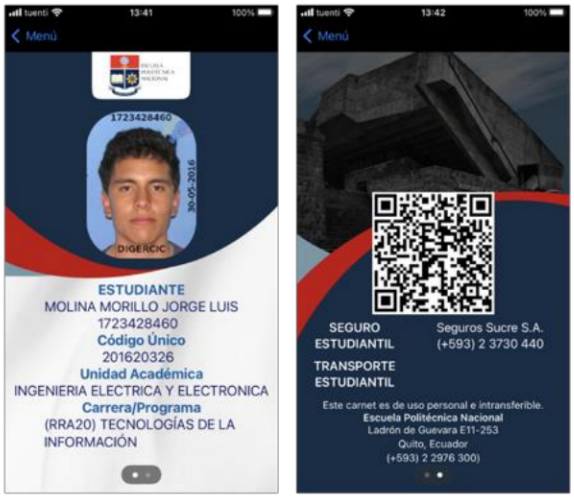
Encuesta de validación de prototipo que muestra el carnet estudiantil en iOS

Hi, NOMBRE DE ESTUDIANTE. When you submit this form, the owner will see your name and email address.

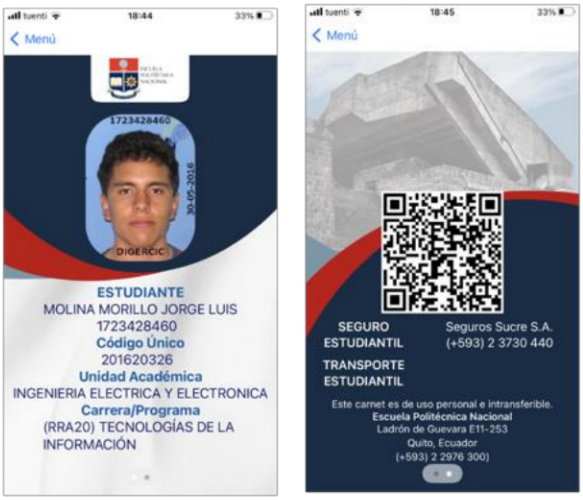
1. ¿Qué tan parecido es el diseño del carnet del prototipo en comparación con el empleado en el SAEw?

   1 significa nada parecido, 10 completamente parecido

Prototipo modo Dark



Prototipo modo Light

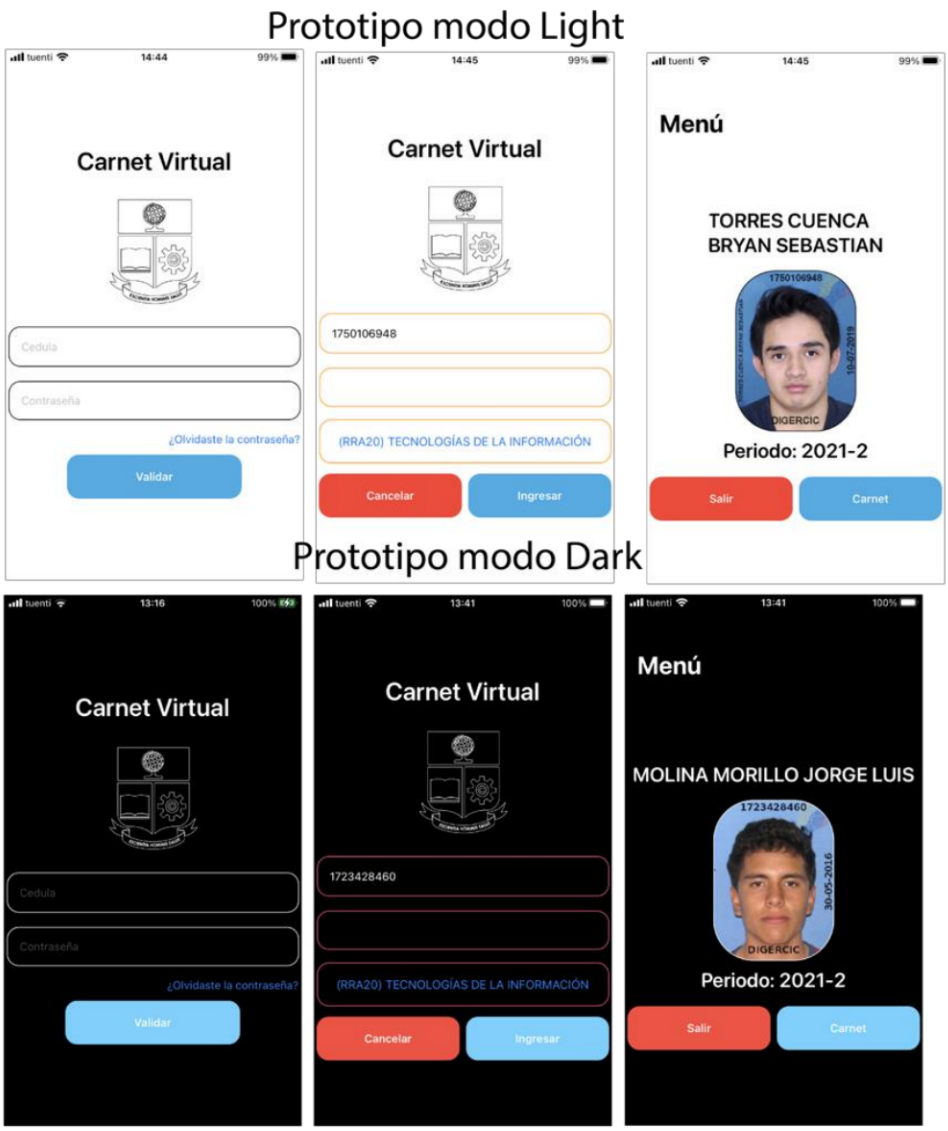


Carnet del SAEw

1	2	3	4	5	6	7	8	9	10
◎	◎	◎	◎	◎	◎	◎	◎	◎	◎

2. (Opcional) Explica tu respuesta a la pregunta 1

3. ¿Que opción prefieres?



◎ Opción Dark

◎ Opción Light

◎ No tengo opción preferida

4. (Opcional) Explica tu respuesta a la pregunta 3

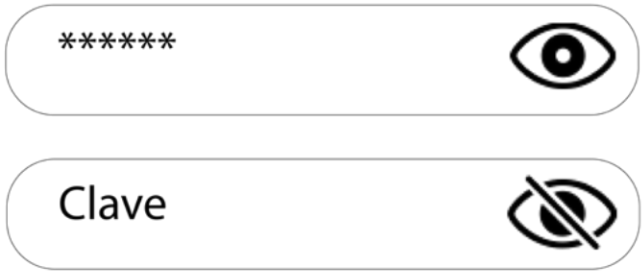
5. En cuestión al funcionamiento, ¿el prototipo cumple tus expectativas?

   1 significa para nada, 10 cumple todas mis expectativas

1	2	3	4	5	6	7	8	9	10
◎	◎	◎	◎	◎	◎	◎	◎	◎	◎

6. (Opcional) Explica tu respuesta a la pregunta 5

7. El prototipo no tiene la opción de mostrar la clave, consideras que debería existir esta opción, ¿cómo la que se muestra en la imagen?



   ◎ Si

   ◎ No

ANEXO VI

Los resultados de la encuesta se encuentran en la siguiente URL:

https://forms.office.com/Pages/AnalysisPage.aspx?id=ak4qaH-nWEmjrJ4mbRiqN-UQtmIJGJFJqmx_RXzPcx1UMDdRWERTN1VJMENESklXMjlQSThFWkVCWi4u&AnalyzerToken=GqShfLnkPveVM8eB8ZSknxEWG5DWjZ34

ANEXO VII

1. Enlace con video del prototipo versión light en iPhone 12 Pro por estudiante 0

   URL: https://youtu.be/WMSoKUNbmLM

2. Enlace con video del prototipo versión dark en iPhone 12 Pro por estudiante 0

   URL: https://youtu.be/GuiisPl1rVE

3. Enlace con video del prototipo versión light en iPhone 11 por estudiante 0

   URL: https://youtu.be/Kk9wyI9cI2M

4. Enlace con video del prototipo versión dark en iPhone 11 por estudiante 0

   URL: https://youtu.be/0-3_I4nyATo

5. Enlace con video del prototipo versión light en iPhone 11 por estudiante 1

   URL: https://youtu.be/szoxVtX0GI4

6. Enlace con video del prototipo versión dark en iPhone 11 por estudiante 1

   URL: https://youtu.be/YFjs8AIi18w

7. Enlace con video del prototipo versión light en iPhone 11 por estudiante 6

   URL: https://youtu.be/Ifkcsi5BGEA

8. Enlace con video del prototipo versión dark en iPhone 11 por estudiante 6

   URL: https://youtu.be/M54PGF6j6WE

ANEXO VIII

El video de la obtención del carnet estudiantil mediante el prototipo de forma rápida se encuentra en la siguiente URL:

https://youtu.be/Er--Lbar1ls

ANEXO IX

Se adjunta el CD que contiene la carpeta `CarnetEPN`, la cual contiene el archivo llamado `CarnetVirtual.xcodeproj` que permite abrir el proyecto en el programa Xcode y a su vez se encarga de enlistar los archivos de la carpeta llamada `CarnetVirtual`.