Covellone, Juan Manuel

Refactorización de un software de estudios clínicos e incorporación de un nuevo módulo

2018

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)
Refactorización de un software de estudios clínicos e incorporación de un nuevo módulo.
Informe Final

DATOS DEL ESTUDIANTE

Apellido y Nombres: Covellone Juan Manuel

DNI: 26.843.665

Nº de Legajo: 3834

Correo electrónico: juancovellone@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 40

PPS enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16. (en caso de ser artículo 7 aclarar en cuál de las dos alternativas posibles se encuadra)

Periodo en que se realizó la PPS: 2018

DOCENTE SUPERVISOR

Apellido y Nombres: Morales Martin

Correo electrónico: martin.morales@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Patricia Medina

Correo electrónico: gaviotadora@gmail.com
medinapatf@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Flux It

Dirección: Camino Parque Centenario 2565. La Plata, Buenos Aires, Argentina.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Teléfono: (+54) 221-5532980 int.344 / 011-36814087
Sector: *Software Factory*

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Martín Saporiti
Correo electrónico: martin.saporiti@fluxit.com.ar

FIRMA DEL COORDINADOR DE LA CARRERA

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Índice

Introducción

Resumen

El presente trabajo, forma parte de las Prácticas Profesionales Supervisadas (PPS[1]) correspondientes a la carrera Ingeniería en Informática de la Universidad Arturo Jauretche (UNAJ).

El mismo tiene como objetivo general realizar modificaciones a un software "Portal Pacientes", el cual tiene como funciones principales el control de turnos e historias clínicas.

Dichas modificaciones se realizarán para poder ofrecer el portal a distintas clínicas. Además, se creará un módulo para que los pacientes puedan obtener datos de un SmartWacht.

Esto permitirá que se puedan observar desde la plataforma web los resultados que proporciona el dispositivo.

Puede observarse que el tema sobre el cual, se desarrollará la PPS, es de total actualidad y con un impacto positivo en la calidad de vida de la sociedad.

Las PPS se llevarán a cabo en la empresa Flux IT en la sucursal central de La Plata.

[1] La Práctica Profesional Supervisada (PPS) es una actividad curricular y en consecuencia obligatoria, en la que el alumno realiza actividades contempladas en los alcances del título y relacionadas con el medio real de desempeño de la profesión.

Objetivos:

Los objetivos se concretarán de la siguiente manera:

En una primera instancia se deberá diseñar el esquema de carpetas, que contengan las vistas de cada módulo de la aplicación (una carpeta por vista para cada módulo).

En una segunda instancia se deberán realizar cambios, en todo el "Modelo Vista Controlador" (MVC), para refactorizar[2] la manera de localizar las vistas de un estilo rígido a un estilo variable, según el cliente al cual se le debe proporcionar la vista.

Para dicho proceso se deberán cambiar las configuraciones dentro de software, no solo en las vistas, sino también del controlador, que debe realizar su tarea según el cliente al cual se le estén proporcionado las vistas.

En una tercera instancia se deberá crear el módulo del SmartWacht, para proporcionar al usuario los siguientes datos:

- Distancia en pasos realizada por el usuario.
- Calorías quemadas en las actividades físicas realizadas en el día.
- Fecha en la cual se realizó la mejor actividad.
- Distancias recorridas y calorías quemadas, desde el día que empezó con el seguimiento de las actividades con su Smartwatch.

Los objetivos que se pretenden lograr ordenados temporalmente son:

✓ Diseñar un esquema de carpetas de vistas.

[2] Refactorizar (del inglés refactoring) es una técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

- ✓ Realizar cambios necesarios en todo el MVC.
- ✓ Optimizar el producto para adecuarse a las necesidades del paciente.
- ✓ Crear el nuevo módulo *Fitbit* que contenga la información requerida.

Tareas

Para llevar a cabo cada etapa de las PPS se realizarán diversas tareas.

En primer lugar, se debe conocer en detalle el software diseñado por el desarrollador.

En segundo lugar, se atenderá a estudiar en detalle el framework utilizado en su diseño.

A continuación, se focalizará sobre la creación de vistas genéricas.

Seguido a ello, la tarea consistirá en modificar los enrutamientos de las vistas.

Inmediatamente lograda la modificación anterior se procederá a cambiar las partes en el controlador de cada módulo.

Finalmente, las condiciones presentaran la oportunidad de agregar un nuevo moduló.

Las acciones especificadas se ven organizadas en el siguiente cronograma.

Cronograma de Trabajo

Tareas	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10
Conocer el software diseñado.										
Estudiar el framework utilizado en su diseño.										
Crear de vistas genéricas										
Modificar los enrutamientos de las vistas										
Cambiar partes en el controlador de cada modulo										
Agregar un nuevo módulo										

(**Cronograma de tareas**, se asume semanas de 20 horas laborales, cumpliendo así en 10 semanas las 200 horas estipuladas para la realización practica de las PPS)

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Desarrollo

La empresa

La PPS se realizan en la empresa Flux IT, en la sucursal situada en Camino Parque Centenario 2565 de la ciudad de La Plata.

Esta compañía está dedicada al desarrollo de software y soluciones informáticas, que se adaptan a las necesidades de cada cliente, basadas en los lineamientos de su arquitectura.

La tarea a realizar en las PPS consiste en modificar un software de turnos clínicos desarrollado anteriormente por la empresa.

Dicha refactorización tiene el propósito de obtener un software que pueda ser presentado a diversas clínicas de una manera más sencilla de exponer, según los logotipos y especificaciones de cada una de ellas.

Este cambio se debe realizar ya que hay otro cliente interesado en este producto y los llamados a los iconos y textos en el código referentes a la clínica están de manera estática, lo que implica que por cada cliente distinto se deban reescribir los archivos que mostrarán al usuario esos datos.

Luego se debe desarrollar e implementar un módulo para visualizar los resultados obtenidos en un *SmartWacht* en el cual, el usuario pueda ver dichos resultados a partir sus ejercicios diarios.

Este módulo le servirá a los pacientes, para tener toda la información de historias clínicas junto con sus datos proporcionados por el dispositivo en un mismo lugar.

Estado Actual del Software

Actualmente, el software se encuentra en un estado funcional y en producción porque ya fue validado por el cliente (Clínica Fleni) y se encuentra en uso.

Dicho software es utilizado para llevar la gestión de pacientes de la clínica mencionada anteriormente para llevar a cabo la gestión de pacientes.

Algunas de sus características son:

a. Permite ver los próximos turnos del paciente.
b. Permite ver el historial de sus consultas médicas.
c. Permite acceder a todos los estudios e imágenes.
d. Permite tener el historial online de informes médicos de la clínica.
e. Permite realizar consultas administrativas (para hacer pedido de recetas y órdenes).
f. Permite compartir acceso al portal de paciente.
g. Permite dejar opiniones sobre el portal o la clínica.

La finalidad de la presente propuesta es, poder ofrecer a distintos clientes, un software que cumpla con las normas de calidad y que se encuentre testeado y en un estado de producción.

Las PPS cumple con varias etapas conceptuales de un proyecto de software: diseñar un módulo, codificarlo, realizar las pruebas de testeo e integrarlo al proyecto.

Este diseño, una vez terminado se pone a disposición de la empresa, su validación e implementación.

Específicamente, las modificaciones a realizar son las de crear las vistas y configuraciones necesarias, para que el software se comporte de manera que, según el cliente solicitante o potencial, el entorno web muestre las

configuraciones específicas o de lo contrario, renderice[3] una configuración genérica, como ser: logos que faciliten la identificación de la clínica, direcciones de la misma y los teléfonos habilitados, entre otras acciones).

Acciones a ejecutar para lograr la propuesta

- Aprender el funcionamiento del software ya diseñado y testeado.
- Estudiar y aprender el *framework* que se utilizó para su desarrollo.
- Crear las vistas para un segundo cliente específico y una tercera vista para ser mostrada a potenciales clientes.
- Modificar los llamados a las vistas, que tendrán que variar según el portal de cada clínica.
- Realizar los cambios, en el controlador de la aplicación.
- Agregar un módulo nuevo, que recibirá datos desde otro portal, para la visualización de ejercicios de los usuarios.
- Realizar ajustes durante todo el proceso del diseño.

[3] Renderización (del inglés rendering) es un anglicismo usado en jerga informática para referirse al proceso de generar una imagen visible e inteligible para el ser humano, a partir de información digital.

Recorrido

1. Conocer el software diseñado

En esta primera tarea, fue necesario reunirse con el desarrollador y empleado de la empresa, Nicolás Romagnoli, quien diseñó y desarrolló el software “Portal salud” a los fines de que explique cómo es el funcionamiento de todo el sistema, para concluir con el diseño y la arquitectura del Portal.

Asimismo, fueron necesarios sus aportes sobre las tecnologías implementadas para su creación y funcionamiento.

Estos datos fueron de mucha utilidad, para poder entender el funcionamiento interno del sitio y luego poder hacer las modificaciones necesarias.

Los datos obtenidos sobre el sistema permitieron conocer como estaba compuesto de lo cual se deduce su arquitectura.

1.1 **El diseño y arquitectura del sistema**

El sistema está compuesto por 3 Aplicaciones. Si bien para la realización de este trabajo solo es necesario centrarse en una sola aplicación, el “*fond-end*”, se debe entender el ciclo completo del manejo de datos.

Estas tres aplicaciones son:

EL “**Front-end**”, el cual tiene la funcionalidad de interactuar con el usuario, mostrándole las vistas. En ellas se puede ver información de usuario guardadas en la base de datos, o enviar le información que se deba guardar.

El “**Back-end**” se encarga de almacenar los datos sensibles de la clínica, los médicos y los pacientes. Aquí se encuentra la base de datos del sistema.

El **“Middleware”** (también llamado lógica de intercambio de información

entre aplicaciones) es un software que asiste a una aplicación (*Back-end*) para interactuar o comunicarse con otras aplicaciones (*Front-end*).

Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones y sincronizaciones que son necesarias en los sistemas distribuidos.

El middleware funciona como una capa de abstracción de software distribuida, en medio de dos aplicaciones para facilitar la comunicación entre las mismas.

Con la visión de estas tres aplicaciones, se puede entender a simple vista como está compuesto el sistema para poder intervenir sobre él.

1.2 **Flujo de datos de las 3 aplicaciones**

Una vez entendido como está separado el sistema en las distintas aplicaciones, es necesario entender como es la comunicación entre dichas aplicaciones. Esta relación se realiza de la siguiente manera:

El *front-end* envía una solicitud de respuesta (*request*) al middleware, pidiendo cierta información.

El middleware recibe la solicitud y envía una consulta (*query*) al *back-end*

El back-end procesa la consulta y devuelve una respuesta de consulta (*query responses*) al middleware.

El *middleware* recibe la respuesta del *back-end* y le proporciona los datos (*data*) al *front-end.*

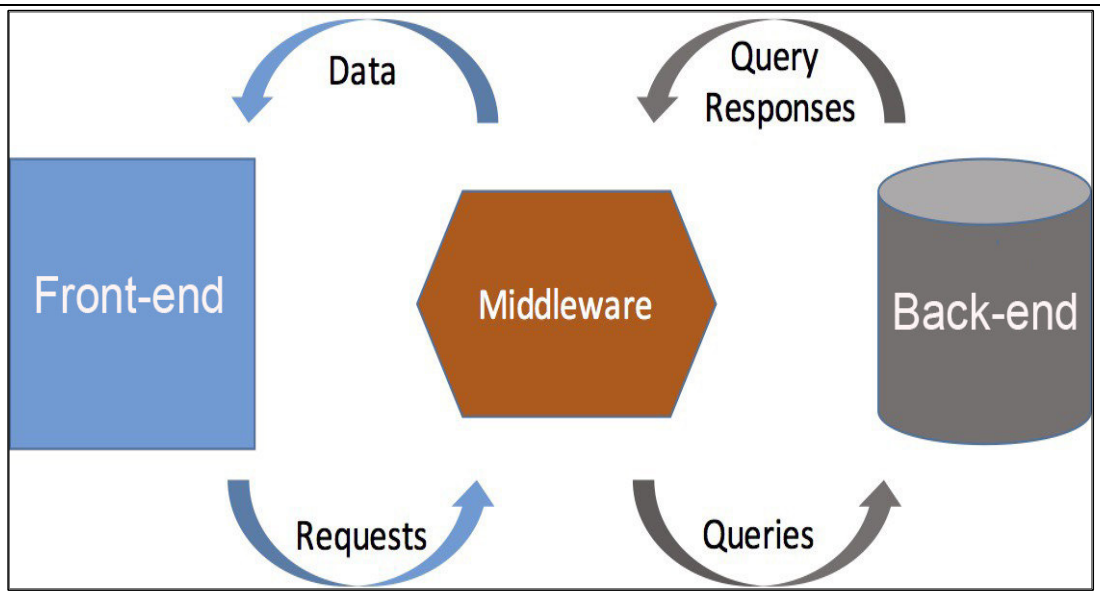


Figura 1 Flujo de datos en las tres aplicaciones.

1.3 **Diseño y arquitectura del Front-end**

Habiendo entendido como es el flujo de datos entre aplicaciones, ahora es posible solo centrarse en el Front-end.

El *front-end* es la aplicación en la cual se centra todo el trabajo de esta PPS, aquí se realizará un detalle más completo de cómo es su diseño y su arquitectura.

El *front-end* está creado utilizando el *Framework* [4] *de Google* llamado "AngularJs" en su versión 1.0.

[4] Framework (ver glosario)

Este *framework* está escrito en el lenguaje de programación JavaScript[5] y utiliza una arquitectura "Modelo – Vista – Controlador" (MVC) que separa los "datos y lógica de negocio" de una aplicación, de su "representación" y el "módulo" encargado de gestionar los eventos y las comunicaciones.

Para ello, MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador (en inglés *model, view, controler)*, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario.

De esta forma la interacción con el usuario se produce de la siguiente manera:

a. El usuario por medio de un navegador web[6] le hace un pedido de solicitud al *controler.*
b. El *controler* recibe el pedido del navegador y genera una comunicación con el modelo, pidiéndole los datos solicitados por el usuario.
c. El modelo recibe la solicitud y hace una consulta a la base de datos (en nuestro caso, le hace una consulta al *middleware*).
d. Cuando el modelo tiene los datos recibidos del *middleware*, se lo envía de al *controler*.
e. El *controler* recibe los datos que le proporciono el modelo, y se los envía a la vista.
f. La vista *renderiza* una página web, mostrando toda la información que le envió el *controler*.

[5] JavaScript es un lenguaje de programación creado con el objetivo de integrarse en HTML y facilitar la creación de páginas interactivas.
[6] La funcionalidad básica de un navegador web es permitir la visualización de documentos de texto, posiblemente con recursos multimedia incrustados. Además, permite visitar páginas web y hacer actividades en ella, es decir, enlazar un sitio con otro, imprimir, enviar y recibir correo, entre otras funcionalidades más.

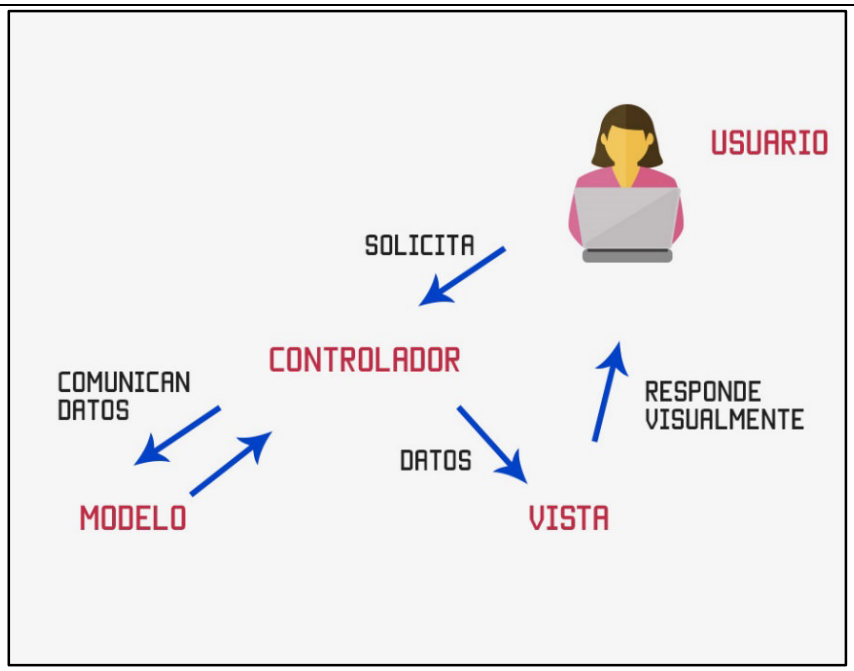


Figura 2. Flujo de datos en Modelo Vista Controlador.

1.4 **Flujo de datos entre el usuario y las tres aplicaciones**

Gracias a esta visión del flujo de datos en mvc, se pudo entender como maneja la información el *front-end.*

Al realizar el reemplazo en el diagrama *"FORT-END" "MIDDLEWARE" "BACK-END"* la aplicación *front-end* por el diagrama "MVC" (utilizado por el "Portal salud") el flujo de datos queda de la siguiente manera:

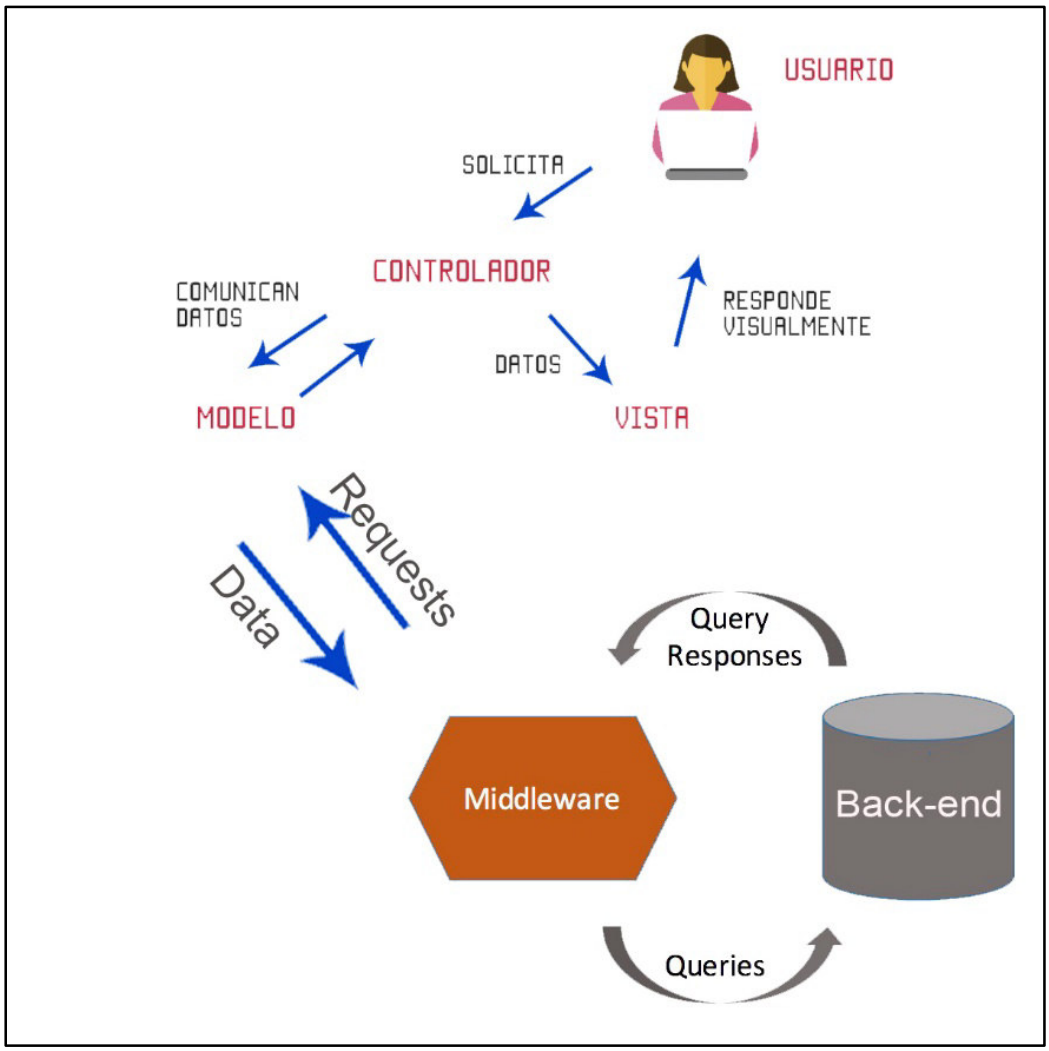


Figura 3. Reemplazo de “Font-end” por “MVC” en diagrama de las tres aplicaciones.

En el diagrama de la figura 3, se puede observar el flujo completo de datos, desde que el usuario hace la petición desde el navegador y como fluye por las tres aplicaciones, hasta que le llega la respuesta solicitada del sistema.

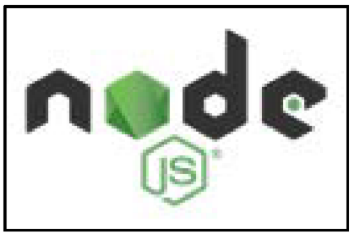
1.5 **Tecnologías utilizadas**

Las tecnologías utilizadas para la realización del *front-end* que se debieron estudiar con el fin de refactorizar el código son:

En orden de prioridad, **AngularJs** que es un framework MVC de JavaScript, para el desarrollo web Front-End. Esta tecnología permite crear aplicaciones SPA "Single Page Applications" o "aplicación de página única", con el propósito de dar una experiencia más fluida a los usuarios, es deciar, como si se tratara de una aplicación de escritorio.

Por ejemplo, en un SPA, todos los códigos de HTML, JavaScript, y CSS[7] se carga de una vez o los recursos necesarios se cargan dinámicamente como lo requiera la página y se van agregando, normalmente como respuesta de las acciones del usuario.

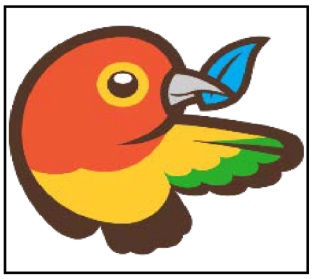
Para un mejor manejo de datos AngularJs, utiliza la tecnología Ajax (ver glosario Ajax) para el envío y recepción de datos, sin tener la necesidad de recargar la página.



En segundo término se encuentra, **NodeJs** que es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Contiene un instalador de paquetes llamado "npm" que se ejecuta desde la línea de comandos y maneja las dependencias para una aplicación. De esta manera con

[7] CSS (siglas en inglés de Cascading Style Sheets), en español "Hojas de estilo en cascada", es un lenguaje de diseño gráfico para definir y crear la presentación de un documento. Es muy usado para establecer el diseño visual de los documentos web.

solo tener instalado NodeJs y escribir "npm install" se instalarán todas las dependencias que necesita el software y que fueron configuradas previamente en un archivo llamado "package.json".

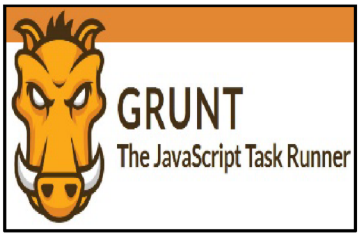


Otra de las tecnologías es, **Bower** es un gestor de dependencias enfocado en proyectos *front-end*. Cuando necesitamos una dependencia podemos pedirle a Bower simplemente que la instale.

Como primer paso, debemos instalar Bower mediante npm, el gestor de dependencias de NodeJS, que es similar que Bower, pero para paquetes de Node y Javascript en general.

Se instala utilizando NodeJS con el comando "npm install bower" para luego con solo ejecutar el comando "bower install" se instalarán las dependencias que la app requiera (las dependencias deben estar configuradas en su respectivo archivo bower.json).

Finalmente, **Grunt** es una librería JavaScript que nos permite configurar tareas automáticas y así ahorrarle tiempo en el desarrollo y despliegue de aplicaciones webs. Con un simple fichero JS llamado "Gruntfile", se indican las tareas que se quieren automatizar con un simple comando.



Estas tareas están escritas en un archivo de formato Json (ver glosario Json).

Una vez entendida las tecnologías utilizadas por el *front-end* y el flujo de datos de todo el Portal Salud, se puede tener una noción completa de que se debe realizar para poder refactorizar el código y luego agregar el nuevo módulo.

2. Estudio del framework utilizado en su diseño

Esta es la tarea que más tiempo demando. Inicialmente se realizó un estudio detallado del framework, para luego seguir consultando el libro durante todos los pasos siguientes. La consulta del libro fue permanente durante el proceso.

Para poder crear o modificar una aplicación Angular, se debe tener conocimientos profundos del framework, ya que cada acción que se esté realizando o modificando, sea en el modelo, la vista o el controlador, es necesario contemplar que existen acciones entrelazadas entre sí.

Para esta tarea, se debió leer la documentación oficial del sitio web del desarrollador de Angular u otra documentación con explicaciones avanzadas.

En este caso se utilizó la segunda opción, eligiendo como libro de estudio "Angular paso a paso".

Este es un libro online en formato pdf, escrito en español con 218 páginas de explicaciones y ejemplos. Para poder entender el código fuente de la aplicación "Portal Salud" esta lectura resulta de suma importancia ya que aporta las ideas elementales para orientar las intervenciones.

3. Creación de vistas genéricas

En esta instancia se crearon tres copias de la carpeta "views" contenida en cada modelo. Esta carpeta contiene todas las vistas de cada módulo. En

Principio se crearon tres copias: una con el nombre “views-fleni”, otra con el nombre “views-orono” y una última con el nombre “views-generic”. De esta manera se contempló que si es necesario hacer un cambio para un cliente, este cambio no perturbe en la vista de los otros clientes.

En la figura 4 se describe el movimiento de cambio de las carpetas que se agregaron para cada moduló.

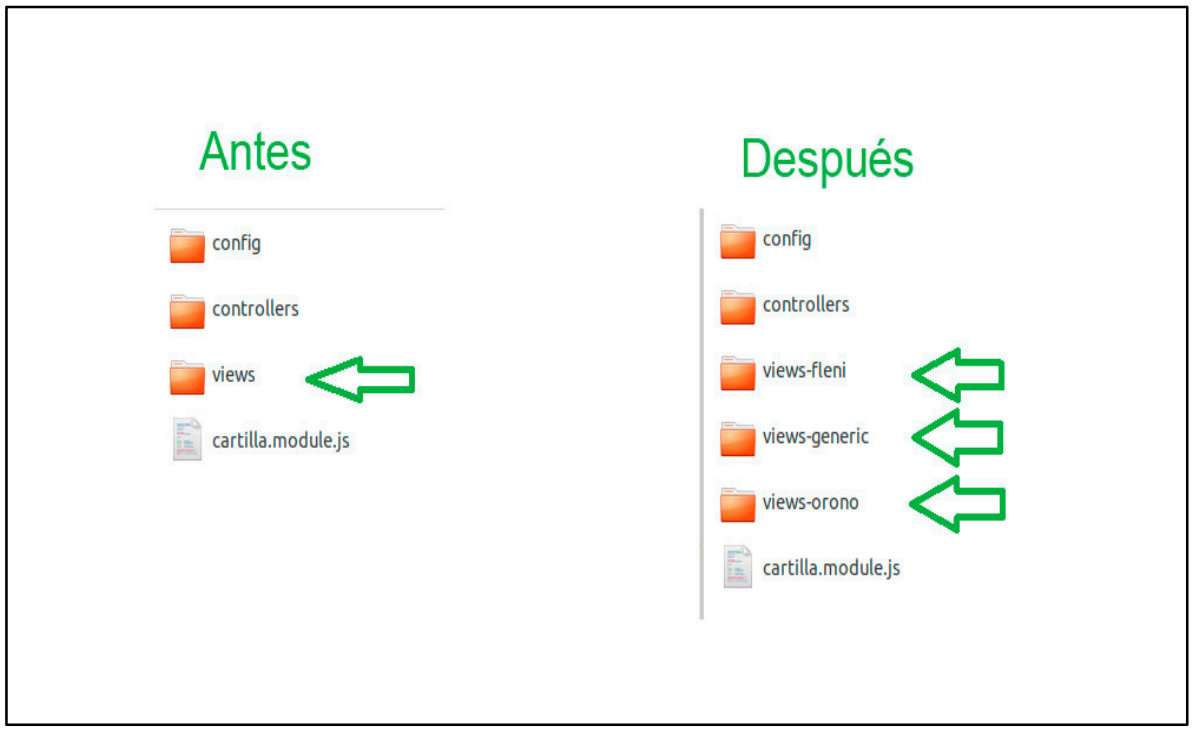


Figura 4. Agregación de carpetas y templetes en cada módulo.

Luego se crearon las respectivas carpetas e incluyeron todos los archivos que contenga la carpeta views.

Aquí se debió revisar cada una de las Tempates (Plantilla que utiliza el framework para crear los archivos HTML (ver glosario HTML)) correspondientes a cada

vista, dado que la gran mayoría incluye dentro del código llamadas a *templates* que se encuentran en subcarpetas.

La refactorización del código se llevó acabo de la siguiente manera:
Según el libro online AngularJs paso a paso se indica que a la hora de hacer la maqueta de la aplicación quizás se necesite tener parte de la aplicación que se usarán en varias plantillas.

La directiva ng-include puede solucionar ese problema permitiendo que se extraiga la porción de la plantilla que será reutilizada a un archivo diferente y luego con el uso de la directiva, se incluirla en varios lugares de la aplicación. Finalmente, la directiva debe recibir el URL de la plantilla para ser cargada.

Sobre la base de lo expuesto por el libro se deberá buscar en todos los archivos HTML la línea donde esté escrito:

"<div ng-include="

Ejemplo
** **

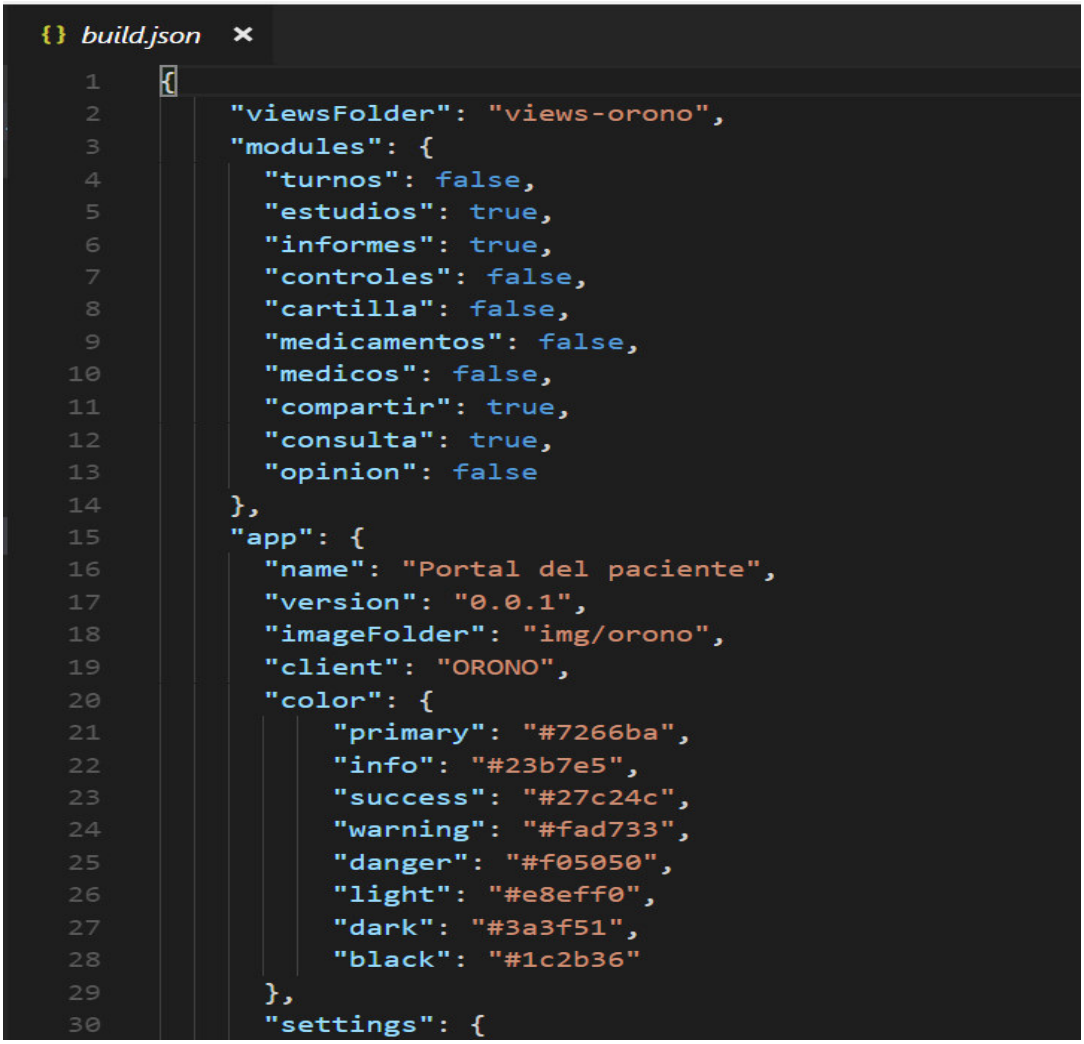
Una vez encontrada una línea con "ng-include" se reemplazó la palabra views (que hace referencia a la antigua carpeta views), por la variable global "viewsFolder" la cual hace referencia al views del cliente que se esté ejecutando. Lo explicado anteriormente quedó de la siguiente manera:

** **

De esta forma, la aplicación reemplazo ' + viewsFolder + ' por la carpeta views del cliente.

Para que esta variable global tome el valor adecuado, se debió cambiar la configuración del archivo "build.json" que se halla ubicado dentro de la carpeta "environments" del ambiente de desarrollo. Para esto se debió contar con un archivo build.json para cada cliente).

En la figura 5 se visualiza el archivo de configuración build.json para ser utilizado por cada cliente.



```
{} build.json ×
{
    "viewsFolder": "views-orono",
    "modules": {
      "turnos": false,
      "estudios": true,
      "informes": true,
      "controles": false,
      "cartilla": false,
      "medicamentos": false,
      "medicos": false,
      "compartir": true,
      "consulta": true,
      "opinion": false
    },
    "app": {
      "name": "Portal del paciente",
      "version": "0.0.1",
      "imageFolder": "img/orono",
      "client": "ORONO",
      "color": {
          "primary": "#7266ba",
          "info": "#23b7e5",
          "success": "#27c24c",
          "warning": "#fad733",
          "danger": "#f05050",
          "light": "#e8eff0",
          "dark": "#3a3f51",
          "black": "#1c2b36"
      },
      "settings": {
```

Figura 5. Fragmento de archivo de configuración “build.json”

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

4 Modificar los enrutamientos de las vistas

Una vez creadas todas las carpetas y *refactorizados* los archivos html que así lo requerían, se focalizó sobre las correcciones a los archivos "routes.js" de cada módulo.

Estos archivos hacen referencia a donde la aplicación debe buscar los *templates* correspondientes para cada vista del módulo.

Dichas modificaciones se realizaron de la siguiente manera: En primer lugar, se debió proporcionar una variable más a la función que se estaba creando. Esta será una nueva variable llamada 'BUILD' la cual fue creada en el archivo "build.json".

La variable Build es una variable global de la APP, es decir la misma toma valor cuando se ejecuta la aplicación.

En segundo lugar cuando se hizo referencia a la URL del *template*, se debió reemplazar el texto donde se encontraba escrita la palabra "*views*" y concatenar[8] el texto restante con la variable "BUILD" con la finalidad de que devuelva el atributo "*viewsFolder*"

La figura 6 muestra la imagen del repositorio donde se puede apreciar los cambios realizados.

[8] En informática, concatenar es la operación por la cual dos caracteres se unen para formar una cadena de caracteres (o string). También se pueden concatenar dos cadenas de caracteres o un carácter con una cadena para formar una cadena de mayor tamaño.

```
...  ...  @@ -8,19 +8,19 @@
 8    8    */
 9    9   angular
10   10       .module('app.cartilla')
11        -  .config(['$stateProvider',
12        -      function($stateProvider) {
      11   +  .config(['$stateProvider', 'BUILD',
      12   +      function($stateProvider, BUILD) {
13   13
14   14           $stateProvider
15   15               .state('app.main.cartilla', {
16   16                   abstract: true,
17   17                   url: '/cartilla',
18        -                  templateUrl: 'js/modules/cartilla/views/cartilla.html'
      18   +                  templateUrl: 'js/modules/cartilla/'+BUILD.viewsFolder+'/cartilla.html'
19   19               })
20   20
21   21               .state('app.main.cartilla.buscar', {
22   22                   url: '/buscar',
23        -                  templateUrl: 'js/modules/cartilla/views/buscar.html',
      23   +                  templateUrl: 'js/modules/cartilla/'+BUILD.viewsFolder+'/buscar.html',
24   24                   controller: 'BuscarCartillaCtrl'
25   25               });
26   26       }
```

Figura 6. Fragmento de “gitlab” que muestra los cambios del archivo routes.js

Vista de uno de los archivos routes.js (en este caso se ve que es del módulo “Cartilla”.

En la imagen se visualiza en rosa lo que se quitó y en verde lo que se agregó. De esta manera vemos que en las líneas 11 y 12 a la función se le envía la variable global ‘BUILD’.

En la línea 18, se observa que antes se estaba llamando a una url estática ‘js/modules/cartilla/views/cartilla.html’, sin embargo, una vez realizada la corrección se le pasa una url dinámica que va a variar según el argumento que ingrese en la variable ‘BUILD’, la cual permita que quede de la siguiente manera:

'js/modules/cartilla/ + BUILD.viewsFolder +/cartilla.html'.

Por consiguiente, el ejemplo se envía el argumento *generic.* La aplicación buscará el archivo de configuración y reemplazará "+ BUILD.viewsFolder +" por "views-generic"

Ejecución de la APP

La ejecución de la app, para iniciar el portal según el cliente deseado se realiza de la siguiente manera:

"grunt serve --target=test --client=generic".

En la ejecución anterior donde se observa el argumento *"target*" se le está indicando cual es el servidor que debe conectar la app. En este caso se está dirigiendo al servidor de testing.

En el argumento "client", se está enviando, qué cliente debe *renderizar.* En este caso se le está diciendo que rendirse la app *generic*. Este último argumento es el que tomará la variable global 'BUILD' la cual será utilizado para el archivo routes.js

5. Realizar cambios en el controlador de cada módulo

Para que el controlador encuentre las carpetas que debe renderizar la vista, se le debió refactorizar el llamado a los templates de la misma manera que se realizó en los archivos "router".

Esta acción cambio la ubicación de la carpeta, en la cual se reemplazó la palabra 'views' que correspondía al path[9] de la ubicación de los témplate, por una variable global ' $scope.viewsFolder '. Ésta contiene la ubicación según el cliente que se esté ejecutando.

En la imagen se ve que el remplazo en la línea 98, en el cual se tuvo que concatenar la variable para realizar el path donde se encuentran los *templates.*

```
...  ...  @@ -95,7 +95,7 @@ angular
95   95                      sharedComment: $scope.form.mensaje
96   96                  }).then(function(response){
97   97                      $scope.dialog = ngDialog.open({
98       -                      template: 'js/modules/compartir/views/partials/modal-ok.html',
     98  +                      template: 'js/modules/compartir/'+$scope.viewsFolder+'/partials/modal-ok.html',
99   99                      className: 'ngdialog-theme-default',
100  100                     appendClassName: 'ps-modal',
101  101                     scope: $scope,
```

Figura 7 Fragmento de "gitlab" que muestra los cambios en archivo controler.js

Resultados de la refactorización

En las siguientes imágenes se pueden observar cómo cambian los logos y la información de cada Login según el cliente que se esté ejecutando.

Cuando ejecutamos el cliente Fleni se pudieron observar los cambios pensados: logo, color de fondo, ubicación clara de los números telefónicos, entre otras cosas.

grunt serve --target=test --client=fleni

[9] En informática, una ruta (path, en inglés) señala la localización exacta de un archivo o directorio mediante una cadena de caracteres concreta.

Figura 8. Vista del Login original (cliente Fleni)

Al ejecutar el cliente Oroño se puedo visualizar con mayor claridad la efectividad de la refactorización del código.

grunt serve --target=test --client=orono

Figura 9. Login cliente Oroño

Finalmente, al ejecutar el cliente genérico, se puedo testear que el código se comportaba de la manera esperada.

grunt serve --target=test --client=generic

Figura 10. Login genérico.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

6. Incorporación de un nuevo módulo

El nuevo módulo creado llamado “Fitbit”, tiene la funcionalidad de representarle a los pacientes los datos recolectados por un SmartWacht que debe tener el paciente colocado en su muñeca. De esta manera, el paciente podrá observar: distancias recorridas en el día, calorías quemadas por su actividad, ritmo cardiaco, entre otras actividades físicas.

Para que el nuevo módulo pueda recibir los datos proporcionados por el smartwatch. El tutor en Flux IT César Vargas realizó las conexiones a la API (ver glosario API) de Fitbit.

Esta conexión permitió recibir los datos enviados desde el dispositivo a la base de datos de la empresa. La tarea realizada es importante ya que permitió tener acceso a los datos del teléfono.

6.1 Ciclo del nuevo módulo

6.1.1 Requerimientos

En esta etapa se tomaron los requerimientos de alto nivel que consisten en tener en claro que se debe hacer, para poder realizar el nuevo moduló. Aquí, personal de la empresa Flux IT, comunico al tutor y posteriormente éste al alumno, que debería hacer el nuevo módulo.

Los requerimientos fueron los siguientes:

- El nuevo módulo debe recibir los datos de un SmartWacht y proporcionar le al paciente que lo utiliza, los datos dentro de la app.
- Se debe proporcionar los datos tanto en español como en inglés.
- El nuevo módulo debe mostrase integrado a la app, donde los estilos sean los mismos, utilizados en todo el proyecto.

6.1.2 Análisis:

En esta etapa se revisaron los requerimientos componiéndolos como un todo coherente.

Esta revisión arrojo como resultado la confirmación de la inexistencia de carencias o inconsistencias en el pedido a ejecutar.

6.1.3 Diseño:

En esta instancia se estudiaron los diferentes componentes tecnológicos y cómo se iba a interactuar entre ellos.

En esta tarea se tuvo que investigar además de la API proporcionada por la empresa del reloj, la documentación de AUTH0 (ver glosario Auth0) para poder realizar el login a la plataforma, dado que Fitbit utiliza esto último para autentificar a los usuarios.

En consecuencia, se decidió crear un middleware con el lenguaje de programación JAVA, el cual hiciera las conexiones al API de la empresa del reloj, autentifique a los usuarios vía AUTH0 y devuelva un Json

6.1.4 Codificación:

En esta etapa se implementaron las diferentes funcionalidades por medio de lenguajes de programación o parametrización de paquetes o sistemas preexistentes. Es decir, en algunos sectores de la codificación se programó y en otras se utilizaron librerías de terceros que solucionaron dificultades sobre dicha programación.

Los pasos del proceso de codificación fueron los siguientes:

En una primera instancia, se creó la carpeta del nuevo módulo llamada Fitbit.

En una segunda instancia, se creó un usuario en la plataforma del SmartWacht y se debió utilizar el reloj inteligente, durante varios días para tener datos de prueba.

En una tercera instancia se crearon los servicios (que cumplen con la función de ser el modelo un MVC).

La finalidad de esa creación es obtener los datos de la API de Fitbit.

Para esto, se debió contar con el middleware creado por el tutor César Vargas y utilizando Ajax.

Figura 11. Imagen de un SmartWacht

Se realizaron las consultas a la plataforma de la empresa del SmartWacht, los datos recolectados anteriormente por el dispositivo para luego poder retornar estos datos a los controladores del nuevo módulo.

En esta instancia se crearon los controler (controladores) que obtuvieron los datos del modelo, los cuales fueron enviados a la vista a través del objeto "$scope" que nos proporciona angular.

En esta última instancia se crearon las vistas. En este caso solo se creó la carpeta de "views-generic" dado que solo se está creando un módulo que no fue pedido por los clientes Fleni y Orono, sino que es un módulo que se le va a ofrecer a dichos clientes o próximos en una nueva versión de la aplicación.

6.1.5 Pruebas

En esta última etapa se debió confirmar que lo desarrollado en el módulo Fitbit no contenía errores, esta es una prueba unitaria. Además, se debió comprobar que el diseño era completo y consistente, a lo cual llamamos prueba de integración. Finalmente, en la prueba funcional y de usabilidad se comprobó que los requisitos y objetivos se cumplieran.

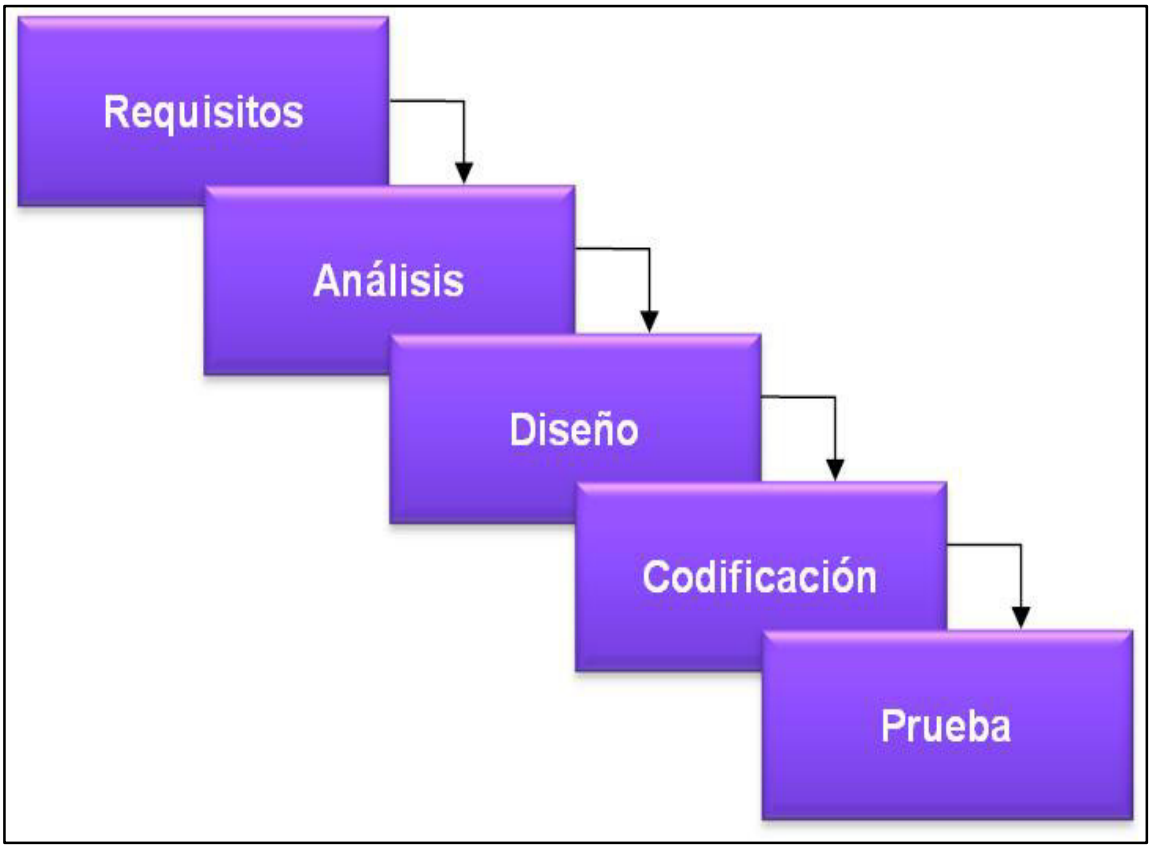


Figura 12. Ciclo de vida del nuevo módulo.

Resultados del nuevo módulo

Figura 13 Primer prototipo del módulo Fitbit

Conclusión

La realización de este trabajo permitió tener noción real de lo que significa el ciclo de vida del software y cómo se dividen las tareas.

El llevar adelante esta PPS fue una experiencia con la cual pude comprobar algunas cuestiones.

En primer lugar, comprendí que cuando vemos un código realizado por otras personas, quienes por lo general son profesionales podemos notar que son software de gran cantidad de archivos.

Estos archivos se van incrementando de a poco y cada etapa del ciclo de vida del software no es algo que va ocurriendo a una velocidad extrema, sino que cada momento se debe planificar, ejecutar y validar con tiempos acordes. En este caso lo pude comprobar al instante de crear el nuevo módulo.

En segundo lugar, fue interesante ver como para realizar un proyecto nos valemos de muchas librerías y framework. Si bien esto lo sabemos por la utilización permanente durante la cursada de la carrera en la universidad, la realidad es que en un proyecto real se utilizan gran cantidad de librerías para cada proyecto y se deben tener en cuentas las versiones de las misma, dado que luego si queremos re crear un ambiente, debemos utilizar las que fueron puesta en juego durante el desarrollo.

En tercer lugar, este proceso me permitió advertir que durante las cursadas no nos damos cuenta de estas cuestiones dado que nuestros proyectos son de poca duración y no sufrimos cambios de versiones de librerías.

En cuanto a la programación, se pudo aprender no solo lenguajes o librerías, lo más importante fue poder ver y entender un software creado por un profesional y poder aprender cómo debe ser más prolijo a la hora de codificar, lo cual no significa que no aprendamos a ser prolijos durante la cursada en la facultad, s no que se aprende a realizar una mejor calidad de código.

Concluyendo, El trabajo fue muy positivo, ya que pude consultar las dudas y los inconvenientes que iban surgiendo a profesionales y ellos estaban predispuestos, pero además comentaban que uno no tiene que saber todo en esta profesión para poder ser un buen desarrollador, sino que debe valerse de todo lo que pueda para resolver los problemas a los cuales se debe hacer frente. No obstante, destaco que no está mal consultar a un compañero de equipo si éste ya se enfrentó con ese problema antes y sabe cómo resolverlo.

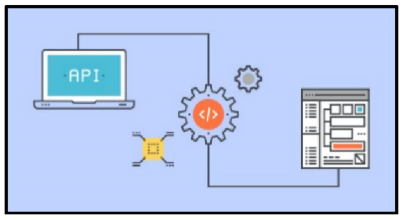
Para finalizar, quedo muy satisfecho por poder haber tenido la posibilidad de aprender de esta manera una de las últimas materias de mi carrera. Me quedo con muchas cosas nuevas que me serán de mucha ayuda al momento de poder trabajar de forma profesional.

Glosario

Ajax: acrónimo de "Asynchronous JavaScript And XML" (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas.

Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador web de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano.

De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones. (volver al texto)



API: proviene de las siglas en inglés "application programming interface", que en español significan "interfaz de programación de aplicaciones".

El concepto hace referencia a los procesos, las funciones y los métodos que brinda una determinada biblioteca de programación a modo de capa de abstracción para que sea empleada por otro programa informático. (volver al texto)

Auth0 Proporciona una API Conformada por desarrolladores e ingenieros con sólida trayectoria en seguridad informática, para dar

una solución a la implementación de autentificación en aplicaciones de manera segura. (volver al texto)

Framework: En el desarrollo de software, un entorno de trabajo (Framework) es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software.
Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. (volver al texto)

Json: acrónimo de *JavaScript Object Notation,* es un formato de texto ligero para el intercambio de datos. (volver al texto)

HTML: sigla en inglés de *Hyper Text Markup Language* (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web.

Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la *World Wide Web* (WWW). (volver al texto)

Video Portal Salud

https://www.youtube.com/watch?time_continue=4&v=ZxK7ei52ViQ

Bibliografía

Libro online: AngularJs paso a paso
http://coreditec.com.co/libros/angularjs-paso-a-paso-pdf.pdf

Creación de módulos AngularJS
AngularJs paso a paso página 51

Creación de servicios AngularJs
AngularJs paso a paso página 56

Directiva AngularJs “ng-include”
AngularJs paso a paso página 107

Rutas AngularJS “ng-Route”
AngularJs paso a paso página 137

Documentación oficial de Auth0
https://auth0.com/docs/quickstart/spa/vanillajs

Fitbit oficial para desarrolladores
https://dev.fitbit.com/

Grunt documentación oficial
https://gruntjs.com/getting-started

NodeJS documentación oficial
https://nodejs.org/es/docs/

Desarrollo web

https://www.w3schools.com/

Bower documentación oficial

https://bower.io/

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

REFLEXIÓN SOBRE LA PRÁCTICA PROFESIONAL SUPERVISADA COMO ESPACIO DE FORMACIÓN

El área en que se realizaron las Prácticas Profesionales Supervisadas, estuvo dividida en dos partes: en una primera instancia relacionada a la refactorización de un software para luego culminar con la creación de un módulo e integrarlo al mismo

En primera instancia se pudo tener conocimientos de lo que es estar trabajando en un proyecto real, donde los roles están divididos y cada integrante del equipo debe cumplir uno específico, sean desarrolladores, arquitectos, responsables de usabilidad o diseño.

En la Universidad Nacional Arturo Jauretche, el trabajo se plantea distinto. Si bien se aprende de forma teórica, lo cual es adecuado y necesario hacerlo así, todos los integrantes de un proyecto participamos de cada tarea, dado que los proyecto son de tamaño más pequeño y los tiempos son distintos.

Con el trabajo realizado en la empresa, he aprendido, que en proyectos profesionales, se utilizan muchas librerías y framework de tercero (llámese “de terceros” a desarrolladores ajenos a la empresa, que publican sus librerías o framework, para que otros desarrolladores los puedan utilizar) si son de procedencia confiables.

En cambio, durante las cursadas se utilizan pocos Framework. Esto se debe más que nada a que uno como estudiante primero debe aprender cómo resolver los inconvenientes desde lo básico, para entender bien el funcionamiento de lo que se esté realizando.

En la segunda instancia de las PPS, se realizó la creación de un módulo, siguiendo los procedimientos del ciclo de vida del software, pasando desde la toma de requerimientos, análisis de los mismos, diseño, programación y pruebas, basándonos en los conocimientos aprendidos en la primera etapa. De esta manera se entendió de forma práctica como se debe realizar el ciclo de vida de un software.

Cabe mencionar que muchos de los incidentes que surgieron durante toda la práctica, se debieron a la gran cantidad de librerías que se utilizaron en el proyecto “Portal Salud” y en especial al framework Angular JS. Este último tenía una curva de aprendizaje muy grande y se debía aprender para poder refactorizar y poder crear el nuevo módulo.

Si bien la mayoría de inconvenientes de las librerías fueron resueltos con un simple acceso al sitio oficial de las librerías o con ayuda del desarrollador o el tutor de Flux IT, con el Framework se debió recurrir a un libro digital que contenía toda la documentación de Angular JS dado que al momento de realizar las PPS este Framework ya había evolucionado y la documentación que se obtenían en la mayoría de los sitios era la de la nueva versión siendo esta incompatible con su predecesor.

Otro inconveniente que surgió fue en el momento de tener que obtener los datos de la API del SmartWacht, dado que se adquirían los datos desde un entorno de desarrollo, pero el momento de obtenerlos desde la aplicación, ésta retornaba un error de autentificación. Este error se debía a que se realizaba una petición AJAX pero no se enviaban todos los datos que se requerían para autentificarme.

Dadas estas circunstancias, los tiempos estipulados para realizar las PPS (200 horas) fueron excedidos, pues había que poder realizarlas en su totalidad. La mejor decisión fue que la tarea se dividió en dos etapas: una de mucho estudio y aprendizaje dado que eran muchos temas nuevos y otra de codificación, siendo estas últimas guiadas por el tutor de la empresa.

Hoy, a la distancia y con la tarea ya cumplida, si tuviera que realizar de nuevo este recorrido, el principal cambio que haría, seria no subestimar nada cuando un profesional nos da a elegir una de las dos tareas y nos advierte que una es mucho más difícil que la otra. Esto me sucedió cuando el primer día se me dio a elegir trabajar en el back-end o en el front-end y se me advirtió que para trabajar en el front-end de la aplicación debía aprender un framework AngularJS que tenía una curva de aprendizaje muy grande y lo tomé como un pequeño desafío, resultando ser más complicado de lo que supuse sin saber.

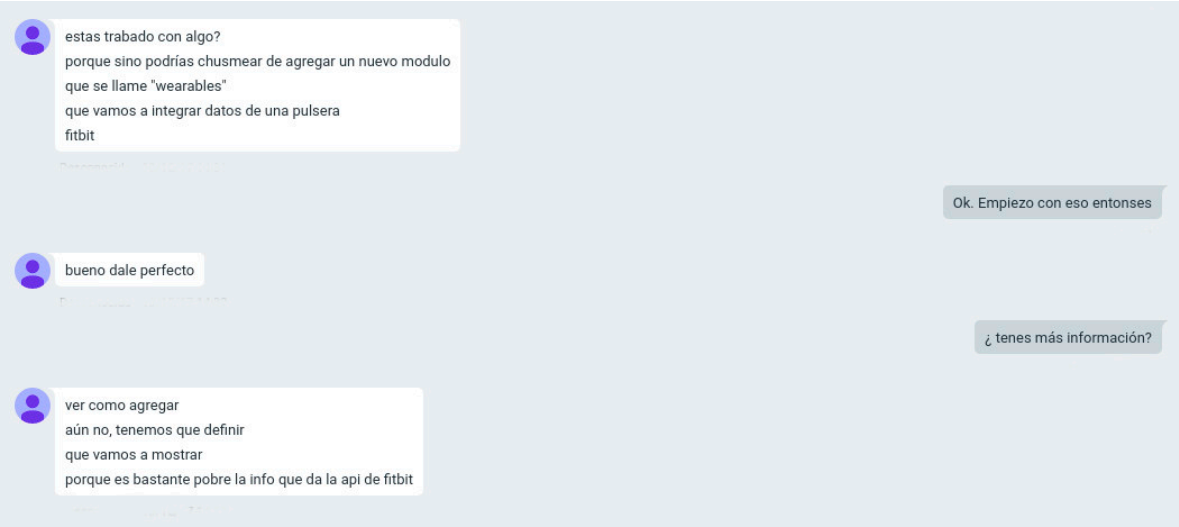
Como conclusión, las PPS son muy importantes para el alumno a punto de culminar su carrera, dado que se terminan de aprender conceptos, que los profesores enseñan de forma teórica, pero la mejor forma de aprenderlos es de forma práctica.

Finalmente, me queda una enseñanza muy grande entender que por más que se trabaje de forma profesional, siempre se puede refactorizar un software para mejorarlo

Anexo

Primera conversación del requerimiento del nuevo módulo.

Módulo Fitbit, primera conversación, donde todavía no se habían definido lo que se iba a mostrar al usuario.



Agradecimientos

Martín Morales director del Instituto de Ingeniería en Informática y docente supervisor por la Universidad Arturo Jauretche de estas PPS.

Gracias por el apoyo continuo durante todo el proceso de la pps, desde el principio buscándome una empresa en la cual pudiera realizar las, como también acompañando durante toda la realización de la misma.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Patricia Medina docente tutora del Taller de Apoyo a la Producción de Textos Académicos de la UNAJ que forma parte del equipo Tapta.

Gracias por el acompañamiento durante todo el proceso de textos corrigiéndome los errores o sugiriéndome maneras más correctas de expresar lo que intentaba redactar.

Cesar Vargas arquitecto de software y tutor en Flux IT

Gracias por guiarme en esta etapa, por enseñarme como resolver los problemas que se me presentaron y siempre estar predispuesto a ayudar me si no entendía algo.

Nicolás Romagnoli Desarrollador de software en Flux IT.

Gracias por explicarme detalladamente cómo funciona el software "Portal Salud" y además siempre estar predispuesto a ayudar me cuando no entendía alguna funcionalidad del software.

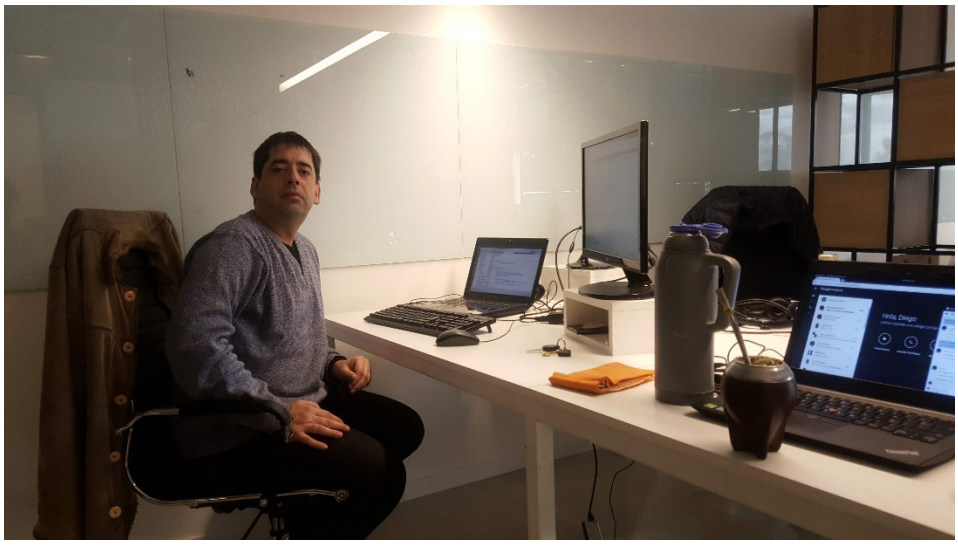
Martin Saporiti Manager y coordinador de desarrolladores en Flux IT**.**

Gracias por recibirme y darme la oportunidad para que personal de tu equipo me guie en estas PPS.

Flux IT Empresa dedicada al desarrollo de software y soluciones informáticas.

Gracias por abrirme las puertas de tu casa y darme la posibilidad de poder terminar de formarme para esta profesión. Estoy muy agradecido de que desde el primer día Flux IT como todo su personal me hicieron sentir que era una más de ustedes, en auxiliarme en cosas como darme un equipo informático desde el primer día sin conocerme, para que me lo pueda llevar a mi casa y pueda trabajar. Siendo yo un pasante, me dio la confianza que estaba en el lugar adecuado para terminar mi formación.

Fotografías



Desarrollando en las instalaciones de Flux It.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Junto al tutor de Flux It Cesar Vargas.

Con Patricia Medina, Tutora del taller de textos de la Unaj

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Junto al manager de desarrolladores, Martin Saporiti.

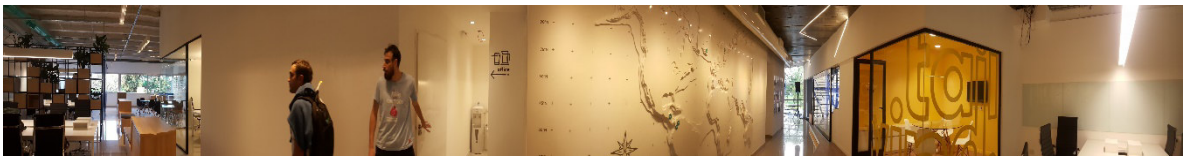


Foto panorámica de las instalaciones de Flux It

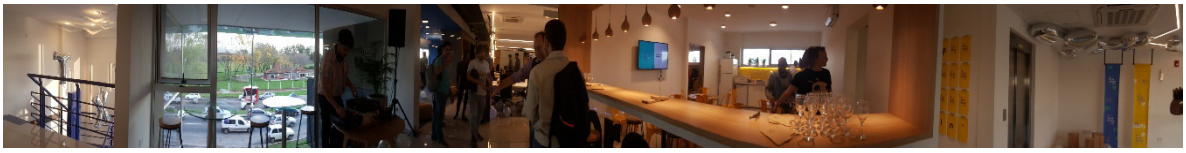


Foto panorámica de las instalaciones de Flux It

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

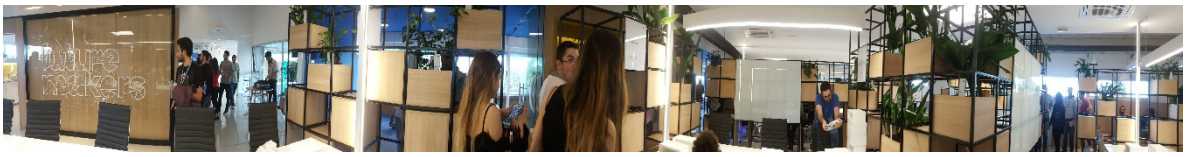


Foto panorámica de las instalaciones de Flux It

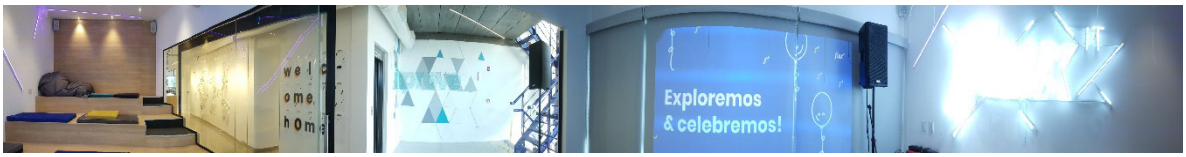


Foto panorámica de las instalaciones de Flux It

Foto panorámica de las instalaciones de Flux It

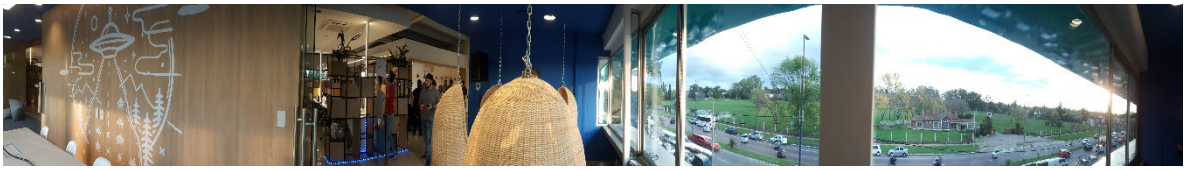


Foto panorámica de las instalaciones de Flux It

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional: