



UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL MENDOZA

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

PROYECTO FINAL

Carpeta de salud personal

Integrantes:

32141 Franco Nicolás CANIZO
33485 Michael Jonathan MANGANIELLO
31904 Yanina Graciela MORALES
33183 Milton Iván TERRENO

Cuerpo docente:

Alejandro VAZQUEZ
Raúl MORALEJO
Gustavo MANINO
Diego VILLA

24 de octubre de 2015

Índice

1. Sprint 2	6
1.1. Planificación	6
1.2. Descripción	6
1.3. User Stories relacionados	6
1.4. Modelo de datos.	7
1.5. Descripción de las Clases	8
1.5.1. Clase Measurement	8
1.5.2. Clase MeasurementType	9
1.5.3. Clase MeasurementUnit	10
1.5.4. Clase MeasurementSource	11
1.6. Modelo funcional.	11
1.6.1. Creación de página de mediciones	12
1.6.2. Creación de página de formulario de carga de mediciones . . .	14
1.6.3. Creación de página de formulario de edición de mediciones . .	15
1.7. Salidas del Sistema - Incrementos	16
1.8. Criterios de aceptación	17
1.9. Casos de Prueba	18
1.9.1. Pruebas de integración entre módulos del Sistema	29
1.9.2. Pruebas de carga	29
1.9.3. Pruebas de seguridad por niveles de usuarios	29
1.10. Pruebas ejecutadas	29
2. Sprint 3:	30
2.1. Planificación	30
2.2. Descripción	30
2.3. User Stories relacionados	30
2.4. Modelo funcional.	31
2.4.1. Validadores	31
2.4.2. Filtrado de mediciones	32
2.5. Criterios de aceptación	33
2.6. Pruebas ejecutadas	34
2.7. Clases involucradas	40
2.8. Estado final de pruebas	42

3. Sprint 4	43
3.1. Planificación	43
3.2. Descripción	43
3.3. User Stories relacionados	43
3.4. Modelo de datos	43
3.5. Modelo funcional.	44
3.5.1. Definición de modelos	44
3.5.2. Recursos	45
3.6. Salidas del Sistema	45
3.7. Criterios de aceptación	47
3.8. Casos de prueba	48
3.8.1. Pruebas de integración entre módulos del Sistema	48
3.8.2. Pruebas de carga	48
3.8.3. Pruebas de seguridad por niveles de usuarios	48
3.9. Pruebas ejecutadas	48
4. Sprint 5	48
4.1. Planificación	48
4.2. Descripción	49
4.3. User Stories relacionados	49
4.4. Modelo de datos	49
4.5. Modelo funcional.	49
4.6. Salidas del Sistema - Incrementos	50
4.7. Criterios de aceptación	54
4.8. Casos de prueba	54
4.8.1. Pruebas de integración entre módulos del Sistema	54
4.8.2. Pruebas de carga	58
4.8.3. Pruebas de seguridad por niveles de usuarios	58
4.9. Pruebas ejecutadas	58
5. Sprint 6	59
5.1. Planificación	59
5.2. Descripción	59
5.3. User Stories relacionados	59
5.4. Modelo de datos	59
5.5. Modelo funcional.	60
5.5.1. Estructura conexión con diferentes medios de almacenamiento	61
5.5.2. Creación de clases y funcionalidades para la gestión de archivos	62

5.5.3.	Creación de la migración para los nuevos modelos	63
5.5.4.	Creación de fields y parsers para las representaciones de los nuevos recursos	63
5.6.	Salidas del Sistema - Incrementos	64
5.7.	Criterios de aceptación	66
5.8.	Casos de prueba	66
5.8.1.	Pruebas de integración entre módulos del Sistema	66
5.8.2.	Pruebas de carga	66
5.8.3.	Pruebas de seguridad por niveles de usuarios	66
5.9.	Pruebas ejecutadas	66
6.	Sprint 7	67
6.1.	Planificación	67
6.2.	Descripción	67
6.3.	User Stories relacionados	67
6.4.	Modelo de datos	67
6.5.	Modelo funcional.	68
6.5.1.	Tarea a describir	68
6.6.	Salidas del Sistema - Incrementos	68
6.7.	Criterios de aceptación	68
6.8.	Casos de prueba	69
6.8.1.	Pruebas de integración entre módulos del Sistema	69
6.8.2.	Pruebas de carga	69
6.8.3.	Pruebas de seguridad por niveles de usuarios	69
6.9.	Pruebas ejecutadas	69
7.	Sprint 8	69
7.1.	Planificación	69
7.2.	Descripción	69
7.3.	User Stories relacionados	69
7.4.	Modelo de datos	70
7.5.	Modelo funcional.	70
7.5.1.	Creación de página de mediciones	70
7.6.	Salidas del Sistema - Incrementos	70
7.7.	Criterios de aceptación	71
7.8.	Casos de prueba	71
7.8.1.	Pruebas de integración entre módulos del Sistema	71
7.8.2.	Pruebas de carga	71

7.8.3. Pruebas de seguridad por niveles de usuarios	71
7.9. Pruebas ejecutadas	71
8. Programación y documentación	72
8.1. Descripción de la funcionalidad	73
8.1.1. Cargar análisis	73
8.1.2. Generación de tablas y gráficas de mediciones	73
8.2. Diseño Técnico	73
8.2.1. Objetivo	81
8.2.2. Descripción	82
8.3. Técnicas de programación utilizadas	82
8.4. Entorno, herramientas y tecnologías utilizadas	82
8.5. código fuente	82
8.6. pruebas	82
9. Planificación de la capacitación	82
9.1. Capacitación e instrucciones de uso para los usuarios	82
9.2. Capacitación e instrucciones de uso para las instituciones	83
9.2.1. Plan de capacitación	83
9.2.2. Alcance	83
9.2.3. Requisitos para realizar el curso	84
9.2.4. Fines de la capacitación	84
9.2.5. Objetivos generales	84
9.2.6. Objetivos específicos	84
9.2.7. Metas	85
9.2.8. Método de capacitación y evaluación	85
10. Ejecución, documentación y retroalimentación de pruebas.	86
11. Manual de usuario del Sistema completo	86
12. Planificación de Implantación del Sistema	86
12.1. Publicidad y propaganda	86
12.2. Configuración y diseño del sistema	87
12.3. Implantación en instituciones	88
12.3.1. Preparación de datos y archivos	88
12.4. Despliegue del servidor de front-end	89

A. Anexo	91
A.1. Código	91

1. Sprint 2

1.1. Planificación

Inicio: Martes 19 de mayo del 2015

Fin: Martes 09 de junio del 2015

1.2. Descripción

En este sprint se llevaran a cabo las interfaces necesarias para que el usuario pueda cargar nuevas mediciones y ver todas sus últimas mediciones ,así teniendo un seguimiento de las mismas con posibilidad de que posteriormente pueda ver su evolución a través de gráficas y tablas. Para la comunicación de estas interfaces con la API, se desarrollaran los correspondientes adaptadores para los recursos.

Para esto en el backend se deben preparar las clases Measurement, MeasurementType, MeasurementSource, MeasurementUnit con las debidas relaciones con la clase Profile desarrollada en el sprint anterior. Para cada una de esas clases, se deben preparar interfaces de acceso a los recursos provistos por la API. Y su correspondiente documentación.

1.3. User Stories relacionados

La **Tabla ??** indicará las características de cada user story para guiarnos en el desarrollo del sprint.

ID	Enunciado de la historia	Prioridad
US-??	Como paciente quiero obtener un resumen de mi información de salud básica para hacer uso de la misma en caso de una emergencia	Alta
US-??	Como paciente quiero cargar mi información personal de salud referido a mediciones (altura, grasa corporal, peso, presión arterial), para que el médico cuente con más y mejor información al momento de realizar el diagnóstico.	Alta

1.4. Modelo de datos.

El Diagrama propio de este sprint se puede ver en la **Figura1**, allí se indican exactamente las clases que se usarán en este sprint y que serán detalladas con detenimiento en el presente documento. Se recuerda que se ha realizado un Diagrama de clases tentativo que se puede ver en la **Figura 1**, dicho diagrama será utilizado como base para este sprint y posee un alcance limitado el cual se irá modificando a medida que se profundice en los temas.

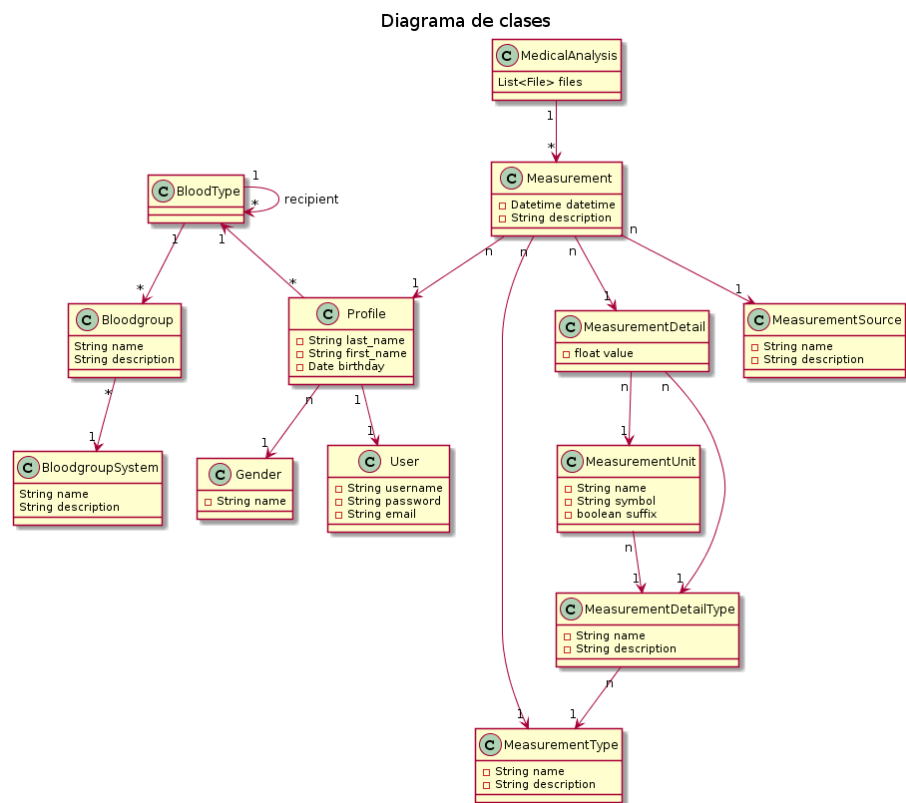


Figura 1: Modelo de datos

1.5. Descripción de las Clases

1.5.1. Clase Measurement

Dicha clase se refiere a las medición realizada por el usuario en un momento específico.

Descripción de los atributos

- **id:** Identificador único de la medición (tipo int).
- **datetime:** Fecha y hora de la medición (tipo datetime).
- **value:** Valor de la medición (tipo float).
- **profile_id:** Identificador único del perfil asociado (tipo int).
- **measurement_source_id:** Identificador único de la fuente de medición asociada (tipo int).
- **measurement_type_id:** Identificador único del tipo de medición asociado (tipo int).
- **measurement_unit_id:** Identificador único de la unidad de medición asociada (tipo int).

Dirección del recurso:

```
1 <BASE URL>/measurements/{:id}
```

Json generado por la API

```
1 {
2   "resource":
3   {
4     "measurement_unit":
5     {
6       "symbol": "Kg",
7       "suffix": true,
8       "name": "Kilogramo",
9       "id": 1
10    },
11    "measurement_source":
12    {
```

```
13     "name": "Manual",
14     "description": null,
15     "id": 1
16 },
17 "value": 50,
18 "measurement_type":
19 {
20     "name": "Peso",
21     "description": "Peso corporal de la persona.",
22     "id": 1
23 },
24 "id": 1,
25 "profile":
26 {
27     "birthday": "1990-10-26",
28     "last_name": "Terreno",
29     "first_name": "Milton",
30     "gender":
31     {
32         "name": "Masculino",
33         "description": null,
34         "id": 1
35     },
36     "id": 1
37 },
38     "datetime": "2015-06-15T02:29:54"
39 }
40 }
```

1.5.2. Clase MeasurementType

Esta clase nos permitirá nomenciar los tipos de medidas, hasta el momento hemos contemplado: peso, dimensión corporal (Ej:altura) y glucosa. Existen ciertas medidas que contemplan dos valores, estas serán agregadas en un sprint futuro.

Descripción de los atributos

- **name:** Nombre del tipo de medición(tipo string).

-
- **description:** Descripción del tipo de medición (tipo string).

Dirección del recurso:

```
1 <BASE URL>/measurement_types/{id}
```

Json generado por la API

```
1 {
2   "resource":
3   {
4     "name": "Peso",
5     "description": "Peso corporal de la persona.",
6     "id": 1
7   }
8 }
```

1.5.3. Clase MeasurementUnit

Esta clase nos permitirá nomenciar las unidades de medicion disponible para que el usuario pueda seleccionarlhas cdo realice la medición, hasta el momento hemos contemplado: Kilogramo, gramo, miligramos, metro, centimetro y milímetro.

Descripción de los atributos

- **id:** Identificador único de la unidad de medición(tipo int).
- **name :** Nombre de la unidad de medición (tipo string).
- **symbol :** Símbolo de la unidad de medición (tipo string).
- **suffix :** Variable booleana que indica si el símbolo de la unidad de medición es un sufijo (verdadero) o un prefijo (falso) del valor de la medición (tipo boolean).

Dirección del recurso

```
1 <BASE URL>/measurement_units/{id}
```

Json generado por la API

```
1 {
2   "resource":
```

```
3      {
4          "symbol": "Kg",
5          "suffix": true,
6          "name": "Kilogramo",
7          "id": 1
8      }
9  }
```

1.5.4. Clase MeasurementSource

Esta clase nos permitirá nomenciar los tipos de fuentes posibles como pueden ser manual, dispositivo móvil, sistema de salud y dispositivo de salud.

Descripción de los atributos

- **name** : Nombre de la fuente de medición (tipo string).
- **description** : Descripción de la fuente de medición (tipo String).

Dirección del recurso

```
1  <BASE URL>/measurement_sources/{:id}
```

Json generado por la API

```
1  {
2      "resource":
3      {
4          "name": "Manual",
5          "description": null,
6          "id": 1
7      }
8  }
```

1.6. Modelo funcional.

Se describirán las funciones usando como marco de apoyo el sprint Backlog, además se armará el diagrama de casos de uso del presente Sprint [Figura 44] que irá creciendo medida se vaya avanzando en el proyecto.

Diagrama de casos de uso yesdoc

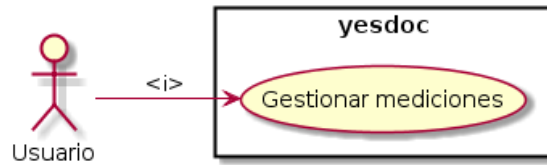


Figura 2: formulario de edición de perfil

Area a cargo	Responsable	Tarea	US
Front-end	Ivan Terreno	Generación de controladores para consumir Json de la Api relacionados a la API	US-?? & US-??
Front-end	Yanina Morales	Creación de página de formulario de carga de mediciones	US-??
Front-end	Ivan Terreno	Creación de página de formulario de carga de perfil	US-??
Back-end	Michael Manganiello	Creación de modulo de mediciones	US-?? & US-??
Back-end	Michael Manganiello	Exposición de métodos como servicios de API	US-?? & US-??
Back-end	Franco Canizo	Adaptación de salida de métodos a formato Json	US-?? & US-??
Back-end	Franco Canizo	Carga de valores a la base de datos, relacionados a la API	US-?? & US-??

1.6.1. Creación de página de mediciones

En esta tarea se generará la pantalla, **Figura 3** donde se muestran las mediciones del usuario, como lo son:

- Altura
- Peso
- Grasa Corporal
- Presión Arterial

Al igual que en la creación del perfil se dará la posibilidad de acceder a la edición de su perfil desde esta misma página.

Para mostrarlas mediciones del usuario es necesario acceder al recurso `/profiles/{profile}` de la API a través de un método **GET**

Especificaciones del recursos `/measurements`

```
1      MeasurementFields {
2          measurement_source (MeasurementSourceFields,
3              optional),
4          profile (ProfileFields),
5          datetime (date-time),
6          value (number),
          measurement_unit (MeasurementUnitFields),
```

```
7      id (integer),
8      measurement_type (MeasurementTypeFields)
9  }
10     MeasurementSourceFields {
11     description (string, optional),
12     id (integer),
13     name (string)
14 }
15     ProfileFields {
16     first_name (string),
17     last_name (string),
18     id (integer),
19     gender (GenderFields, optional),
20     birthday (date-time, optional)
21 }
22     GenderFields {
23     description (string, optional),
24     id (integer),
25     name (string)
26 }
27     MeasurementUnitFields {
28     id (integer),
29     symbol (string),
30     suffix (boolean, optional),
31     name (string)
32 }
33     MeasurementTypeFields {
34     description (string, optional),
35     id (integer),
36     name (string)
37 }
```

En el perfil de usuario se mostrarán de cada tipo de medición que ha realizado el usuario la última de cada una, indicando el nombre, el valor, el simbolo, la fecha y hora y el método con el que ha sido realizada la medición. Además para cada una de las mediciones se mostrarán dos iconos que corresponden a la edición y a la compartición de las mediciones, este último será implementado en un sprint futuro.

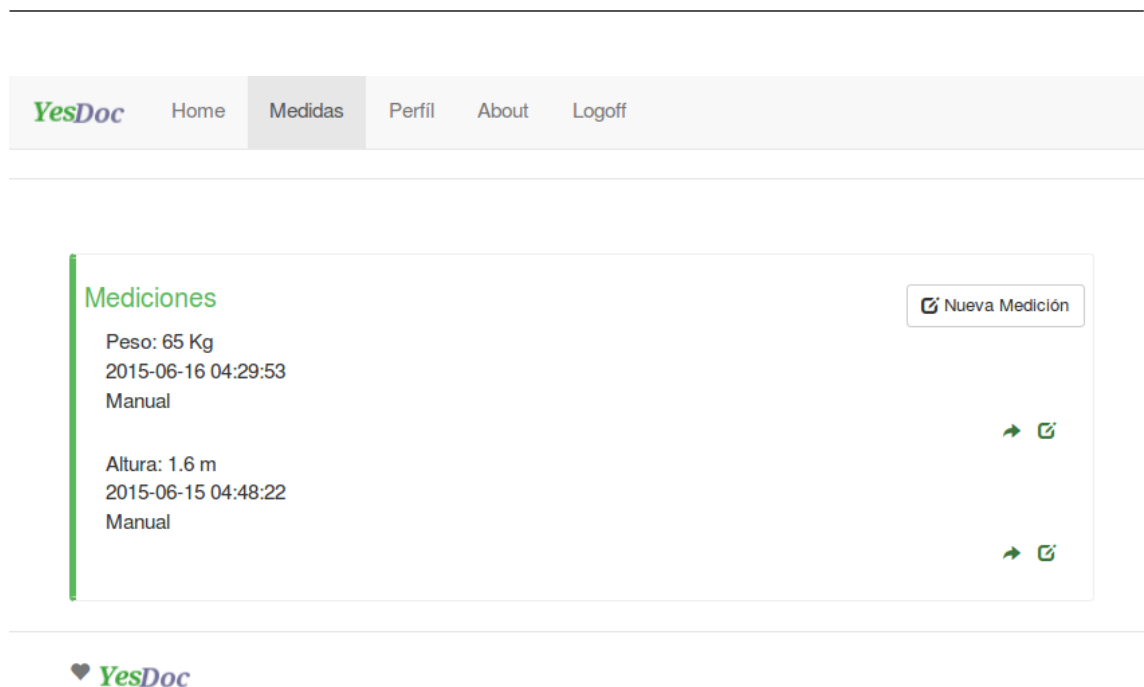


Figura 3: Perfil de mediciones

1.6.2. Creación de página de formulario de carga de mediciones

Se generará el formulario necesario, que se muestra en la **Figura 4** para que el usuario pueda cargar las mediciones antes nombrados, para ello es necesario acceder al recurso `/measurements` de la API a través de un método **POST**

```
1      MeasurementFields {
2      measurement_source (MeasurementSourceFields,
3      optional),
4      profile (ProfileFields),
5      datetime (date-time),
6      value (number),
7      measurement_unit (MeasurementUnitFields),
8      id (integer),
9      measurement_type (MeasurementTypeFields)
10     }
11     MeasurementSourceFields {
12     description (string, optional),
13     id (integer),
14     name (string)
```

```
14     }
15     ProfileFields {
16         first_name (string),
17         last_name (string),
18         id (integer),
19         gender (GenderFields, optional),
20         birthday (date-time, optional)
21     }
22     GenderFields {
23         description (string, optional),
24         id (integer),
25         name (string)
26     }
27     MeasurementUnitFields {
28         id (integer),
29         symbol (string),
30         suffix (boolean, optional),
31         name (string)
32     }
33     MeasurementTypeFields {
34         description (string, optional),
35         id (integer),
36         name (string)
37     }
```

Desde el perfil de mediciones se presentará un icono que representa a la creación de un elemento para que el usuario pueda seleccionarlo. Esta acción llevará al usuario al formulario de creación de mediciones, donde los campos estarán vacíos, para que el usuario los cargue con los valores correspondientes. Una vez terminada la carga, se mostrará un mensaje avisando al usuario que se ha realizado con éxito y luego lo redireccionará al perfil de mediciones.

1.6.3. Creación de página de formulario de edición de mediciones

Se generará el formulario necesario, que se muestra en la **Figura 5** para que el usuario pueda editar una medición previamente seleccionada, para ello es necesario acceder al recurso `/measurements/{:id}` a través del método **PUT** enviando por URL el id correspondiente a dicha medición.

YesDoc Home Medidas Perfil About Logoff

Tipo:

Valor:

Resultado de medición

Unidad:

Fuente:

Fecha:

2015-06-16

05:19:35

Save

♥ YesDoc

Figura 4: Formulario de nueva medición

Desde el perfil de mediciones se presentará un icono que representa a la edición de un elemento para que el usuario pueda seleccionarlo. Esta acción llevara al usuario al formulario de edición de mediciones, donde los campos estarán cargados con los valores antiguos, de este modo el usuario modifica lo que desea y no tiene que cargar todo nuevamente. Una vez terminada la edición se redireccionará al perfil de mediciones.

1.7. Salidas del Sistema - Incrementos

Luego de finalizado este user story se obtendrán 4 pantallas que se detallarán a continuación:

1. **Presentación de las últimas mediciones [Figura 3]** con posibilidad de edición de cada una de las mediciones. Los datos posible a presentar son altura, peso, grasa corporal y glucosa.

La interfaz mostrará el valor de la medición, la fecha y hora en que fue realizada y la fuente que se utilizó para dicha medición.

2. **Carga de mediciones: [Figura 4]** Se le permitirá cargar mediciones que realice en algún momento del día como son peso, altura, grasa corporal y glucosa.

YesDoc Home Medidas Perfil About Logoff

Tipo:

Valor:

Unidad:

Fuente:

Fecha:

♥ **YesDoc**

Figura 5: Formulario de edición de medición

Deberá indicar la fuente, tipo, unidad y fecha de la medición

3. **Edición de mediciones:** [Figura 5] Se le permitirá seleccionar una medición del perfil de mediciones para ser modificada.

1.8. Criterios de aceptación

Criterio de aceptación			
Id	Contexto	Evento	Resultado
1	En caso de que exista una persona sin mediciones	cuando este desee observar sus mediciones	El sistema no mostrará nada
2	Cuando el usuario registrado ingresa dos mediciones del mismo tipo	y luego quiera consultarlas	El sistema solo le mostrará la ultima medición, del mismo tipo, realizada
3	Cuando el usuario seleccione una medición	y luego quiera editarla	El sistema le permitira la correspondiente edición

4	Si el usuario existe y no está logeado	y quiera ingresar a ver sus mediciones.	El sistema no le permitirá ingresar
---	--	---	-------------------------------------

1.9. Casos de Prueba

Caso de prueba	Consultar mediciones (sin medidas precargadas)
Descripción del escenario	Nombre: Marita; Apellido Martinez; fecha de Nacimiento:20-08-1989; id:3
Criterio de aceptación	En caso de que exista una persona sin mediciones cargadas, si el usuario desea verlas el sistema no debería mostrar ninguna medición
Datos de entrada	consultar mediciones
Condiciones de prueba	Se necesita que esté previamente cargado el usuario "Marita Martinezz no tenga datos más allá de su nombre y apellido en el perfil

Cuadro 2: Caso de prueba para criterio de aceptación 1

Procedimiento de Prueba - “Consultar mediciones”		
Actor	Sistema	Resultado Esperado
El usuario ingresa al sistema con su id nº3		
	El sistema valida con los perfiles de la API si el Id:3 del usuario existe	Se presenta por pantalla el perfil del usuario con sus datos
El usuario selecciona la pestaña mediciones y realiza la consulta		
	El sistema valida que las cookies estén activas	
	El sistema solicita a la API las mediciones del perfil con id:3	El sistema no muestra ninguna medida cargada

Cuadro 4: Procedimiento de prueba para criterio de aceptación 1

Salida obtenida	No se presentan datos de mediciones
Resultado	Correcto
¿Que fue mal?	Nada
Evidencia	
Seguimiento	No es necesario ya que el caso de prueba no causó fallos
Estado	Terminado
¿Que se puede mejorar?	En otra itreación se debería añadir carteteles de avisos, informando que faltan cargar datos

Cuadro 5: Resultado esperado para el criterio de aceptación 1

Caso de prueba	Consultar mediciones (medida precargada)
Descripción del escenario	Nombre: Marita; Apellido Martinez; fecha de Nacimiento:20-08-1989; id:3; altura: 2m
Criterio de aceptación	Cuando el usuario registrado ingresa dos mediciones del mismo tipo y luego quiera consultarlas. El sistema solo le mostrará la ultima medición, del mismo tipo, realizada
Datos de entrada	id:3; Peso 1: 67kg; peso 2: 55kg
Condiciones de prueba	el usuario Marita Martinez existe

Cuadro 6: Caso de prueba para criterio de aceptación 2

Procedimiento de Prueba - Consultar mediciones		
Actor	Sistema	Resultado Esperado
El usuario se logea en el sistema		
	El sistema consulta la API para corroborar que el perfil con id:3 existe	
	El sistema redirecciona al usuario a la vista de perfil de usuario	Se muestra el perfil de usuario con los datos respectivos
El usuario selecciona la pestaña de mediciones de la barra de navegación		
	El sistema verifica las cookies del usuario para determinar si ya se encuentra logeado	
	El sistema redirecciona al usuario a la vista de mediciones	Se muestra la vista de mediciones con las últimas mediciones del usuario correspondientes
El usuario selecciona el botón de añadir “nueva medicion”		

	El sistema verifica que las cookies posean los datos del usuario	
	Se redirecciona a la vista de carga de mediciones	Se presenta el formulario de medición para la carga respectiva
El usuario selecciona: Tipo de medicion: Peso; medida 55; unidad: Kg; Fuente: manual; fecha: deja la precargada y luego de esto presiona el botón save		
	El sistema valida que estén todos los datos cargado, excepto fuente el cual no es necesario, carga los nuevos datos en la API a través del método POST y redirecciona a la vista de mediciones	
	El sistema a través del método GET trae las últimas mediciones.	Se muestran las últimas mediciones en la vista de mediciones
El usuario presiona el botón “cargar mediciones” nuevamente		
	El sistema verifica que las cookies posean los datos del usuario	
	Se redirecciona a la vista de carga de mediciones	Se presenta el formulario de mediciones para la carga respectiva

El usuario selecciona: Tipo de medicion: Peso; medida 67; unidad: Kg; Fuente: manual; fecha: deja la precargada y luego de esto presiona el botón save		
	El sistema valida que estén todos los datos cargado, excepto fuente el cual no es necesario, carga los nuevos datos en la API a través del método POST y redirecciona a la vista de mediciones	
	El sistema a través del método GET trae las últimas mediciones.	El sistema muestra la última medición cargada “Peso 55 Kg, fecha de carga y método: manual

Cuadro 7: Procedimiento de prueba para criterio de aceptación 2

Salida obtenida	Se mostraron correctamente la ultima medición del mismo tipo
Resultado	Correcto
¿Que fue mal?	Nada
Evidencia	En Listing 1 se puede observar que el usuario posee 3 mediciones, dos de las cuales son del mismo tipo (tipo Peso) y en la Figura 6 se puede ver que solo se muestra la última medición del Peso.
Seguimiento	no es necesario
Estado	Terminado
¿Que se puede mejorar?	

Cuadro 8: Resultado esperado para el criterio de aceptación 2

Listing 1: Json de las mediciones del perfil id:3

```

1  {"resource":
2      [{
3          "measurement_source":
4              {
5                  "id": 1,
6                  "description": null,
7                  "name": "Manual"
8              },
9          "measurement_type":
10             {
11                 "id": 1,
12                 "description": "Peso corporal de la persona.",
13                 "name": "Peso"
14             },
15             "datetime": "2015-07-03T11:51:39.436000",
16             "value": 55,
17             "id": 16,
18         "measurement_unit":
19             {
20                 "id": 1,
21                 "suffix": true,
22                 "name": "Kilogramo",
23                 "symbol": "Kg"
24             }
25         },
26     ],
27     {
28         "measurement_source":
29             {
30                 "id": 1,
31                 "description": null,
32                 "name": "Manual"
33             },
34         "measurement_type":

```



```

35      {
36          "id": 1,
37          "description": "Peso corporal de la persona.",
38          "name": "Peso"
39      },
40      "datetime": "2015-07-03T11:54:22.806000",
41      "value": 67,
42      "id": 17,
43      "measurement_unit":
44      {
45          "id": 1,
46          "suffix": true,
47          "name": "Kilogramo",
48          "symbol": "Kg"
49      }
50  },
51  {"measurement_source":
52  {
53      "id": 0,
54      "description": null,
55      "name": null
56  },
57  "measurement_type":
58  {
59      "id": 2,
60      "description": "Longitud de la persona",
61      "name": "Altura"
62  },
63      "datetime": "2015-07-03T13:05:57.375000",
64      "value": 2,
65      "id": 18,
66      "measurement_unit":
67      {
68          "id": 2,
69          "suffix": true,
70          "name": "Metros",
71          "symbol": "m"
72      }
73  }
74  }

```

Mediciones				Nueva Medición	
Peso: 67 Kg	2015-07-03 11:54:22	Manual			
Altura: 2 m	2015-07-03 13:05:57				

Figura 6: perfil de medicion de usuario con id:3

Caso de prueba	Editar Medición
Descripción del escenario	Nombre: Marita; Apellido Martinez; fecha de Nacimiento:20-08-1989; id:3; id:3; Peso 1: 67kg, id:17; peso 2: 55kg, id:16
Criterio de aceptación	Cuando el usuario seleccione una medición y luego quiera editarla. El sistema le permitira la correspondiente edición
Datos de entrada	peso:65Kg
Condiciones de prueba	Usuario logeado con al menos una medida cargada

Procedimiento de Prueba - Editar mediciones		
Actor	Sistema	Resultado Esperado
El usuario, ya logeado selecciona el botón de editar de una de las mediciones (medición peso 67Kg) que se muestra en su perfil de mediciones		
	El sistema valida las que las cookies estén activas	
	El sistema consulta a la API la medición 17 correspondiente al perfil con id:3	Se muestra el perfil del formulario de carga de mediciones. con los datos precargados de la medición seleccionada
El usuario modifica los datos de la medición seleccionada cambiando 67 por 65		
	el sistema confirma la carga guardando los datos en la API a través del método PUT	

	El sistema redirecciona al usuario a la vista de perfil de mediciones	Se le presenta al usuario la vista de las ultimas mediciones realizadas. Mostrando 65Kg
--	---	---

Cuadro 9: Procedimiento de prueba para criterio de aceptación 3

Salida obtenida	La vista presento la medición modificada de forma correcta
Resultado	Correcto
¿Que fue mal?	Nada
Evidencia	En la figura 7 se puede ver como el formulario se encuentra precargado con los valores de la medición que se desea editar, en el Json 2 se puede observar que se modifiko el valor del peso y que ha cambiado la fecha de carga
Seguimiento	No es necesario ya que el caso de prueba no causó fallos
Estado	Terminado
¿Que se puede mejorar?	En otro sprint se debería añadir carteles de avisos, informando que la edición fue realizada con éxito

Cuadro 10: Resultado esperado para el criterio de aceptación 3

Listing 2: Json de las medicion modificada del perfil id:3

```

1 {
2     "id": 17,
3     "measurement_source":
4     {
5         "id": 1,
6         "description": null,
7         "name": "Manual"
8     },
9     "measurement_unit":
10    {
11        "id": 1,

```

Tipo:	<input type="text" value="Peso"/>
Valor:	<input type="text" value="67"/>
Unidad:	<input type="text" value="Kilogramo"/>
Fuente:	<input type="text" value="Manual"/>
Fecha:	<input type="text" value="2015-07-03"/>
	<input type="text" value="11:54:22"/>
	<input type="button" value="Save"/>

Figura 7: Formulario de edición de medición

```
12     "suffix": true,  
13     "symbol": "Kg",  
14     "name": "Kilogramo"  
15     },  
16     "measurement_type":  
17     {  
18         "id": 1,  
19         "description": "Peso corporal de la persona.",  
20         "name": "Peso"  
21     },  
22     "value": 65,  
23     "datetime": "2015-07-03T11:54:22.806000"  
24 }
```

Caso de prueba	Ingresar a mediciones
Descripción del escenario	Nombre: Marita; Apellido Martinez; fecha de Nacimiento:2015-06-01; género: femenino; id:3; Peso 1: 54kg; peso 2: 55kg
Criterio de aceptación	Si el usuario existe y no está logeado y quiere ingresar a ver sus mediciones. El sistema no le permitirá ingresar
Datos de entrada	
Condiciones de prueba	El usuario no debe encontrarse logeado.

Cuadro 11: Caso de prueba para criterio de aceptación 4

Procedimiento de Prueba -Ingresar mediciones		
Actor	Sistema	Resultado Esperado
El usuario selecciona la pestaña de mediciones, para ver sus mediciones		
	El sistema verifica que exista una cookie activa como na existe no lo redirección a ninguna parte	Se muestra la ventana de logeo

Cuadro 12: Procedimiento de prueba para criterio de aceptación 4

Salida obtenida	Se obtuvo lo q se esperaba, ya que no fue enviado a la interface de mediciones.
Resultado	Correcto
¿Que fue mal?	Nada
Evidencia	No es necesaria
¿Que fue mal?	Nada
Seguimiento	No es necesario ya que el caso de prueba no causó fallos
Estado	Terminado
¿Que se puede mejorar?	En una futura iteración se podría añadir carteles de avisos informando de la situación

Cuadro 13: Resultado esperado para el criterio de aceptación 4

1.9.1. Pruebas de integración entre módulos del Sistema

Estas pruebas se realizarán mas adelante

1.9.2. Pruebas de carga

En este sprint no se realizarán este tipo de pruebas.

1.9.3. Pruebas de seguridad por niveles de usuarios

En este sprint no se realizarán este tipo de pruebas, ya que la seguridad será un tema a tratar más adelante.

1.10. Pruebas ejecutadas

Aqui se realizará una conclusión general de lo que se descubrió en las pruebas.

▪ ¿Que fue bien?

- Las cargas y ediciones se llevan a cabo correctamente.

▪ ¿Que se mejoró?

- **Cerrado** Al crear una nueva medición, se mostraba un cartel (alert de javascript) con una fecha, dicho alert fue eliminado.
- **Cerrado** Se encontró un problema con la zona horaria que usa el servidor y la zona horaria del usuario, para solucionarlo hubo q hacer un casteo previo cuando se solicitaba la fecha y hora del usuario para mostrar.

▪ ¿Que se puede mejorar?

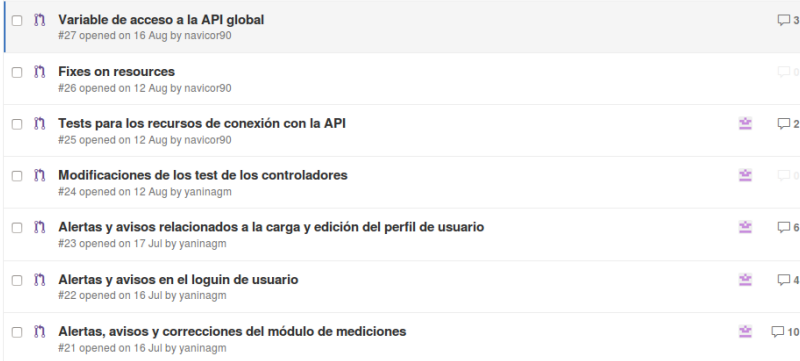
- **Abierto** En el futuro se deberá mejorar las validaciones de los datos a la hora de cargar información en los formularios.
- **Abierto** Se deberá mejorar la manera de seleccionar la fecha y la hora.
- **Abierto** Solo debería mostrarse las unidades relacionadas al tipo de medición que se ha seleccionado
- **Abierto** Deberá realizarse los carteles de advertencia necesarios.

2. Sprint 3:

2.1. Planificación

Inicio: Martes 09 de junio del 2015

Fin: Martes 30 de junio del 2015






















<input type="checkbox"/>	 Variable de acceso a la API global #27 opened on 16 Aug by navicor90	 3
<input type="checkbox"/>	 Fixes on resources #26 opened on 12 Aug by navicor90	 0
<input type="checkbox"/>	 Tests para los recursos de conexión con la API #25 opened on 12 Aug by navicor90	  2
<input type="checkbox"/>	 Modificaciones de los test de los controladores #24 opened on 12 Aug by yaninagm	  0
<input type="checkbox"/>	 Alertas y avisos relacionados a la carga y edición del perfil de usuario #23 opened on 17 Jul by yaninagm	  6
<input type="checkbox"/>	 Alertas y avisos en el login de usuario #22 opened on 16 Jul by yaninagm	  4
<input type="checkbox"/>	 Alertas, avisos y correcciones del módulo de mediciones #21 opened on 16 Jul by yaninagm	  10

Figura 8: Pull request realizados en el sprint 3

2.2. Descripción

En este sprint se corregirán los errores detectados en las pruebas realizadas con anterioridad en los sprint referidos a:

- Generar perfil de datos personales.
- Generar perfil de mediciones.

Cabe destacar que sólo se documentarán aquellas correcciones que se refieran a las funcionalidades antes mencionadas

Se desarrollarán las interfaces que permiten mostrar las gráficas de ñas mediciones de un usuario.

Y se realizarán las validaciones necesarias para que el sistema funcione correctamente.

2.3. User Stories relacionados

La **Tabla ??** indicará las características de cada user story para guiarnos en el desarrollo del sprint.

ID	Enunciado de la historia
US-2	Como paciente, quiero añadir al sistema los estudios realizados para evitar posibles perdidas.
US-5	Como paciente quiero que los sistemas de salud existentes puedan cargar sus resultados directamente en mi carpeta de salud para centralizar mi información.
US-7	Como paciente quiero categorizar mis estudios por rama de medicina, para lograr una mejor organización y navegabilidad en el sistema.
US-8	Como laboratorio, quiero cargar información de un paciente en su cuenta para ahorrarle las molestias de volver.
US-17	Como paciente quiero ver gráficas que resuman mi información en particular para poder ver mis cambios a lo largo de la historia.
US-15	Como médico quiero ver gráficas que resuman la información de un paciente para poder ver sus cambios a lo largo de la historia y así apoyar la toma de decisiones y el diagnóstico.

2.4. Modelo funcional.

Se describirán las funciones usando como marco de apoyo el sprint Backlog.

Area a cargo	Responsable	Tarea	US
Back-end	Franco Canizo	Agrega a la estructura de la API paquete para validadores generales.	US-?? & US-??
Back-end	Franco Canizo	Creación de validadores para números enteros positivos, fecha, fecha-hora, fecha-hora previa, string consin números	US-?? & US-??
Back-end	Michael Manganiello	Filtrado de mediciones en base al tipo, fuente y unidad de medición	US-?? & US-??
Back-end	Michael Manganiello	Corrección error de comparación de fechas con información de zona horaria y fechas sin esa información	US-?? & US-??
Back-end	Michael Manganiello	Simplificación del parseo de argumentos date y datetime	US-?? & US-??
Back-end	Michael Manganiello	Creación validador fecha-hora previa	US-?? & US-??
Front-end	Yanina Morales	Creación de validadores y mensajes de alerta	
Front-end	Ivan Terreno	Generación de gráficas	US-17 & US-15
Front-end	Ivan Terreno	Capacitación en D3	US-17 & US-15

2.4.1. Validadores

Del lado del backend el trabajo consistió en la definición de validadores usados para controlar errores y, de ser posible, sanear los datos que envían las representaciones a los recursos. Lo que se hizo es definir en un paquete una serie de validadores generales usados en el paquete parsers en common. Cómo podemos apreciar en la definición del parser usado por el recurso de mediciones:

```

1 parser = reqparse.RequestParser()
2 parser.add_argument('datetime', type=is_valid_previous_datetime,
3                     required=True)
4 parser.add_argument('value', type=float, required=True)
5 parser.add_argument('analysis_id', type=is_valid_id, required=True)
6 parser.add_argument('profile_id', type=is_valid_id, required=True)
7 parser.add_argument('measurement_source_id', type=is_valid_id, required=True)
8 parser.add_argument('measurement_type_id', type=is_valid_id, required=True)
9 parser.add_argument('measurement_unit_id', type=is_valid_id, required=True)

```

Se define en el argumento `type` el llamado a una función `is_valid_previous_datetime`, esta función está definida en el paquete `validators` en el archivo `generic validators`.

```

1 datetime_var = is_valid_datetime(var)
2 if datetime_var.year < 1900:
3     raise ValueError("La fecha y hora ingresada no puede ser
4     anterior al año 1900.")
5 elif datetime_var > datetime.utcnow():
6     raise ValueError("La fecha y hora ingresada no debe ser
7     posterior a la fecha y hora actual.")
8 else:
9     return datetime_var

```

Este a su vez llama al validador `is_valid_datetime` para controlar que el dato “datetime” tenga un valor de fecha-hora correcto. Si el validador no encuentra error devuelve la fecha-hora, en caso contrario lanza un error. En este parser también se define un método para controlar que el id recibido es valido, por el momento lo que hace este es controlar que el id recibido sea positivo.

2.4.2. Filtrado de mediciones

El backend identifica como útil la definición de un recurso que permita devolver medidas de un perfil específico filtradas según tipo, fuente y unidad de medida. Para esto se agregó un recurso `profileidmeasurements` en el archivo principal de la aplicación y un recurso que define un método `get` para responder a una solicitud HTTP con operador GET. El mismo utiliza los datos recibidos en el “query string” de la URL para determinar según que tipo, fuente o unidad de medida debe filtrar. El recurso en sí no se encarga del filtrado sino que toma los datos verificados por el parse, obtiene los datos del query string y pasa estos datos a un método auxiliar definido en el archivo `measurement` del paquete `persistence` de la API “`get_by_profile`” el cual devuelve todas las instancias existentes de medición, asociadas a un perfil específico,

ordenadas por fecha y hora de la medición, y filtradas por fuente, tipo y unidad de medición. Una vez recibida la respuesta arma el response con el código de estado correspondiente y en el cuerpo los datos resultantes serializados de acuerdo a la representación utilizada.

2.5. Criterios de aceptación

Criterio de aceptación			
Id	Contexto	Evento	Resultado
1	En caso de que se envíe como id un valor menor o igual a 0 en cualquier representación de una solicitud	Al ejecutar el método post, get, delete o put correspondiente del recurso	El sistema devolverá un json con el mensaje de error por entero no positivo y el código de error 400
2	En caso de que se indique una fecha cuyo año sea menor a 1900	al ejecutar el validador is_valid_previous_date del argumento	El sistema devolverá un json con el mensaje de error correspondiente a una fecha previa no permitida y el código de estado 400
3	En caso de que se indique en la solicitud una fecha-hora con formato invalida	Al ejecutar el validador is_valid_datetime	El sistema devolverá un json con el mensaje de error correspondiente a una fecha hora con formato inválido y el código de estado 400
4	En caso de que no exista un usuario registrado con el id indicado	al ejecutar el método getput del recurso usersid	El sistema devolverá un json con un mensaje de error y un código de error 404
5	En caso de que no exista un perfil registrado con el id indicado	al ejecutar el método get del recurso profileidmeasurements	El sistema devolverá un json con un mensaje de error por no encontrar la instancia correspondiente

2.6. Pruebas ejecutadas

A continuación se detalla la situación en la que quedaron las pruebas ejecutadas en los sprint anteriores, luego se desarrollarán las soluciones que se usaron y por última se cambiará el estado de aquellas errores encontrados por *cerrado* ".

- **¿Qué fue bien?**
 - Las cargas y ediciones se llevan a cabo correctamente.
- **¿Qué se mejoró?**
 - **Cerrado** Al crear una nueva medición, se mostraba un cartel (alert de javascript) con una fecha, dicho alert fue eliminado.
 - **Cerrado** Se encontró un problema con la zona horaria que usa el servidor y la zona horaria del usuario, para solucionarlo hubo q hacer un casteo previo cuando se solicitaba la fecha y hora del usuario para mostrar.
- **¿Qué se puede mejorar?**
 - **Abierto** Solo debería mostrarse las unidades relacionadas al tipo de medición que se ha seleccionado
 - **Abierto** En el futuro se deberá mejorar las validaciones de los datos a la hora de cargar información en los formularios.
 - **Abierto** Se deberá mejorar la manera de seleccionar la fecha y la hora.
 - **Abierto** Deberá realizarse los carteles de advertencia necesarios.

Mostrar unidades relacionadas al tipo de medición seleccionado

Para evitar errores humanos fue necesario mostrar sólo las unidades de medida que se encuentran relacionadas a un tipo de medición seleccionada por el usuario, por ejemplo: si selecciona Tipo de medición, "*Peso*", como se muestra en la **Figura 10** el sistema solo debería mostrar las unidades que correspondan a ese tipo de medición y no mostrar metros como una posible unidad igual para el caso de que seleccione la altura **Figura 11**.

A nivel de frontEnd fue necesario deshabilitar la selección de unidades cuando el usuario no ha seleccionado el tipo de medición como se indica en la **Figura 9** y una vez seleccionado se tuvo que solicitar a un recurso de la API las unidades relacionadas al tipo de medición seleccionado como se muestran en las figuras antes citadas.

Formulario de registro de medicación. Campos: Tipo (Peso), Valor (56), Unidad (mensaje de error: "Seleccione una unidad de medición."), Fuente (Manual), Fecha (2015-10-13, 18:37:58). Botón Save.

Figura 9: Mensaje sutil que solicita que se seleccione un tipo de medición

Formulario de registro de medicación. Campos: Tipo (Peso), Valor (56), Unidad (lista desplegable abierta mostrando Kilogramo y gramo), Fuente, Fecha (2015-10-13, 19:01:44). Botón Save.

Figura 10: Lista de unidades al seleccionar el tipo de unidad “Peso”

El diagrama de clases, las relaciones y los recursos de la API necesarios para poder establecer esta relación se detallan en el apartado 2.7

Carteles de alertas

Luego de ejecutar las pruebas se detectó que para mejorar la experiencia del usuario es necesario añadir mensajes de avisos, como se muestra en la **Figura 12 y Figura 13** indicando al usuario situaciones importantes como las que indican ausencia de información importante en el formulario, además el botón para enviar el formulario se mantiene deshabilitado hasta que se haya completado los datos obligatorios (que en este caso serían todos) del formulario.

The image shows a web form with the following fields: 'Tipo' (set to 'Altura'), 'Valor' (set to '56'), 'Unidad' (a dropdown menu with 'Metros' and 'centimetros' visible, where 'Metros' is highlighted), 'Fuente' (empty), and 'Fecha' (set to '2015-10-13' and '19:01:44'). A green 'Save' button is at the bottom.

Figura 11: Lista de unidades al seleccionar el tipo de unidad “Altura”

The image shows the same form as Figure 11, but with validation messages. The 'Tipo' field is empty. The 'Valor' field contains 'Resultado de medición' and has a message 'Debe ingresar un valor de medición.' below it. The 'Unidad' field is empty and has a message 'Debe seleccionar un unidad de medición' below it. The 'Fecha' field is set to '2015-10-13' and '18:48:46'. A green 'Save' button is at the bottom.

Figura 12: Mensaje sutil que aparece si se ha presionado el campo y no se ha escrito nada

The image shows a web form for recording measurements. It contains five main sections, each with a label, a text input, and a red error message below it:

- Tipo:** A dropdown menu is empty. Error: "Seleccione un tipo de medición."
- Valor:** A text input contains "Resultado de medición". Error: "Debe ingresar un valor de medición."
- Unidad:** A dropdown menu is empty. Error: "Debe seleccionar un unidad de medición"
- Fuente:** A dropdown menu is empty. Error: "Seleccione una fuente de medición."
- Fecha:** Two stacked text inputs. The top one contains "Date" and the bottom one contains "Time". Error: "Debe ingresar fecha y hora de medición."

At the bottom of the form is a green button labeled "Save".

Figura 13: Mensaje vistoso que aparece luego de borrar lo escrito

Listing 3: recurso que solicita las unidades de un tipo de medición específico

```
1
2 angular.module('saludWebApp')
3 .factory('MeasurementTypeUnit', function (global, $resource
4 ) {
5     // URL of specific API resource
6     var url=global.getApiUrl()+'/measurement_types/:id_type
7     /units';
8
9     return $resource( url,
10     { id_type: '@_id_type' },
11     { query:{method:'GET',isArray:false},
12     update: {method: 'PUT'}
13 });
14 });
```

Variable de acceso a la API global

Fue necesario añadir un servicio para manejar la dirección global de la API, dicho servicio se detalla en **Listing 4**

Listing 4: Servicio de la dirección global de la API

```
1 'use strict';
2
3 /**
4  * @ngdoc service
5  * @name saludWebApp.global
6  * @description
7  * # global
8  * Factory in the saludWebApp.
9  */
10 angular.module('saludWebApp')
11 .factory('global', function () {
12     // Service logic
13
14     // URL of yesdoc API
15     var _api_url='https://yesdoc-api.herokuapp.com';
16
17     // Public methods
```

```
18     return {
19         getApiUrl: function () {
20             return _api_url;
21         }
22     };
23 });
24 ~
```


2.7. Clases involucradas

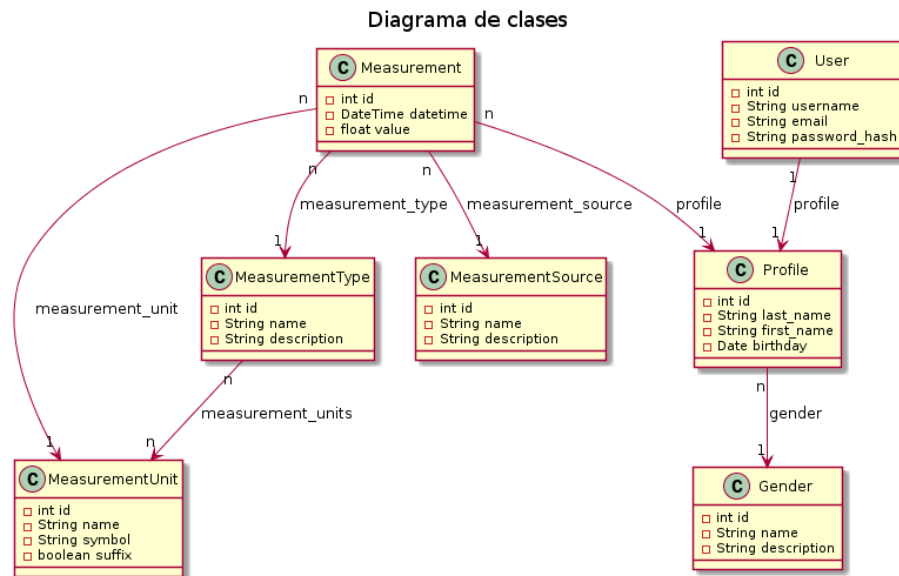


Figura 14: Diagrama de clases donde se puede ver la relación entre el tipo de medición y la unidad

Clase **MeasurementType**

Esta clase nos permitirá nombrar los tipos de medidas, hasta el momento hemos contemplado: peso, dimensión corporal (Ej:altura) y glucosa. Existen ciertas medidas que contemplan dos valores, estas serán agregadas en un sprint futuro.

Descripción de los atributos

- **name:** Nombre del tipo de medición(tipo string).
- **description:** Descripción del tipo de medición (tipo string).

Dirección del recurso:

```
1 <BASE URL>/measurement_types/{id}
```

Json generado por la API

```
1 {
2   "resource":
3   {
4     "name": "Peso",
```

```
5         "description": "Peso corporal de la persona.",
6         "id": 1
7     }
8 }
```

Clase MeasurementUnit

Esta clase nos permitirá nomenciar las unidades de medicion disponible para que el usuario pueda seleccionarlasy cdo realice la medición, hasta el momento hemos contemplado: Kilogramo, gramo, miligramos, metro, centimetro y milímetro.

Descripción de los atributos

- **id:** Identificador único de la unidad de medición(tipo int).
- **name :** Nombre de la unidad de medición (tipo string).
- **symbol :** Símbolo de la unidad de medición (tipo string).
- **suffix :** Variable booleana que indica si el símbolo de la unidad de medición es un sufijo (verdadero) o un prefijo (falso) del valor de la medición (tipo boolean).

Dirección del recurso

```
1 <BASE URL>/measurement_units/{id}
```

Json generado por la API

```
1 {
2     "resource":
3     {
4         "symbol": "Kg",
5         "suffix": true,
6         "name": "Kilogramo",
7         "id": 1
8     }
9 }
```

Pero fundamentalmente necesitamos el recurso que nos permite traer las unidades de medidas a partir de un tipo particular de medición, dicho recurso se accede por:

Dirección del recurso

```
1 <BASE URL>/measurement_types/{id}/units
```

Json generado por la API

Retorna la lista de unidades de medición relacionadas a un tipo de medición específico.

Listing 5: Json generado por la api

```
1
2 {    "resource":
3     [{
4       "id": 1,
5       "symbol": "Kg",
6       "suffix": true,
7       "name": "Kilogramo"
8     },
9     {
10      "id": 6,
11      "symbol": "g",
12      "suffix": true,
13      "name": "gramo"
14    }]
15 }
```

2.8. Estado final de pruebas

- ¿Qué fue bien?
 - Las cargas y ediciones se llevan a cabo correctamente.
- ¿Qué se mejoró?
 - **Cerrado** Al crear una nueva medición, se mostraba un cartel (alert de javascript) con una fecha, dicho alert fue eliminado.
 - **Cerrado** Se encontró un problema con la zona horaria que usa el servidor y la zona horaria del usuario, para solucionarlo hubo q hacer un casteo previo cuando se solicitaba la fecha y hora del usuario para mostrar.
 - **Cerrado** Solo debería mostrarse las unidades relacionadas al tipo de medición que se ha seleccionado

-
- **Cerrado** En el futuro se deberá mejorar las validaciones de los datos a la hora de cargar información en los formularios.
 - **Cerrado** Se deberá mejorar la manera de seleccionar la fecha y la hora.
 - **Cerrado** Deberá realizarse los carteles de advertencia necesarios.

3. Sprint 4

3.1. Planificación

Inicio: 1 de Julio del 2015

Fin: 24 de Julio del 2015

3.2. Descripción

Este sprint tiene por objetivo definir un proceso de autenticación del usuario para con la API, sin importar el perfil del mismo, que garantice un nivel de seguridad adecuado para tranquilidad en el uso de la aplicación por parte de los interesados.

3.3. User Stories relacionados

La **Tabla ??** indicará las características de cada user story para guiarnos en el desarrollo del sprint.

ID	Enunciado de la historia
US-11	Como paciente, quiero modificar los permisos de visualización de mis datos con respecto a cada uno de los integrantes de grupo familiar para tener un control total sobre mi privacidad.
US-21	Como usuario quiero contar con un acceso único y privado a mi información.

3.4. Modelo de datos

El Diagrama propio de este sprint se puede ver en la **Figura15**, allí se indican exactamente las clases que se usarán en este sprint y que serán detalladas con determinimiento en el presente documento.

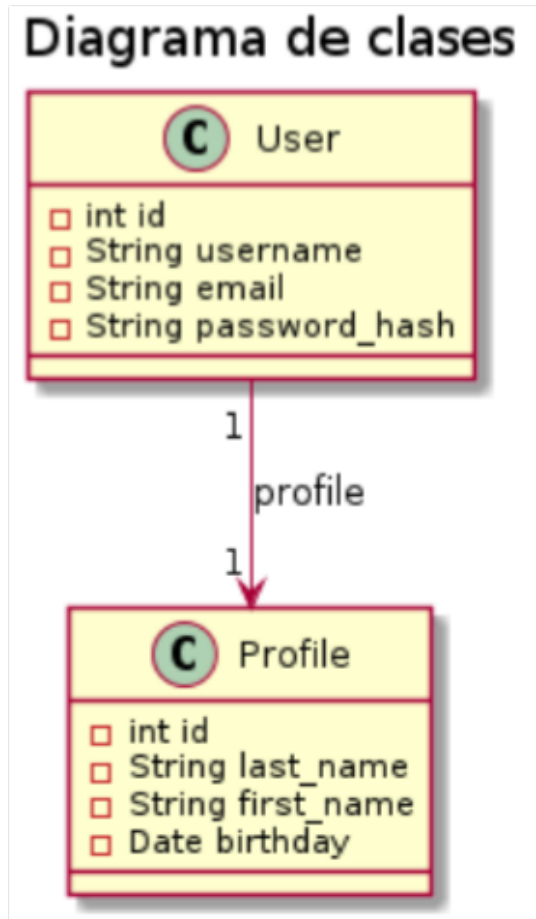


Figura 15: Diagrama de clases autenticación.

3.5. Modelo funcional.

Se define el diagrama de casos de uso del presente Sprint **[Figura 16]**.

Area a cargo	Responsable	Tarea	US
Backend	Michael Manganiello	Generación del modelo User y relación del mismo con el modelo Profile.	US-?? & US-??
Backend	Michael Manganiello	Generación del recurso relacionado con el modelo User y los métodos post y get que manejan los operadores HTTP correspondientes.	US-?? & US-??
Backend	Michael Manganiello	Generación del recurso para obtener un token para un usuario autenticado.	US-?? & US-??

3.5.1. Definición de modelos

La definición del modelo User es bastante compleja ya que define métodos para tomar la password y guardarla como un hash, a su vez, puede recibir passwords encriptadas y desencriptarlas. Por otro lado, define métodos para la generación y

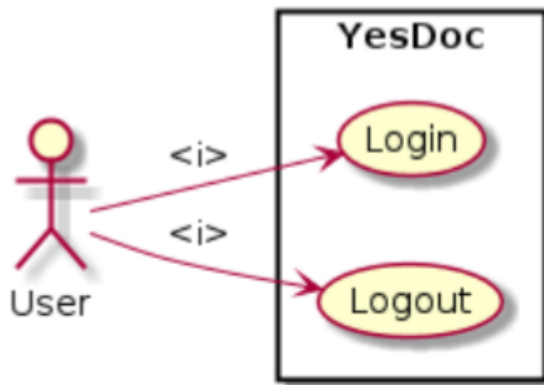


Figura 16: Casos de uso autenticación

verificación del token asignado al usuario. Por último, define una relación uno a uno con el modelo Profile.

3.5.2. Recursos

El desafío en cuanto a la definición de recursos radica en que según lo que establece REST, la restricción de que la API debe ser stateless invalida la posibilidad de usar sesiones para que la API sea escalable, es por esto que definimos una solución que se presenta en un punto gris entre puristas de REST y quienes realmente no hacen REST. La solución consiste en generar un token para el usuario autenticado el cual se almacena en las cookies y es usado en cada solicitud para autenticar al usuario. Por esto se definen recursos para dar de alta al usuario y para entregarle un token.

```
1  api.add_resource(UserView, '/users/<int:id>')
2  api.add_resource(UserList, '/users')
3  api.add_resource(Token, '/token')
4  api.add_resource(MyUserView, '/my/user')
5
```

3.6. Salidas del Sistema

1. Solicitud post al recurso del perfil

Para crear un usuario debe existir un perfil creado, para esto usamos el recurso

“profiles” a través del método POST y pasando como argumento los datos first_name, last_name, birthday y gender_id.

2. Solicitud post al recurso del user

Realizamos ahora una solicitud HTTP, con método post utilizando curl al recurso del usuario usando el id del perfil creado previamente:

```
curl -i http://localhost:5000/users -H "Content-Type: application/json" -X POST -d '{"username":"akathy", "email":"kathy@gmail.com", "password":"kathy1234", "profile_id":"2"}'
```

Obtenemos la siguiente respuesta de la API, con un código 201.

```
1 HTTP/1.0 201 CREATED
2 Content-Type: application/json
3 Content-Length: 425
4 Server: Werkzeug/0.10.4 Python/2.7.6
5 Date: Tue, 20 Oct 2015 05:21:37 GMT
6
7 {
8     "resource": {
9         "email": "kathy@gmail.com",
10        "id": 2,
11        "profile": {
12            "birthday": "1989-06-17",
13            "first_name": "Katherina",
14            "gender": {
15                "description": "female gender",
16                "id": 2,
17                "name": "female"
18            },
19            "id": 2,
20            "last_name": "Aguirre"
21        },
22        "username": "akathy"
23    }
24 }
```

3. Solicitud post al recurso token usando user y password

Luego con este usuario y contraseña solicitamos un token al recurso correspondiente:

```
curl -u akathy:kathy1234 http://localhost:5000/token
```

Obtenemos así el token que debe ser almacenado en las cookies, tendrá una duración de 10 minutos y será utilizado en cada solicitud para autenticación.

```
1 {
2   "resource": {
3     "duration": 600,
4     "token": "eyJhbGciOiJIUzI1NiIsImV4cCI6MTQ0
      NTMxOTM3MCwiaWF0IjoxNDQ1MzE4NzcwfQ.eyJpZCI6
      Mn0.2eZRjbMq9tg4GykJx8EU-Ux4ZoyUW6
      WnBlADsvnpQvE"
5   }
6 }
```

3.7. Criterios de aceptación

Criterio de aceptación			
Id	Contexto	Evento	Resultado
1	En caso de que exista un usuario registrado con el mismo username	al ejecutar el método post del recurso users	El sistema devolverá un json vacío y un código de error 400
2	En caso de que exista al menos un usuario registrado	al ejecutar el método get del recurso users	El sistema devolverá una lista de json con los datos de los users registrados
3	En caso de que exista un usuario registrado con el mismo username	al ejecutar el método post del recurso users	El sistema devolverá un json vacío y un código de error 400

4	En caso de que no exista un usuario registrado con el id indicado	al ejecutar el método getput del recurso usersid	El sistema devolverá un json con un mensaje de error y un código de error 404
---	---	--	---

3.8. Casos de prueba

3.8.1. Pruebas de integración entre módulos del Sistema

3.8.2. Pruebas de carga

3.8.3. Pruebas de seguridad por niveles de usuarios

3.9. Pruebas ejecutadas

Aquí se realizará una conclusión general de lo que se descubrió en las pruebas.

- **¿Que fue bien?**
 - Las cargas y ediciones se llevan a cabo correctamente.
- **¿Que se mejoró?**
 - Cerrado problema
- **¿Que se puede mejorar?**
 - Abierto En el futuro se deberá mejorar ...

4. Sprint 5

4.1. Planificación

Inicio: Martes 18 de setiembre del 2015

Fin: Martes 17 de octubre del 2015

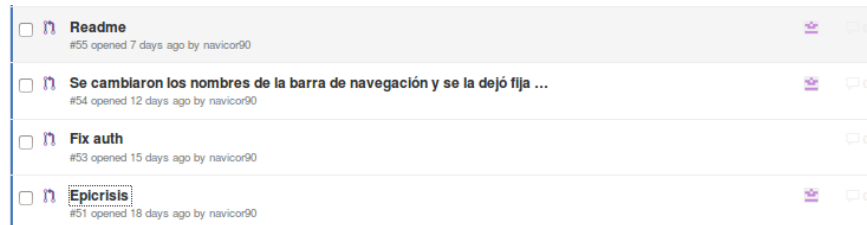


Figura 17: Pull request realizados en el sprint 5

4.2. Descripción

A partir de las funcionalidades ya existentes de carga de medición, se va a crear un análisis consistente y representativo de un análisis de la vida real.

En este sprint se le permitirá al usuario poder cargar las imágenes y las mediciones correspondientes a un mismo análisis, indicando la fecha y hora en la que fue realizado. Esta funcionalidad es muy importante ya que refleja el análisis que el paciente se realiza y le permitirá al medico poder contrastar la validez de las mediciones con un análisis certificado.

Otra funcionalidad importante a desarrollar son las vistas necesarias para mostrar los análisis y el contenido del mismo

4.3. User Stories relacionados

La **Tabla ??** indicará las características de cada user story para guiarnos en el desarrollo del sprint.

4.4. Modelo de datos

El Diagrama propio de este sprint se puede ver en la **Figura 18**, allí se indican exactamente las clases que se usarán en este sprint y que serán detalladas con detenimiento en el presente documento.

4.5. Modelo funcional.

Se describirán las funciones usando como marco de apoyo el sprint Backlog, además se armará el diagrama de casos de uso del presente Sprint [**Figura 44**] que irá creciendo medida se vaya avanzando en el proyecto.

ID	Enunciado de la historia
US-2	Como paciente, quiero añadir al sistema los estudios realizados para evitar posibles perdidas.
US-5	Como paciente quiero que los sistemas de salud existentes puedan cargar sus resultados directamente en mi carpeta de salud para centralizar mi información.
US-7	Como paciente quiero categorizar mis estudios por rama de medicina, para lograr una mejor organización y navegabilidad en el sistema.
US-8	Como laboratorio, quiero cargar información de un paciente en su cuenta para ahorrarle las molestias de volver.

Area a cargo	Responsable	Tarea	US
Front-end	Ivan Terreno	Diseño de Interface de epicrisis	US 2
Front-end	Ivan Terreno	Enlazado de interfaces ya existentes de mediciones con la nueva interfaz	US 2
Front-end	Ivan Terreno	Creación de los recursos que consumirán la información de la API	US 2& US-??
Front-end	Ivan Terreno	Búsqueda de iconos representativos	US-?? & US 2
Front-end	Ivan Terreno	Detección de tipo de archivo	US-?? & US 2
Front-end	Ivan Terreno	Carga de archivos,por POST	US-?? & US 2
Front-end	Ivan Terreno	funcionalidad de editar y ver archivos y análisis	US-?? & US 2

4.6. Salidas del Sistema - Incrementos

Luego de finalizado este user story se obtendrán 5 pantallas que se detallarán a continuación:

1. **Vista para acceder a carga de nueva medición:** [Figura 19] En la imagen se muestra la forma que tiene el usuario de acceder a realizar la carga de un análisis.
2. **Crear Análisis:** [Figura 20] En la imagen se muestra el formulario que corresponde a un análisis. Por lo general los análisis son un conjunto de mediciones las cuales suelen provenir de un estudio presentada en papel. Por eso en el formulario contemplamos dos campos el de carga de mediciones y el de carga de imágenes que a continuación se detallan.
3. **Carga de mediciones:** [Figura 22] Se le permitirá cargar mediciones que realice en algún momento del día como son peso, altura, grasa corporal y glucosa.

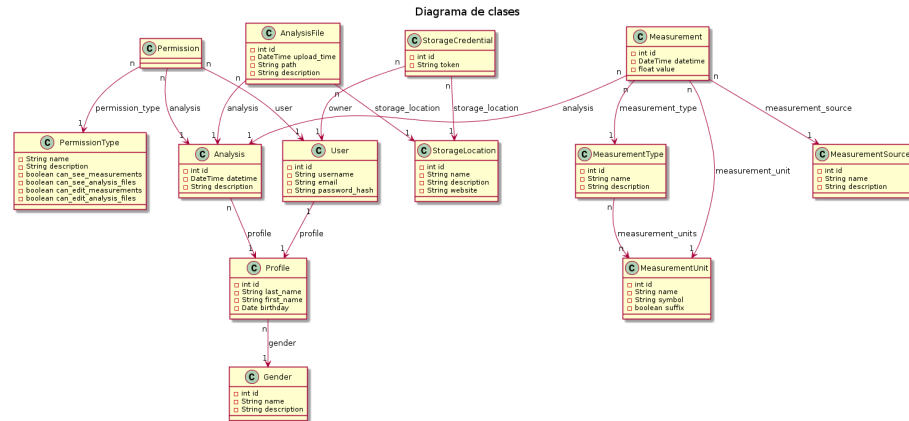


Figura 18: Diagrama de clases referido a la carga de análisis

Deberá indicar la fuente, tipo, unidad y fecha de la medición. Este formulario es el mismo que el descrito en el **Sprint 2**.

4. **Carga de imágenes [Figura 21]** Como se explicó anteriormente se le permite al usuario cargar imágenes correspondiente al estudio/análisis realizado.
5. **Imágenes y mediciones cargadas en el formulario: [Figura 23]** Esta interfaz destaca la funcionalidad de edición, eliminación y vista previa que brinda el formulario.

YesDoc Histórico Resumen Perfil

Nueva Medición Nuevo Análisis

Mediciones

Peso: 55 Kg
2015-10-19 22:57:55
Manual

Altura: 1.3 m
2015-06-16 20:18:13
Manual

Figura 19: vista para acceder a cargar nuevo análisis

The screenshot shows the 'Análisis' form in the YesDoc application. The form is titled 'Análisis' and is located within a green header bar that also contains navigation links: 'YesDoc', 'Histórico', 'Resumen', 'Perfil', and 'Mi cuenta'. The form fields include: 'Fecha:' with two input fields containing '2015-10-20' and '14:37:30'; 'Descripción:' with a text input field containing 'Descripción'; 'Adjuntos:' with a blue plus icon and a large empty text area; and 'Mediciones:' with a blue plus icon and a large empty text area. A green 'Save' button is located at the bottom right of the form.

Figura 20: formulario para la creación de análisis

The screenshot shows the 'Adjunto' modal form in the YesDoc application. The modal is titled 'Adjunto' and is located within a green header bar that also contains navigation links: 'YesDoc', 'Histórico', 'Resumen', 'Perfil', and 'Mi cuenta'. The modal fields include: 'Fecha:' with two input fields containing '2015-10-20' and '14:37:30'; 'Descripción:' with a text input field containing 'Una descripción'; and a green 'Save' button at the bottom right. The modal also features a 'Examinar...' button and a message 'No se seleccionó un archivo.' above a file input field.

Figura 21: formulario de carga de imágenes

Medición

Tipo:

Valor:

Unidad:
Debe seleccionar un tipo de medición

Fuente:

Fecha:

Figura 22: formulario de carga de medición

Análisis

Fecha:

Descripción:

Adjuntos:

1	6378581293_b6b6081a1_b.jpg	<input type="button" value="📄"/>	<input type="button" value="✏"/>	<input type="button" value="🗑"/>
---	----------------------------	----------------------------------	----------------------------------	----------------------------------

Mediciones:

1	Peso	65	Kg	<input type="button" value="✏"/>	<input type="button" value="🗑"/>
2	Peso	75	Kg	<input type="button" value="✏"/>	<input type="button" value="🗑"/>
3	Altura	170	cm	<input type="button" value="✏"/>	<input type="button" value="🗑"/>

Figura 23: Función de editar, borrar y visualizar

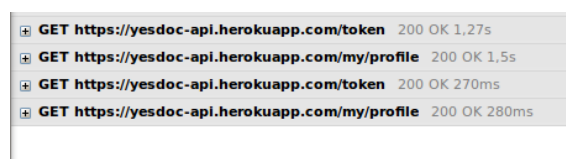
4.7. Criterios de aceptación

Criterio de aceptación			
Id	Contexto	Evento	Resultado
1	En caso de que exista una persona sin mediciones	cuando este desee observar sus mediciones	El sistema no mostrará nada
2	Cuando el usuario registrado ingresa dos mediciones del mismo tipo	y luego quiera consultarlas	El sistema solo le mostrará la ultima medición, del mismo tipo, realizada
3	Cuando el usuario seleccione una medición	y luego quiera editarla	El sistema le permitira la correspondiente edición
4	Si el usuario existe y no está logeado	y quiera ingresar a ver sus mediciones.	El sistema no le permitirá ingresar

4.8. Casos de prueba

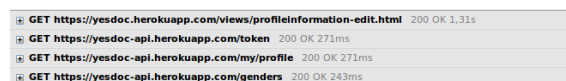
4.8.1. Pruebas de integración entre módulos del Sistema

Actor	Sistema	Resultado Esperado
-------	---------	--------------------



+	GET https://yesdoc-api.herokuapp.com/token	200 OK 1,27s
+	GET https://yesdoc-api.herokuapp.com/my/profile	200 OK 1,5s
+	GET https://yesdoc-api.herokuapp.com/token	200 OK 270ms
+	GET https://yesdoc-api.herokuapp.com/my/profile	200 OK 280ms

Figura 24: Respuesta de la API al loguearse



+	GET https://yesdoc.herokuapp.com/views/profileinformation-edit.html	200 OK 1,31s
+	GET https://yesdoc-api.herokuapp.com/token	200 OK 271ms
+	GET https://yesdoc-api.herokuapp.com/my/profile	200 OK 271ms
+	GET https://yesdoc-api.herokuapp.com/genders	200 OK 243ms

Figura 25: Respuesta de la API al mostrar perfil

```
+ GET https://yesdoc-api.herokuapp.com/token 200 OK 256ms
+ GET https://yesdoc-api.herokuapp.com/my/measurements/latest 200 OK 309ms
```

Figura 26: Respuesta de la API al mostrar últimas mediciones

```
+ GET https://yesdoc-api.herokuapp.com/measurement_types 200 OK 261ms
+ GET https://yesdoc-api.herokuapp.com/measurement_sources 200 OK 255ms
```

Figura 27: Respuesta de la API al crear nuevo análisis

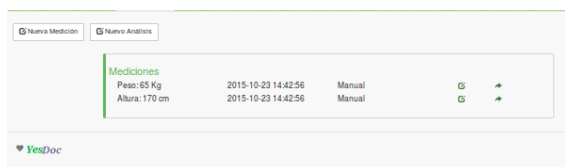


Figura 28: Perfil de mediciones

```
+ GET https://yesdoc-api.herokuapp.com/token 200 OK 238ms
+ GET https://yesdoc-api.herokuapp.com/measurement_types 200 OK 711ms
+ GET https://yesdoc-api.herokuapp.com/measurement_sources 200 OK 251ms
+ GET https://yesdoc-api.herokuapp.com/measurements/103 200 OK 305ms
+ GET https://yesdoc-api.herokuapp.com/measurement_types/1/units 200 OK 285ms
```

Figura 29: Respuesta de la API al editar mediciones

```
+ PUT https://yesdoc-api.herokuapp.com/measurements/103 200 OK 713ms
+ GET https://yesdoc-api.herokuapp.com/token 200 OK 430ms
+ GET https://yesdoc-api.herokuapp.com/my/measurements/latest 200 OK 566ms
```

Figura 30: Respuesta de la API al editar nuevo análisis

```
+ GET https://yesdoc.herokuapp.com/views/home.html 304 Not Modified 755ms veni
+ GET https://yesdoc.herokuapp.com/views/weight.html 304 Not Modified 218ms veni
+ GET https://yesdoc-api.herokuapp.com/token 200 OK 198ms
+ GET https://yesdoc.herokuapp.com/views/partials/measurement_same_type.htm veni
+ GET https://yesdoc-api.herokuapp.com/token 200 OK 323ms
+ GET https://yesdoc-api.herokuapp.com/measurement_types 200 OK 307ms
+ GET https://yesdoc-api.herokuapp.com/my/measurements?type=1 200 OK 669ms
```

Figura 31: Respuesta de la API al mostrar gráficas



Figura 32: Vista de gráfica y tablas de mediciones

```

GET https://yesdoc.herokuapp.com/views/height.html 200 OK 1,42s
GET https://yesdoc-api.herokuapp.com/token 200 OK 214ms
GET https://yesdoc-api.herokuapp.com/measurement_types 200 OK 217ms
GET https://yesdoc-api.herokuapp.com/my/measurements?type=2 200 OK 404ms

```

Figura 33: Respuesta de la API al mostrar la gráfica de altura

```

GET https://yesdoc-api.herokuapp.com/token 200 OK 205ms
GET https://yesdoc-api.herokuapp.com/token 200 OK 208ms
GET https://yesdoc-api.herokuapp.com/my/analyses 200 OK 230ms
GET https://yesdoc-api.herokuapp.com/analysis/49/files 200 OK 255ms

```

Figura 34: Respuesta de la API al mostrar los análisis

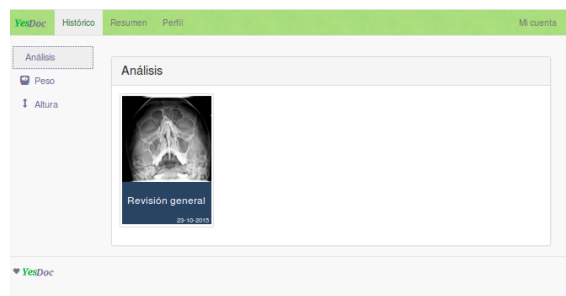


Figura 35: Vista de la lista de análisis

```

+ GET https://yesdoc-api.herokuapp.com/token 200 OK 2,04s
+ GET https://yesdoc-api.herokuapp.com/analysis/49/measurements 200 OK 2,19s
+ GET https://yesdoc-api.herokuapp.com/token 200 OK 630ms
+ GET https://yesdoc-api.herokuapp.com/analysis/49 200 OK 523ms
+ GET https://yesdoc-api.herokuapp.com/analysis/49/files 200 OK 822ms

```

Figura 36: Respuesta de la API al consultar los análisis

#	Tipo	Valor	Unidad	Fuente	Fecha
1	Peso	75Kg	Kilogramo	Manual	23/10/2015 14:42
2	Altura	170cm	centímetros	Manual	23/10/2015 14:42
3	Peso	65Kg	Kilogramo	Manual	13/10/2015 07:42

Figura 37: Vista del contenido del análisis

```

+ GET https://yesdoc.herokuapp.com/views/logoff.html 200 OK 880ms
+ GET https://yesdoc.herokuapp.com/views/login.html 200 OK 198ms
+ GET https://yesdoc-api.herokuapp.com/token 401 UNAUTHORIZED 971ms

```

Figura 38: Respuesta de la API al cerrar sesión

4.8.2. Pruebas de carga

4.8.3. Pruebas de seguridad por niveles de usuarios

4.9. Pruebas ejecutadas

Aqui se realizará una conclusión general de lo que se descubrió en las pruebas.

- **¿Que fue bien?**
 - Las cargas y ediciones se llevan a cabo correctamente.
- **¿Que se mejoró?**
 - **Cerrado** problema
- **¿Que se puede mejorar?**
 - **Abierto** En el futuro se deberá mejorar ...

5. Sprint 6

5.1. Planificación

Inicio: 16 de septiembre del 2015

Fin: 05 de octubre del 2015

5.2. Descripción

En este sprint se busca brindar al usuario la funcionalidad para poder cargar imágenes, así este puede dejar un respaldo digital de lo que puede ser un análisis clínico o una imagen de radiografía y demás documentos médicos físicos. Para el mismo se plantean dos variantes, una local y otra en un servicio de almacenamiento, debido a que el desarrollo de ambas presentan sus diferencias y a su vez existen diferentes servicios de almacenamiento, se debe prever una estructura que soporte el impacto frente al cambio.

5.3. User Stories relacionados

La **Tabla ??** indicará las características de cada user story para guiarnos en el desarrollo del sprint.

ID	Enunciado de la historia
US-1	Como paciente, quiero añadir información de mi perfil de salud o mediciones regulares para que el médico cuente con más y mejor información al momento de realizar el diagnóstico.
US-2	Como paciente, quiero añadir al sistema los estudios realizados para evitar posibles pérdidas.
US-16	Como paciente, quiero acceder a mis documentos desde cualquier lugar para hacer uso de ellos cuando los necesite.

5.4. Modelo de datos

El Diagrama propio de este sprint se puede ver en la **Figura39**, allí se indican exactamente las clases que se usarán en este sprint y que serán detalladas con de-

tenimiento en el presente documento. Se recuerda que se ha realizado un Diagrama de clases específico para este sprint y puede variar en futuras iteraciones.

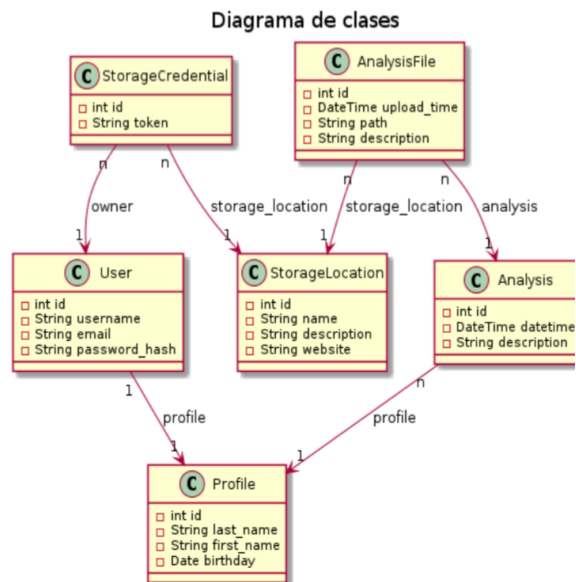


Figura 39: Clases para la gestión de archivos.

5.5. Modelo funcional.

Se describirán las funciones usando como marco de apoyo el sprint Backlog, además se armará el diagrama de casos de uso del presente Sprint [**Figura 40**] que irá creciendo medida se vaya avanzando en el proyecto.

EJEMPLO

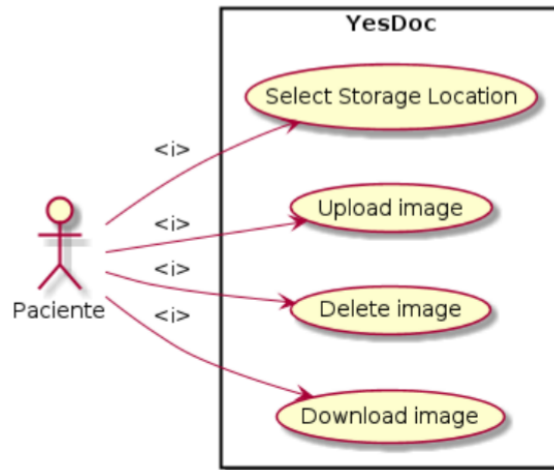


Figura 40: CU-Gestión de archivos

Area a cargo	Responsable	Tarea	US
Back-end	Franco Canizo	Se define la estructura para la gestión de archivos	US-?? & US-??
Back-end	Franco Canizo	Se crea el adaptador para la gestión de archivos en dropbox y en los servidores de yesdoc, se crea el recurso para la carga de archivos de análisis.	US-?? & US-??
Back-end	Michael Manganiello	Se añaden las clases del dominio y funcionalidades para el manejo de análisis, archivos asociados, y la gestión de credenciales a diferentes servicios de almacenamiento personales del usuario	US-?? & US-??
Back-end	Michael Manganiello	Creación de la migración para los nuevos modelos	US-?? & US-??
Back-end	Michael Manganiello	Creación de fields y parsers para las representaciones de los nuevos recursos	US-?? & US-??
Back-end	Michael Manganiello	Correcciones generales de integración, claves de la aplicación en Dropbox, devolución y eliminación de archivos	US-?? & US-??

5.5.1. Estructura conexión con diferentes medios de almacenamiento

Se define el paquete adapters en el cual se define una fábrica encargada de la creación del adaptador específico para la conexión y gestión de archivos con el medio de almacenamiento específico. Para la situación actual del sistema se definen dos adaptadores específicos, uno para el almacenamiento de archivos local en el servidor

de YesDoc y otro para el almacenamiento de archivos en Dropbox. El [Modelo 41] específico es el siguiente.

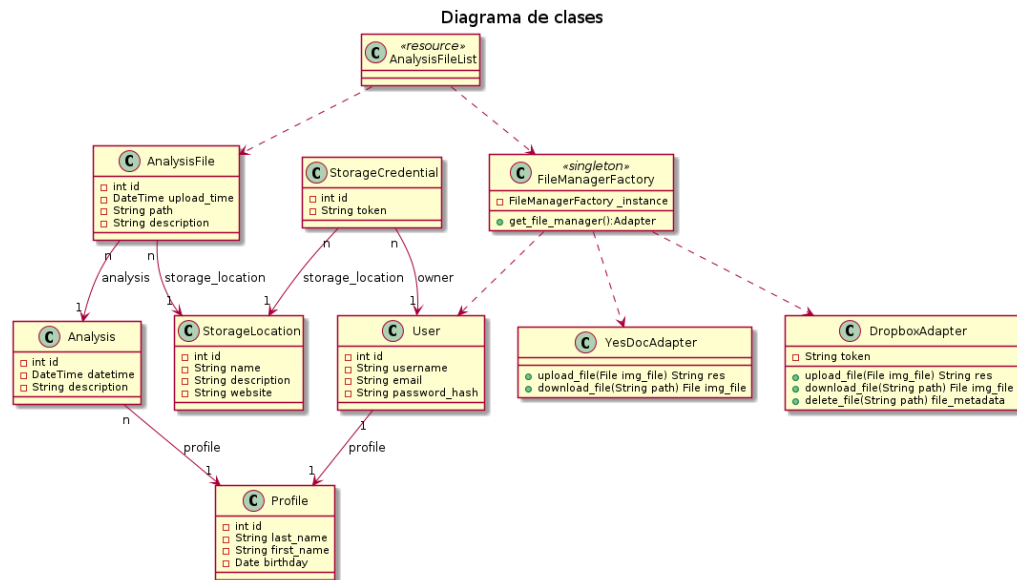


Figura 41: DCD-Carga de archivos

5.5.2. Creación de clases y funcionalidades para la gestión de archivos

Se crean las clases del modelo lógico que definimos en la **Figura39**, de cada una de estas se definen los atributos siguiendo la documentación de SQLAlchemy, ORM utilizado para mapeo de las entidades de la API, sus relaciones y funcionalidades. Un ejemplo es la clase StorageCredential:

```

1
2 class StorageCredential(db.Model):
3     # Attributes
4     id = db.Column(db.Integer, primary_key=True)
5     token = db.Column(db.String(255))
6     # Foreign keys
7     owner_id = db.Column(db.Integer, db.ForeignKey('user.id'))
8     storage_location_id = db.Column(db.Integer, db.ForeignKey('storage_location.id'))
9     # Relationships
10    owner = db.relationship('User',

```

```

11         backref=db.backref('
storage_credentials ', lazy='dynamic'))
12     storage_location = db.relationship('StorageLocation ',
13         backref=db.backref('
storage_credentials ', lazy='dynamic'))
14
15     def __init__(self, token, owner_id, storage_location_id):
16         self.token = token
17         self.owner_id = owner_id
18         self.storage_location_id = storage_location_id
19
20     def __repr__(self):
21         return '<StorageCredential: %i>' % self.id

```

Esta clase del modelo guarda el token generado por el medio de almacenamiento, en este caso Dropbox, la relación con el StorageLocation que es el medio de almacenamiento y el User al que pertenece esta credencial.

5.5.3. Creación de la migración para los nuevos modelos

Una vez definidas las clases se ejecuta el comando db del script de python manager.py con la opción migrate lo que crea un script sql con la nueva versión de la base de datos. Luego se ejecuta con la opción upgrade lo que ejecuta el script y carga la base de datos. Eso es posible gracias al uso de las extensiones **flask-migrate** para las migraciones de la base de datos y **flask-script** para la ejecución de comandos externos.

5.5.4. Creación de fields y parsers para las representaciones de los nuevos recursos

Hemos definido nuevas clases. Para que el cliente se comuniqué con las mismas la API define recursos que manejan estas clases y representaciones para una interfaz única para con los recursos. El recurso que maneja las credenciales de almacenamiento es el siguiente:

```

1 class StorageCredentialList(Resource):
2     @marshal_with(StorageCredentialFields.resource_fields, envelope='
resource ')
3     def get(self):
4         storage_credentials = StorageCredential.query.all()
5         return storage_credentials
6     @marshal_with(StorageCredentialFields.resource_fields, envelope='
resource ')

```

```

7     def post(self):
8         args = parser_post.parse_args()
9         new_storage_credential = StorageCredential(args['token'],
10                                                    args['owner_id'],
11                                                    args['
storage_location_id'])
12         db.session.add(new_storage_credential)
13         db.session.commit()
14         return new_storage_credential, 201

```

La representación para las solicitudes HTTP del cliente al servidor.

```

1 # Parser general
2 parser = reqparse.RequestParser()
3 parser.add_argument('token', type=str, required=True)
4 parser.add_argument('owner_id', type=is_valid_id, required=True)
5 parser.add_argument('storage_location_id', type=is_valid_id, required=
    True)
6 # Parser para recurso POST
7 parser_post = parser.copy()
8 # Parser para recurso PUT
9 parser_put = parser.copy()
10 # Parser para recurso POST con usuario autenticado
11 parser_post_auth = parser.copy().remove_argument('owner_id')

```

La representación para las respuestas del servidor al cliente.

```

1 class StorageCredentialFields:
2     resource_fields = {
3         'id': fields.Integer,
4         'token': fields.String,
5         'owner': fields.Nested(UserFields.resource_fields),
6         'storage_location': fields.Nested(StorageLocationFields.
resource_fields),
7     }
8     required = ['id',
9                'token',
10               'owner',
11               'storage_location']

```

5.6. Salidas del Sistema - Incrementos

Carga de una credencial de almacenamiento

Para la carga debemos utilizar el recurso que la API presenta con el identificador único URL **storage_credentials**. Suponemos que previamente el usuario de prueba

ha autorizado a nuestra aplicación a que utilice espacio del almacenamiento de su cuenta de dropbox y por lo tanto contamos con el token: “Jl0_uroqYBoAAAAAAAF6KUxPA1gAMTqFf9ES2S0zZl_27V5QAmEn5V58IUxcck1”. Suponemos que ha sido cargado en API un usuario con id 1 y un storage location con id 2. Teniendo en cuenta la representación definida para la solicitud al recurso y que debemos usar el método POST para cargar la credencial, ejecutamos la siguiente línea CURL:

```
curl -i http://localhost:5000/storage_credentials -H "Content-Type: application/json" -X POST -d '{"token": "Jl0_uroqYBoAAAAAAAF6KUxPA1gAMTqFf9ES2S0zZl_27V5QAmEn5V58IUxcck1", "owner_id": "1", "storage_location_id": "1"}'
```

La API nos responde con la siguiente respuesta:

```
1 HTTP/1.0 201 CREATED
2 Content-Type: application/json
3 Content-Length: 812
4 Server: Werkzeug/0.10.4 Python/2.7.6
5 Date: Tue, 20 Oct 2015 19:10:33 GMT
6
7 {
8     "resource": {
9         "id": 2,
10        "owner": {
11            "email": "fncañizo@gmail.com",
12            "id": 1,
13            "profile": {
14                "birthday": "1990-06-20",
15                "first_name": "Franco",
16                "gender": {
17                    "description": "male gender",
18                    "id": 1,
19                    "name": "male"
20                },
21                "id": 1,
22                "last_name": "Canizo"
23            },
24            "username": "coco19"
25        },
26    },
27 }
```

```

26         "storage_location": {
27             "description": "yesdoc file manager",
28             "id": 2,
29             "name": "YesDoc",
30             "website": "http://yesdoc.herokuapp.com"
31         },
32         "token": "Jl0_uroqYBoAAAAAAAF6KUxPAlgAMTqFf9ES2S
33             0zZl_27V5QAmEn5V58IUxcck1"
34     }

```

5.7. Criterios de aceptación

Ejemplo - esto se debe modificar

Criterio de aceptación			
Id	Contexto	Evento	Resultado
1	En caso de que exista una persona sin mediciones	cuando este desee observar sus mediciones	El sistema no mostrará nada

5.8. Casos de prueba

5.8.1. Pruebas de integración entre módulos del Sistema

5.8.2. Pruebas de carga

5.8.3. Pruebas de seguridad por niveles de usuarios

5.9. Pruebas ejecutadas

Aquí se realizará una conclusión general de lo que se descubrió en las pruebas.

- ¿Que fue bien?
 - Las cargas y ediciones se llevan a cabo correctamente.

-
- ¿Que se mejoró?
 - Cerrado problema
 - ¿Que se puede mejorar?
 - Abierto En el futuro se deberá mejorar ...

6. Sprint 7

6.1. Planificación

Inicio: ?? del 2015

Fin: ?? de octubre del 2015

6.2. Descripción

6.3. User Stories relacionados

La **Tabla ??** indicará las características de cada user story para guiarnos en el desarrollo del sprint.

ID	Enunciado de la historia
US-2	Como paciente, quiero añadir al sistema los estudios realizados para evitar posibles perdidas.

6.4. Modelo de datos

El Diagrama propio de este sprint se puede ver en la **Figura1**, allí se indican exactamente las clases que se usarán en este sprint y que serán detalladas con detenimiento en el presente documento. Se recuerda que se ha realizado un Diagrama de clases tentativo que se puede ver en la **Figura 1**, dicho diagrama será utilizado como base para este sprint y posee un alcance limitado el cual se irá modificando a medida que se profundice en los temas.

6.5. Modelo funcional.

Se describirán las funciones usando como marco de apoyo el sprint Backlog, además se armará el diagrama de casos de uso del presente Sprint [Figura 44] que irá creciendo medida se vaya avanzando en el proyecto.

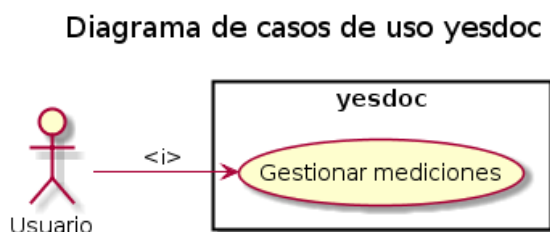


Figura 42: formulario de edición de perfil

Area a cargo	Responsable	Tarea	US
Front-end	Ivan Terreno	Generación de controladores para consumir Json de la Api relacionados a la API	US-?? & US-??

6.5.1. Tarea a describir

6.6. Salidas del Sistema - Incrementos

Esto es un ejemplo. Debe listarse las pantallas y explicar que hacen

1. **Presentación de las últimas mediciones [Figura 3]** con posibilidad de edición de cada una de las mediciones. Los datos posible a presentar son altura, peso, grasa corporal y glucosa.

La interfaz mostrará el valor de la medición, la fecha y hora en que fue realizada y la fuente que se utilizó para dicha medición.

6.7. Criterios de aceptación

Ejemplo - esto se debe modificar

Criterio de aceptación			
Id	Contexto	Evento	Resultado
1	En caso de que exista una persona sin mediciones	cuando este desee observar sus mediciones	El sistema no mostrará nada

6.8. Casos de prueba

6.8.1. Pruebas de integración entre módulos del Sistema

6.8.2. Pruebas de carga

6.8.3. Pruebas de seguridad por niveles de usuarios

6.9. Pruebas ejecutadas

Aquí se realizará una conclusión general de lo que se descubrió en las pruebas.

- **¿Que fue bien?**
 - Las cargas y ediciones se llevan a cabo correctamente.
- **¿Que se mejoró?**
 - Cerrado problema
- **¿Que se puede mejorar?**
 - Abierto En el futuro se deberá mejorar ...

7. Sprint 8

7.1. Planificación

Inicio: ?? del 2015

Fin: ?? de octubre del 2015

7.2. Descripción

7.3. User Stories relacionados

La **Tabla ??** indicará las características de cada user story para guiarnos en el desarrollo del sprint. **ESTO ES UN EJEMPLOOOO HAY QUE INDICAR TDOS LOS US RELACIONADO**

ID	Enunciado de la historia
US-2	Como paciente, quiero añadir al sistema los estudios realizados para evitar posibles perdidas.

7.4. Modelo de datos

El Diagrama propio de este sprint se puede ver en la **Figura1**, allí se indican exactamente las clases que se usarán en este sprint y que serán detalladas con detenimiento en el presente documento. Se recuerda que se ha realizado un Diagrama de clases tentativo que se puede ver en la **Figura 1**, dicho diagrama será utilizado como base para este sprint y posee un alcance limitado el cual se irá modificando a medida que se profundice en los temas.

AÑADIR DIAGRAMA DE CALSEEEES

7.5. Modelo funcional.

Se describirán las funciones usando como marco de apoyo el sprint Backlog, además se armará el diagrama de casos de uso del presente Sprint [**Figura 44**] que irá creciendo medida se vaya avanzando en el proyecto.

CARGAR CASO DE USOO

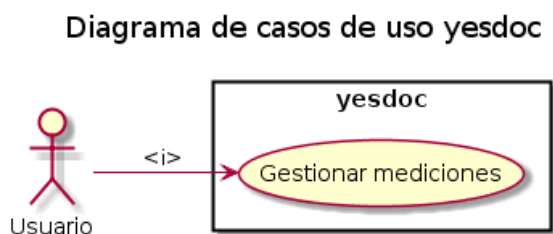


Figura 43: formulario de edición de perfil

EJEMPLO

Area a cargo	Responsable	Tarea	US
Front-end	Ivan Terreno	Generación de controladores para consumir Json de la Api relacionados a la API	US-?? & US-??

Describimos cada tarea???

7.5.1. Creación de página de mediciones

7.6. Salidas del Sistema - Incrementos

Esto es un ejemplo. Debe listarse las pantallas y explicar que hacen

-
1. **Presentación de las últimas mediciones [Figura 3]** con posibilidad de edición de cada una de las mediciones. Los datos posible a presentar son altura, peso, grasa corporal y glucosa.

La interfaz mostrará el valor de la medición, la fecha y hora en que fue realizada y la fuente que se utilizó para dicha medición.

7.7. Criterios de aceptación

Ejemplo - esto se debe modificar

Criterio de aceptación			
Id	Contexto	Evento	Resultado
1	En caso de que exista una persona sin mediciones	cuando este desee observar sus mediciones	El sistema no mostrará nada

7.8. Casos de prueba

7.8.1. Pruebas de integración entre módulos del Sistema

7.8.2. Pruebas de carga

7.8.3. Pruebas de seguridad por niveles de usuarios

7.9. Pruebas ejecutadas

Aquí se realizará una conclusión general de lo que se descubrió en las pruebas.

- **¿Que fue bien?**
 - Las cargas y ediciones se llevan a cabo correctamente.
- **¿Que se mejoró?**
 - Cerrado problema
- **¿Que se puede mejorar?**
 - **Abierto** En el futuro se deberá mejorar ...

8. Programación y documentación

Este apartado tiene como objetivo describir en que consiste el desarrollo y documentación asociado a una parte del sistema a mostrar, para ejemplificar a partir de las historias de usuario involucradas, cómo es la forma de trabajo.

Se ha seleccionado dos *EPICS* del sistema, que a continuación se detalla, para describirlos, el código desarrollado se encuentra en el **Anexo A.1**

1. *Permitir al usuario añadir los análisis realizados*

User Stories relacionados

- **US-2** Como paciente, quiero añadir al sistema los estudios realizados para evitar posibles perdidas.
- **US-5** Como paciente quiero que los sistemas de salud existentes puedan cargar sus resultados directamente en mi carpeta de salud para centralizar mi información
- **US-7** Como paciente quiero categorizar mis estudios por rama de medicina, para lograr una mejor organización y navegabilidad en el sistema
- **US-8** Como laboratorio, quiero cargar información de un paciente en su cuenta para ahorrarle las molestias de volver

2. *Permitir a los usuarios generar vistas de su información a lo largo del tiempo, a través de gráficas , tablas y resúmenes*

User Stories relacionados

- **US-17** Como paciente quiero ver gráficas que resuman mi información en particular para poder ver mis cambios a lo largo de la historia.
- **US-15** Como médico quiero ver gráficas que resuman la información de un paciente para poder ver sus cambios a lo largo de la historia y así apoyar la toma de decisiones y el diagnóstico.

Estos User Stories fueron desarrollados en los Sprints anteriores...

8.1. Descripción de la funcionalidad

8.1.1. Cargar análisis

8.1.2. Generación de tablas y gráficas de mediciones

8.2. Diseño Técnico

En este documento se efectúa un diseño lógico que permite, a partir de la arquitectura y la descripción del requisito en la fase de análisis, dar soporte a la implementación de la funcionalidad.

Aquí se describen los artificios utilizados a nivel técnico para resolver el problema que se planteó en la historia de usuario. Se incluyen diagramas de clases, modelos de datos, etc. Para esto se utiliza un documento compuesto de 4 partes

- Objetivo

El objetivo consiste en poder cargar mediciones de distintos tipos, de diferentes fuente y en diferentes unidades para un usuario con un perfil creado.

- Diseño de a fase de datos

Se requiere la preparación de datos para poblar las tablas que guardan los datos de los objetos de las siguientes [clases 44].

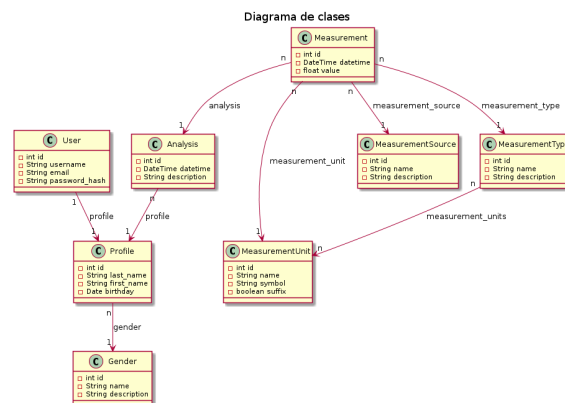


Figura 44: DC-Mediciones

- Diseño API REST

La estructura de nuestra api busca aplicar lo más fielmente posible el principio arquitectónico que define REST, Representational State Transfer, el cual

consiste en un estilo de arquitectura de software para construir web services escalables. La comunicación con la api es a través del protocolo HTTP y se busca utilizar los mismos verbos, GET, POST, PUT, entre otros. Para que la API sea REST cumple con las siguientes 6 restricciones según establece el paper que introduce REST:

1. Cliente-Servidor

La API debe estar separada del cliente y se comunican a través del protocolo de red HTTP.

2. Stateless

No se mantienen sesiones, cada solicitud y respuesta están totalmente aisladas unas de otras. Los clientes deben ser autenticados en cada solicitud. Un gris en stateless podría ser usar cookies para almacenar información que se mantiene entre solicitudes.

3. Cache

El servidor debe proveer directivas de cacheo que indiquen a intermediarios las condiciones bajo las cuales cachear información.

4. Uniform interface

Identificación de recursos, los recursos son todas las entidades en el dominio de nuestra aplicación. Cada recurso tiene un único identificador (URL), accediendo a los identificadores podemos obtener colecciones de representación de recursos.

Representación de recursos, el cliente se comunica y opera con los recursos a través de las representaciones que la api ofrece. Las representaciones pueden presentarse en distintos formatos (en nuestro caso JSON). Las representaciones se separan de los recursos.

Mensajes descriptivos, tanto solicitudes como respuestas cliente-servidor están aisladas completamente por lo que no hay información relacionada entre sucesivas solicitudes y respuestas.

Hypermedia, plantea una forma de usar la api a través de enlaces, el cliente descubre nuevos recursos a través de la información que brindan los recursos previamente descubiertos.

5. Layered system Cliente y servidor no necesariamente se comunican directamente, pueden existir intermediarios.

6. Code on demand Restricción opcional, el servidor provee código que el cliente puede utilizar.

La estructura de nuestra API se presenta de la siguiente forma:

Uno de los archivos principales es el archivo de configuración, Flask no necesita mucha configuración, solo una serie de pares clave-valor. En nuestra aplicación tenemos 4 modos en los que puede correr la aplicación. development, staging, testing and production. Lo que hacemos es definir una clase principal Config donde colocamos las variables de configuración que aplicarán a todas las configuraciones, y luego tres subclases en las cuales determinamos configuraciones que son específicas para cada uno de los tipos de configuración. La clave Secret key la usamos para encriptación de todas las fuentes. Tanto flask como las extensiones usan secret key para encriptar. Para definir el modo de configuración el valor lo tomamos de una variable del environment y como mostramos en el código, la configuración por defecto es la de producción.

```
1 class Config(object):
2     DEBUG = False
3     TESTING = False
4     CSRF_ENABLED = True
5     # encriptacion de todas las fuentes
6     SECRET_KEY = 'this-really-needs-to-be-changed'
7     SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL', '
    postgresql:///salud_dev?client_encoding=utf8')
8     Parametro que indica si el parser de argumentos debe devolver
    la totalidad de los errores encontrados en una
9     peticion a la API (True), o solo el primer error (False).
10    BUNDLE_ERRORS = True
11    # Directorio donde guardaremos el archivo
12    #UPLOAD_FOLDER = '/tmp'
13    #ALLOWED_IMG_EXTENSIONS = set(['png'])
14    UPLOADED_PHOTOS_DEST = '/tmp/imagenes'
15    MAX_CONTENT_LENGTH = 6 * 1024 * 1024
16    uploaded_photos = UploadSet('photos', IMAGES)
17    # Claves, publica y privada, de autenticacion de la aplicacion
    en Dropbox.
18    DROPBOX_APP_KEY = 'i7u47ht1t730nar'
19    DROPBOX_APP_SECRET = os.environ.get('DROPBOX_APP_SECRET', '')
20
21 class ProductionConfig(Config):
22     DEBUG = False
23
24 class StagingConfig(Config):
25     DEVELOPMENT = True
26     DEBUG = True
27
```

```
28 class DevelopmentConfig(Config):
29     DEVELOPMENT = True
30     DEBUG = True
31
32 class TestingConfig(Config):
33     TESTING = True
```

Otro de los archivos importantes es el que presenat el código de inicio de la aplicación, en este encontramos tres puntos principales, en primer lugar la creación de la instancia de flask para nuestra api

```
1 app = Flask(__name__)
2 flask_config_mode = os.getenv('FLASK_CONFIG_MODE', 'production')
3 app.config.from_object(get_config_class(flask_config_mode))
```

El punto donde definimos los identificadores para acceder a un determinado recurso, usando el método `add_resource` de la extensión de flask, **flaskRESTful** que nos permite registrar una ruta en flask. Esto permite que nuestra API presente una interfaz única.

```
api.add_resource(GenderView, '/genders/<int:id>')
api.add_resource(GenderList, '/genders')
api.add_resource(ProfileView, '/profiles/<int:id>')
api.add_resource(ProfileList, '/profiles')
api.add_resource(MeasurementView, '/measurements/<int:id>')
api.add_resource(MeasurementList, '/measurements')
api.add_resource(MeasurementSourceView, '/measurement_sources/<int:id>')
api.add_resource(MeasurementSourceList, '/measurement_sources')
api.add_resource(MeasurementTypeView, '/measurement_types/<int:id>')
api.add_resource(MeasurementTypeList, '/measurement_types')
api.add_resource(MeasurementUnitView, '/measurement_units/<int:id>')
api.add_resource(MeasurementUnitList, '/measurement_units')
api.add_resource(PermissionTypeList, '/permission_types')
```

Cabe destacar que las rutas expuestas presentan un resumen y no la totalidad de rutas que presenta acutalmente la API. Como podemos observar, el cliente tendrá acceso a la representación de respuesta del recurso `MeasurementList` solo enviando una representación correcta en una solicitud a la url `http://dominio_raiz/measurements`.

Por último, el tercer punto, se ubica en otro archivo y nos permite iniciar la API.

```

1 #!flask/bin/python
2
3 # -*- coding: utf-8 -*-
4
5 from app import app
6
7 if __name__ == '__main__':
8     app.run(host='0.0.0.0')
```

Ejecutando este script de python iniciamos la API. Los recursos de la API presentan la siguiente estructura:

```

1 class MeasurementList(Resource):
2     @marshal_with(MeasurementFields.resource_fields , envelope='
    resource ')
3     def get(self):
4         measurements = Measurement.query.all()
5         return measurements
6     @marshal_with(MeasurementFields.resource_fields , envelope='
    resource ')
7     def post(self):
8         args = parser_post.parse_args()
9         new_measurement = Measurement(args['datetime'],
10                                     args['value'],
11                                     args['analysis_id'],
12                                     args['profile_id'],
13                                     args['measurement_source_id'] ,
14                                     args['measurement_type_id'],
15                                     args['measurement_unit_id'])
16         db.session.add(new_measurement)
17         db.session.commit()
18         return new_measurement, 201
```

Como vemos el recurso define dos métodos, uno con el nombre get y otro con el nombre post, estos métodos responden a las solicitudes HTTP al identificador /measurements, que indiquen en el atributo method el operador GET y POST respectivamente. En el caso del recurso indicado, la solicitud con operando GET devolverá al cliente una colección de representaciones de medidas en la base de datos mientras que con el operando POST cargará una nueva medida. La clase del modelo que para las medidas se encuentra en el paquete models del paquete mod_profiles

```

1 class Measurement(db.Model):
```

```

2  # Atributos , columnas de la base de datos
3  id      = db.Column(db.Integer , primary_key=True)
4  datetime = db.Column(db.DateTime)
5  value    = db.Column(db.Float)
6  # Claves foraneas
7  analysis_id      = db.Column(db.Integer , db.ForeignKey('
analysis.id'))
8  profile_id       = db.Column(db.Integer , db.ForeignKey('
profile.id'))
9  measurement_source_id = db.Column(db.Integer , db.ForeignKey('
measurement_source.id'))
10 measurement_type_id  = db.Column(db.Integer , db.ForeignKey('
measurement_type.id'))
11 measurement_unit_id  = db.Column(db.Integer , db.ForeignKey('
measurement_unit.id'))
12 # Relaciones entre tablas
13 analysis            = db.relationship('Analysis' ,
14                                       backref=db.backref('
measurements' , lazy='dynamic'))
15 profile             = db.relationship('Profile' ,
16                                       backref=db.backref('
measurements' , lazy='dynamic'))
17 measurement_source   = db.relationship('MeasurementSource' ,
18                                       backref=db.backref('
measurements' , lazy='dynamic'))
19 measurement_type     = db.relationship('MeasurementType' ,
20                                       backref=db.backref('
measurements' , lazy='dynamic'))
21 measurement_unit     = db.relationship('MeasurementUnit' ,
22                                       backref=db.backref('
measurements' , lazy='dynamic'))
23 # Constructor de la clase
24 def __init__(self , datetime , value , analysis_id , profile_id ,
source_id , type_id , unit_id):
25     self.datetime      = datetime
26     self.value          = value
27     self.analysis_id    = analysis_id
28     self.profile_id     = profile_id
29     self.measurement_source_id = source_id
30     self.measurement_type_id  = type_id
31     self.measurement_unit_id  = unit_id
32 # Representacion string de la instancia
33 def __repr__(self):
34     return '<Measurement: %>' % (self.datetime)

```

La definición de la clase del modelo se hace de acuerdo a lo que establece el ORM usado por el equipo, **SQLAlchemy**.

El nombre de la tabla en la base de datos se genera automáticamente cambiando CamelCase por camel_case, el atributo column define la columna de la base de datos y las restricciones que se aplican a la misma, el atributo relationship define la relación entre las distintas tablas.

Como establecimos en la introducción de REST, la API funciona recibiendo solicitudes HTTP y respondiendo a las mismas. La API define identificadores a los cuales los clientes pueden referir, para que las solicitudes sean aceptadas. Éstas deben enviar información en el cuerpo que esté de acuerdo a las representaciones que definimos para la API, en nuestro caso, las representaciones deben ser del tipo JSON y tener los pares clave-valor que define el parser del recurso en el paquete parsers en common. Para el caso actual de medidas se define el siguiente parser.

```
1 # Parser general
2 parser = reqparse.RequestParser()
3 parser.add_argument('datetime', type=is_valid_previous_datetime,
4                     required=True)
5 parser.add_argument('value', type=float, required=True)
6 parser.add_argument('analysis_id', type=is_valid_id, required=True)
7 parser.add_argument('profile_id', type=is_valid_id, required=True)
8 parser.add_argument('measurement_source_id', type=is_valid_id)
9 parser.add_argument('measurement_type_id', type=is_valid_id,
10                    required=True)
11 parser.add_argument('measurement_unit_id', type=is_valid_id,
12                    required=True)
13
14 # Parser para recurso POST
15 parser_post = parser.copy()
16
17 # Parser para recurso PUT
18 parser_put = parser.copy()
19
20 # Parser para recurso POST con usuario autenticado
21 parser_post_auth = parser.copy().remove_argument('profile_id')
```

Como podemos ver en la tercer línea, la API recibe un dato con el nombre 'datetime', el cual es un dato de carácter obligatorio que en caso de que no esté lanzará una excepción y será validado por el método "is_valid_previous_datetime".

Este parser es usado en el método POST para sanear los datos recibidos de la solicitud y poder armar el objeto cuyos datos serán luego almacenados en la base de datos. Para el parseo de datos usamos la interfaz **Request Parsing** de flask-RESTful el cual valida los datos entrantes en la solicitud y los deserializa en objetos a nivel de la API. Así como manejamos las representaciones para recibir solicitudes también definimos representaciones para las respuestas de la API en el paquete fields del directorio common. La clase define como serializar los objetos a nivel de la app a objetos con datos de tipo primitivos que luego son formateados a JSON (Javascript object notation) para devolverse en el cuerpo de la respuesta HTTP.

```
1 class MeasurementFields:
2     # Definicion de campos de la representacion de respuesta
3     resource_fields = {
4         'id': fields.Integer,
5         'datetime': fields.DateTime(dt_format='iso8601'),
6         'value': fields.Float,
7         'analysis': fields.Nested(AnalysisFields.resource_fields),
8         'profile': fields.Nested(ProfileFields.resource_fields),
9         'measurement_source': fields.Nested(
10             MeasurementSourceFields.resource_fields),
11         'measurement_type': fields.Nested(MeasurementTypeFields.
12             resource_fields),
13         'measurement_unit': fields.Nested(MeasurementUnitFields.
14             resource_fields),
15     }
16     # Definicion de campos requeridos
17     required = ['id',
18                 'datetime',
19                 'value',
20                 'analysis',
21                 'profile',
22                 'measurement_type',
23                 'measurement_unit']
```

Para la serialización de datos usamos el decorador **marshal_with** que define flask-restful. El proceso de serialización lo inicia el decorador @marshal_with, toma los datos que pueden estar en formato de diccionario, lista, objeto, un diccionario con los campos a entregar y devuelve los datos envueltos en un JSON. Además de contener la representación en el cuerpo, la respuesta contiene un código que determina si la solicitud se procesó satisfactoriamente o no utilizando los códigos de error que define HTTP.

```

1 class MeasurementList(Resource):
2     @marshal_with(MeasurementFields.resource_fields, envelope='
    resource')
3     def post(self):
4         args = parser_post.parse_args()
5         new_measurement = Measurement(args['datetime'],
6                                       args['value'],
7                                       args['analysis_id'],
8                                       args['profile_id'],
9                                       args['measurement_source_id
10                                     ],
11                                       args['measurement_type_id'],
12                                       args['measurement_unit_id'])
13         db.session.add(new_measurement)
14         db.session.commit()
15         return new_measurement, 201

```

Como podemos ver en el código del método post del recurso para las medidas utilizamos parser para manejar las representaciones de solicitudes y marshal_with para manejar las representaciones de respuesta. Esta definición de representaciones de solicitud y de respuesta hacen que nuestra API brinde una interfaz uniforme cumpliendo con la restricción establecida por la arquitectura REST. Este desarrollo se repite para los recursos de las clases de nuestro modelo.

- Front End

8.2.1. Objetivo

El objetivo de este documento consiste en describir técnicamente el diseño de la solución planteada para poder implementar en la aplicación la carga y muestra de análisis.

8.2.2. Descripción

8.3. Técnicas de programación utilizadas

8.4. Entorno, herramientas y tecnologías utilizadas

8.5. código fuente

8.6. pruebas

9. Planificación de la capacitación

9.1. Capacitación e instrucciones de uso para los usuarios

Se grabarán videos que expliquen y ejemplifiquen el funcionamiento del sistema, tanto desde el punto de vista del paciente como del médico. Dichos videos se alojarán en *YouTube*, y se publicarán en la página oficial de *YesDoc*, dentro de la sección de ayuda y asistencia.

Se añadirán mensajes de ayuda sobre la interfaz del sitio web de *YesDoc*, lo que permitirá explicar al usuario todas y cada una de las opciones existentes en la misma. Esto se logrará mediante la integración de un asistente de iniciación interactiva que le permite, al usuario que ingresa por primera vez, conocer cuáles son las funcionalidades que brinda el sistema. Además, se hará uso de la opción “Ayuda”, que le permite al usuario poder encontrar rápidamente lo que está buscando y aprovechar al máximo cada una de las funcionalidades otorgadas.

El manual de usuario incluirá los siguientes apartados, y se desarrollará en la **Sección 11**, ubicado en la **página 86**.

- Portada.
- Título.
- Derechos de autor.
- Prefacio: contiene detalles de los documentos relacionados y la información sobre cómo navegar por la guía del usuario.
- Índice de contenido.
- Guía de funciones: explica cómo utilizar las principales funciones del sistema, es decir, sus funciones básicas.

-
- Solución de problemas: detalla los posibles errores o problemas que pueden surgir, junto con la forma de solucionarlos.
 - Preguntas frecuentes.
 - Dónde encontrar más ayuda, y datos de contacto.
 - Glosario de términos.

9.2. Capacitación e instrucciones de uso para las instituciones

A continuación se describirán todas las actividades que se llevarán a cabo para implementar el nuevo sistema en instituciones médicas. Se identificará a todas las personas responsables de cada actividad y el tiempo correspondiente a cada una:

9.2.1. Plan de capacitación

La **capacitación** es un proceso educacional de carácter estratégico aplicado de manera organizada y sistemática, mediante el cual el personal adquiere o desarrolla conocimientos y habilidades específicas relativas al trabajo, y modifica sus actitudes frente a aspectos de la organización, el puesto o el ambiente laboral.

Para permitirle a las organizaciones de salud explotar las funcionalidades del sistema al máximo se realizarán capacitaciones al personal de salud que lo utilizará para de este modo asegurar de forma correcta la carga de los datos y garantizar el uso de la totalidad de las funcionalidades brindadas.

El recurso más importante en cualquier organización lo forma el personal implicado en las actividades laborales. Por ello creemos importante y necesario la capacitación completa ya que la conducta y rendimiento de los individuos influye directamente en la calidad y optimización de los servicios que se brindan.

En tal sentido se plantea, a continuación, el Plan de Capacitación Anual que ayudará al uso fluido y completo de la aplicación.

9.2.2. Alcance

El Plan de Capacitación incluye al personal de la empresa como

- Personal Administrativo.
- Personal Recepcionistas.
- Personal de atención telefónica.

-
- Personal técnico.

9.2.3. Requisitos para realizar el curso

El alumno debe tener conocimientos de uso y manejo de computadoras, este requisito es fundamental y creemos que las instituciones hacen cumplir este requisito a su personal.

9.2.4. Fines de la capacitación

Se pretende capacitar al personal involucrado sobre el uso del sistema para que puedan realizar la carga fluida de los análisis a un paciente específico.

El fin general es el de lograr suavizar el cambio de plataforma, y de este modo lograr una adecuada adopción del sistema. Además ayuda a disminuir los errores por incomprensión de consignas básicas y ayuda a lograr un consenso de metodologías a llevar a cabo, disminuyendo la incertidumbre sobre la acción a llevar a cabo al toparse ante casos comunes ya contemplados, a través de la capacitación constante e integradora del proceso de desarrollo.

9.2.5. Objetivos generales

- Preparar al personal para la ejecución eficiente de las responsabilidades que asuman en sus puestos.
- Brindar oportunidades de desarrollo personal en los cargos actuales y para otros puestos para los que el colaborador puede ser considerado.
- Modificar actitudes para contribuir a crear un clima de trabajo satisfactorio, incrementar la motivación del trabajador y hacerlo más receptivo a la supervisión y acciones de gestión
- Colaborar con política de calidad de la empresa de capacitación de personal en forma constante.

9.2.6. Objetivos específicos

- Proporcionar orientación e información relativa a los objetivos de sistema, funcionamiento, normas y políticas.
- Proveer conocimientos y desarrollar habilidades que cubran la totalidad de requerimientos para el desempeño de puestos específicos.

-
- Actualizar y ampliar los conocimientos requeridos en áreas especializadas.
 - Contribuir a elevar y mantener un buen nivel de eficiencia individual y rendimiento colectivo.
 - Ayudar en la preparación de personal calificado, acorde con los planes, objetivos y requerimientos de la Empresa.
 - Apoyar la continuidad y desarrollo institucional.

9.2.7. Metas

Capacitar al personal para que sea capaz de utilizar de manera adecuada el sistema, explotando al máximo todas las funcionalidades ofrecidas y disminuyendo los tiempos de atención.

9.2.8. Método de capacitación y evaluación

- **Curso de 3 semanas con dictado matutino:** El primer día se detalla la carga horaria restante, pero se pretende que la primer semana sea de 3 horas y las últimas dos de 2 horas.

Para el curso es necesario asistir con un pendrive, el cual se usará para compartir la documentación. Además se recomienda traer algún elemento para tomar notas (cuaderno, notebook, etc).

La capacitación se hará de a lotes de a pequeños lotes en las primeras horas de la mañana para evitar retrasos en el trabajo.

- **Presentación inicial:** Se expondrá a todo el público la filosofía de YesDoc y los objetivos que persigue para lograr, de este modo, un compromiso conjunto de todos los participantes de sistema.
- **Presentación de “situaciones tipo”:** Instructor transmite por escrito y explica en las capacitaciones, casos referentes a circunstancias que posiblemente se presentarán, además expone un caso en tiempo real de una situación no planificada. En estas exposiciones se pide a los participantes que tomen nota del caso para comprender el funcionamiento de modo general.
- **Metodología de exposición - diálogo:** Se expondrá la guía de uso completa, para exponer todas las funcionalidades y para indicar punto por punto como utilizar el sistema.

-
- **Revisión de evaluación de desempeño:** En los controles de evaluación de desempeño, se realizará un seguimiento del personal evaluándolos en el cumplimiento de sus actividades.

Se harán observaciones individuales y el seguimiento se realizará de forma cercana para controlar y corregir situaciones.

10. Ejecución, documentación y retroalimentación de pruebas.

11. Manual de usuario del Sistema completo

12. Planificación de Implantación del Sistema

12.1. Publicidad y propaganda

Antes de comenzar a definir que técnicas utilizaremos, es necesario diferenciar a estos dos métodos. Por un lado, la publicidad es una herramienta que se utiliza con objetivos comerciales; en nuestro caso conseguir una venta. La propaganda por otro lado difiere de la publicidad, su objetivo es modificar ideologías, costumbres y la visión de la realidad, objetivo fundamental de nuestro sistema.

Invertiremos en realizar publicidad en aquellos sitios web de salud que lo permitan, para conquistar a aquellos usuarios interesados por su cuidado personal.

Utilizaremos lugares relacionados a la salud para dar a conocer nuestro producto, estos lugares pueden ser hospitales, farmacias, centros de salud, etc. Aprovecharemos la íntima relación que tiene nuestro sistema con los lugares antes citados para establecer convenios que permitan el beneficio mutuo a partir de la prestación del servicio de nuestro sistema. Por ejemplo actualmente existen farmacias que brindan, a sus clientes, la posibilidad de acceder a una cuenta para ver los productos que se ha comprado a lo largo de su historia como cliente, dándoles puntos por cada compra que luego podrán canjear, sería interesante ofrecerle a esas farmacias nuestro sistema para que ellos le brinden a sus clientes mas beneficios y así nuestro sistema se beneficiaría con la popularidad del mismo.

También será necesario realizar una buena campaña de marketing utilizando propaganda en Youtube para atrapar a aquellos usuarios que no se encuentran familiarizados con la tecnología, pero que si se interesarán por los beneficios que ofrece nuestro sistema.

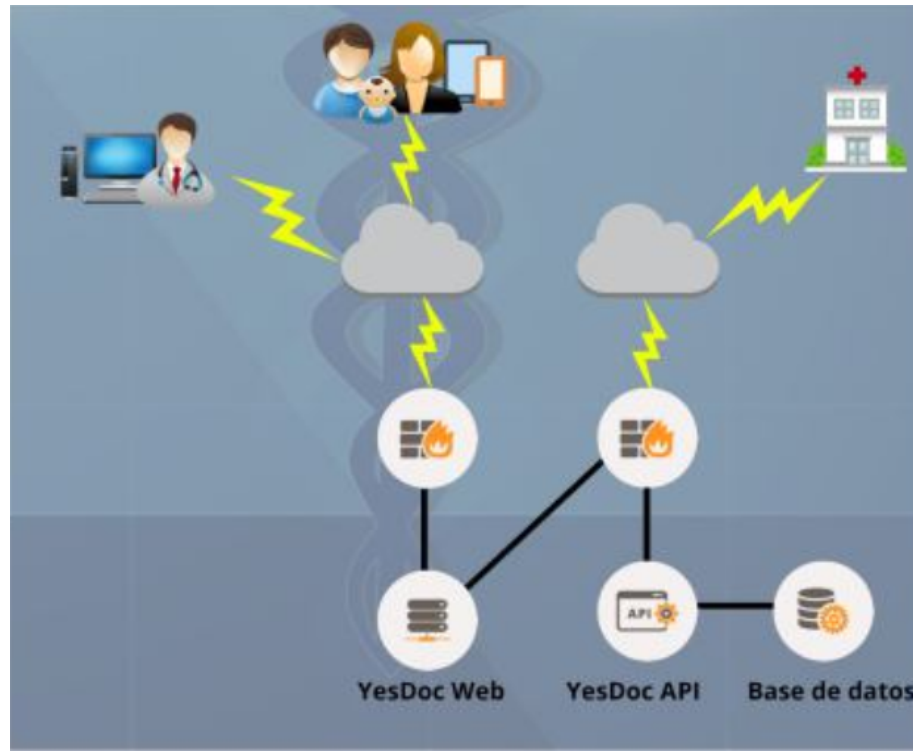


Figura 45: Arquitectura del sistema

12.2. Configuración y diseño del sistema

Para facilitar el acceso al usuario se implementará el sistema en una plataforma web utilizando un servidor diferente del servidor que gestionará las conexiones a la API y a la base de dato.

Esta arquitectura del sistema se puede observar en la figura 45 en ella se puede ver como los dispositivos móviles que acceden a través de la aplicación tienen acceso directo a la capa de servicio a diferencia de aquellos que acceden al sistema a través del navegador que tiene acceso a la capa de servicio a través del servidor Web.

Esta topología será mejor a la hora de escalar para soportar y brindar servicio a los usuarios que lo requieran, sin incurrir en tiempos prolongados de espera ni modificaciones en la arquitectura y el diseño del sistema.

12.3. Implantación en instituciones

Si bien se brinda una interfaz web que facilita el uso a cualquier usuario, contemplamos que muchas de las instituciones necesitarán gestionar el sistema en un servidor propio. Por este motivo se ha desarrollado un plan de implantación que permitirá llevar a cabo de manera adecuada la correcta migración del sistema.

12.3.1. Preparación de datos y archivos

Será necesario determinar cuál será la base de conocimiento que se cargará en el sistema, estas deberán cumplir con los estándares medicinales, algunos ejemplos son:

- Especialidades
- Tipos de análisis
- Tipo de mediciones
- Formatos de análisis
- Sobre el paciente: sexo, enfermedades, estados fisiológicos, etc.
- Información farmacológica: alergias, medicamentos.
- Productos comerciales: productos farmacéuticos que se comercializan en un área o región, nombre comercial, presentación, dosificación, precio, cobertura, tipo de dispensación, conservación, origen, laboratorio que lo produce. Las fuentes de donde se obtiene la información son la industria farmacéutica, financiadores de salud y farmacólogos clínicos. Estos datos son generalmente mantenidos por empresas abocadas a tal fin y en algunos casos por organismos oficiales encargadas de informar periódicamente a sus suscriptores sobre las altas, bajas o modificaciones de productos medicinales.
- Principios activos o monodrogas: nombre genérico, sinónimos, clasificación farmacológica y/o terapéutica, farmacodinamia y farmacocinética, preparación, formas de administración, rango de dosis recomendada, dosificación en pacientes pediátricos, dosificación en ancianos, dosificación en insuficiencia renal, dosificación en insuficiencia hepática severa, dosificación en cirrosis, embarazo y lactancia, sobredosis, precauciones, indicaciones, contraindicaciones, reacciones adversas, antagonismos y antidotismos, interacciones, efectos sobre exámenes de diagnóstico e información para los pacientes

-
- Terminología médica: Unified Medical Language System (UMLS); SNOMED CT terminología clínica integral, multilingüe y codificada de mayor amplitud, precisión e importancia desarrollada en el mundo; variantes léxicas, opciones controladas, reglas terminológicas, CIAP-2 Clasificación Internacional de Atención Primaria.
 - Prestadoras, aseguradoras, lugares físicos y prestaciones.

Además se deberá modelar el sistema para que permita futuras conexiones con otros similares ya existentes, como pueden ser de laboratorios o la historia clínica del hospital al que incurre el paciente que quiere utilizar el sistema. Es por este motivo que se ofrece una API que permita la conexión de sistemas ajenos al nuestro.

12.4. Despliegue del servidor de front-end

Para aquellas personas que deseen utilizar el sistema de manera particular, añadiéndole funcionalidades y cambiando el diseño o las interfaces para que el tiempo de adaptación sea el mínimo, se presenta un documento detallado indicando la forma de desplegar el sistema localmente para que puedan evaluarlo y modificarlo como mas lo deseen.

Dichos pasos además de detallarse a continuación se encuentran en el archivo **README.md** del proyecto que se encuentra hosteado en *github*.

1. Instalar NodeJS

```
curl --silent --location https://deb.nodesource.com/  
  setup_0.12 | sudo bash -  
sudo apt-get install --yes nodejs
```

2. Instalar las dependencias

```
# Instalar el administrador de paquetes  
sudo apt-get install npm  
sudo npm install -g npm  
  
# Dependencias de NodeJS  
cd web/  
sudo npm install  
sudo npm install -g yo bower grunt-cli
```

```
# Dependencias de YesDoc  
cd web/  
bower install
```

3. Iniciar el servidor

```
# Para iniciar el servidor  
grunt serve
```

A. Anexo

A.1. Código

código