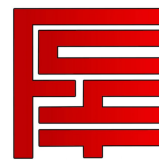




UNIVERSIDAD MAYOR DE SAN SIMÓN  
FACULTAD DE CIENCIAS Y TECNOLOGÍA  
INGENIERÍA INFORMÁTICA



# SISTEMA DE CONTROL Y SEGUIMIENTO DE VEHÍCULOS

PROYECTO DE GRADO, PRESENTADO PARA OPTAR AL DIPLOMA ACADÉMICO DE  
LICENCIATURA EN INFORMÁTICA

PRESENTADO POR: PÉREZ SOTO, JOSÉ BENJAMÍN

TUTOR: LIC. LAIME ZAPATA, VALENTIN

COCHABAMBA, BOLIVIA  
DICIEMBRE 2014

# FICHA RESUMEN

Aqui va tu resumen.

# Índice general

<b>1</b>	<b>INTRODUCCIÓN</b>	<b>1</b>
1.1	Antecedentes . . . . .	1
1.2	Descripción del problema . . . . .	1
1.3	Objetivo General . . . . .	1
1.4	Objetivos Específicos . . . . .	1
1.5	Alcances y Limitaciones . . . . .	1
1.6	Justificación . . . . .	1
<b>2</b>	<b>MARCO REFERENCIAL</b>	<b>2</b>
<b>3</b>	<b>METODOLOGÍA DE DESARROLLO PARA EL SISTEMA</b>	<b>3</b>
3.1	Test-Driven Development - Donde todo empezó . . . . .	4
3.2	Behaviour-Driven Development: El siguiente paso . . . . .	6
3.3	El ciclo de BDD . . . . .	12
3.4	Herramientas para BDD . . . . .	13
<b>4</b>	<b>DESARROLLO INICIAL DEL SISTEMA</b>	<b>15</b>
4.1	Escenarios del sistema . . . . .	15
4.2	Código generado para los test . . . . .	24
4.3	Modelo del sistema . . . . .	28
<b>5</b>	<b>IMPLEMENTACIÓN DEL SISTEMA WEB</b>	<b>34</b>
5.1	Estructura del Proyecto . . . . .	37
5.2	Configuración del proyecto . . . . .	39
5.3	Aplicaciones extras para el sistema . . . . .	41
5.4	Formularios para los registros . . . . .	42

6	PRUEBAS DEL SISTEMA	<b>48</b>
6.1	django-behave . . . . .	49
6.2	splinter . . . . .	50
6.3	Configuración de los escenarios . . . . .	50
6.4	Escenario Account . . . . .	52
7	PUESTA EN PRODUCCIÓN	<b>54</b>
7.1	Servidor Debian Linux . . . . .	54
7.2	Servidor Web Nginx . . . . .	54
7.3	Base de datos MySQL . . . . .	55
7.4	Instalación y Configuración del Sistema . . . . .	55
7.5	Configuración en el servidor . . . . .	57
8	CONCLUSIONES Y RECOMENDACIONES	<b>62</b>
8.1	Conclusiones . . . . .	62
8.2	Recomendaciones . . . . .	63
A	INSTALACIÓN Y CONFIGURACIÓN DE HERRAMIENTAS	<b>64</b>
A.1	Entorno Virtual . . . . .	65
A.2	Behave . . . . .	65
A.3	Django-Behave . . . . .	65
A.4	Splinter . . . . .	66
B	TESTS DE EJECUCIÓN	<b>67</b>
B.1	Steps - Antes del desarrollo del sistema . . . . .	68
B.2	Steps - Con el sistema concluido . . . . .	72
C	MANUAL DE USUARIO	<b>75</b>
	BIBLIOGRAFÍA	<b>76</b>

# Índice de figuras

3.1.1 TDD . . . . .	4
3.1.2 TDD: red, green, refactor . . . . .	5
3.3.1 BDD . . . . .	12
3.3.2 BDD . . . . .	13
4.1.1 Diagrama de secuencia - Registro de vehículos . . . . .	18
4.1.2 Diagrama de secuencia - Registro de personas . . . . .	19
4.1.3 Diagrama de secuencia - Programar mantenimiento . . . . .	20
4.1.4 Diagrama de secuencia - Solicitud de reportes . . . . .	22
4.1.5 Diagrama de secuencia - Administrar notificaciones . . . . .	23
4.2.1 Behave - primera ejecución . . . . .	24
4.2.2 Directorio features . . . . .	25
4.2.3 Ejecución con Behave con error . . . . .	26
4.2.4 Ejecución con Behave sin error . . . . .	27
4.3.1 ACL - Roles del sistema . . . . .	28
4.3.2 Modelo ER - Registro de vehículos . . . . .	30
4.3.3 Modelo ER - Registro de visitas . . . . .	30
4.3.4 Modelo ER - Programar mantenimiento . . . . .	31
4.3.5 Modelo ER - Registro de salida a taller . . . . .	32
4.3.6 Modelo ER - Notificaciones . . . . .	33
5.0.1 Model Template View - MTV . . . . .	35
5.0.2 El flujo de Django . . . . .	36
5.1.1 Estructura de una aplicación de un proyecto Django . . . . .	38
5.1.2 Django Urls . . . . .	39
5.3.1 Página de inicio del Sistema Web . . . . .	42
5.4.1 Formulario de Registro de Vehículos . . . . .	44

5.4.2 Formulario de Registro de Personas . . . . .	46
5.4.3 Formulario para Reportes . . . . .	47

*Dedicado a mis queridos padres Roberto y Lita por el apoyo que siempre me dieron y a mis hermanos y familiares por motivarme todos estos años.*

# Agradecimientos

GRACIAS a mi tutor Lic. Valentin Laine por compartir sus conocimientos y guiarme en este proyecto y a todos los docentes de la carrera, pero principalmente al Dr. Pablo Azero, Lic. Leticia Blanco, Lic. Vladimir Costas, Lic. Rolando Jaldin y a la Lic. Patricia Romero por brindarme sus conocimientos que me guiaron a lo largo de mis estudios. Tambien agradezco a la Sociedad Científica de Estudiantes de Sistemas e Informática – SCESI que me brindo su apoyo y conocimientos.



# 1

## Introducción

- 1.1. ANTECEDENTES
- 1.2. DESCRIPCIÓN DEL PROBLEMA
- 1.3. OBJETIVO GENERAL
- 1.4. OBJETIVOS ESPECÍFICOS
- 1.5. ALCANCES Y LIMITACIONES
  - 1.5.1. ALCANCES
- 1.6. JUSTIFICACIÓN

# 2

Marco Referencial

# 3

## Metodología de Desarrollo para el Sistema

DESARROLLO GUIADO POR COMPORTAMIENTO - BDD<sup>1</sup>, es la metodología que se eligió para el desarrollo del Sistema.

Es una técnica de desarrollo ágil de software que fomenta la colaboración entre desarrolladores, testers y clientes. Fue nombrado originalmente en el año 2003 por Dan North<sup>2</sup> en respuesta al Desarrollo Guiado por Pruebas - TDD<sup>3</sup>. Podemos considerarlo una evolución o un mejor entendimiento y una forma mejor de explicar los procesos TDD. Se observó que los *desarrolladores* estaban teniendo momentos difíciles en relación con TDD que es más una herramienta para diseño que para hacer pruebas.

---

<sup>1</sup>BDD, por sus siglas en inglés, Behaviour-Driven Development

<sup>2</sup>Dan North, <http://dannorth.net/> consultado en fecha 10/06/2014

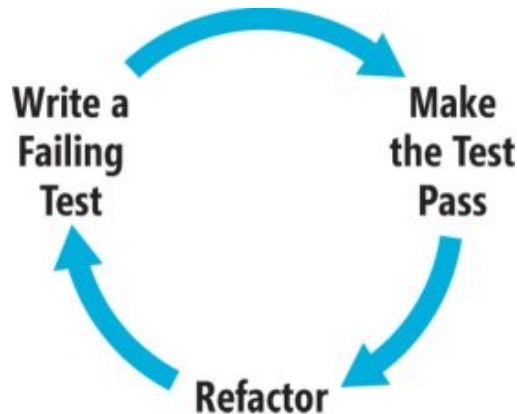
<sup>3</sup>TDD, por sus siglas en inglés, Test-Driven Development

### 3.1. TEST-DRIVEN DEVELOPMENT - DONDE TODO EMPEZÓ

TDD es una practica de desarrollo que involucra escribir primero los “*Test*” antes que el código a ser *testeado* o probado. Se empieza escribiendo test muy pequeños para el código que aun no existe, luego de esto si corremos los test notaremos que los test fallan naturalmente, ahora hay que escribir el código necesario para que el test pase.

Una vez que pase los tests, observe el diseño resultante, y refactorizar cualquier duplicación o código innecesario que se encuentre. Es natural que en este momento el diseño es demasiado simple para manejar todas las responsabilidades que tendrá.

En lugar de añadir mas código, documente la siguiente responsabilidad en el formulario de los siguientes tests. Ejecútelo, y vea los fallos, escriba el código necesario para que pase el test, revise el diseño, remueva el código innecesario o duplicado. Ahora añada el siguiente test, vea los fallos, consiga que pase los tests, refactorizar, fallo, paso, refactorizar, fallo, paso, refactorizar, ...etc.

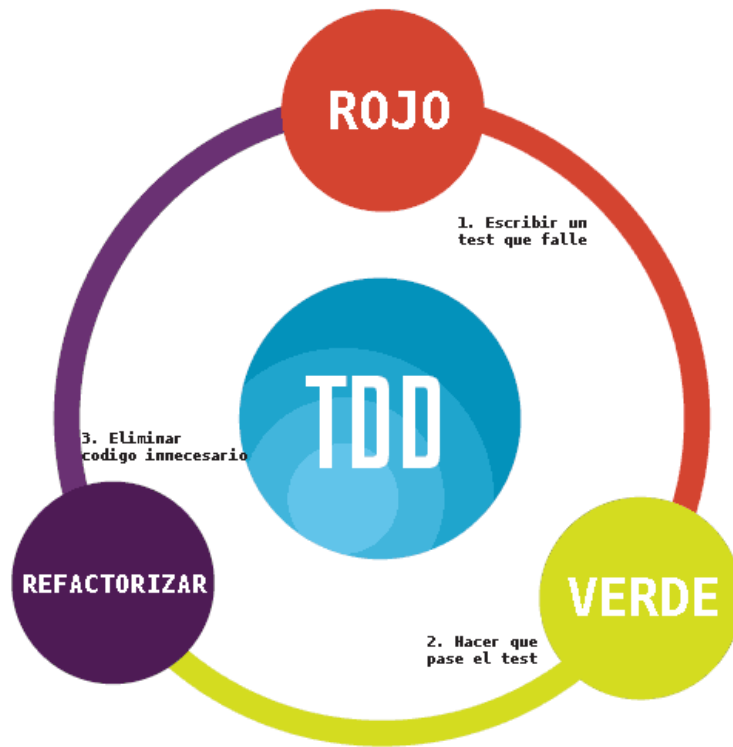


**Figura 3.1.1:** Ciclo del Desarrollo Guiado por Pruebas.

En muchos *Sistemas de Tests*, cuando un test falla, nosotros vemos los resultados pintados en *rojo*. Después cuando esto pasa, los resultados son pintados en *verde*. Debido a esto, a menudo nos referimos a este ciclo como *rojo/verde/refactorizar*.

#### 3.1.1. DISEÑO EMERGENTE

Como el código base incrementa en tamaño, encontraron que se da mucha atención en el paso de refactorización. El diseño esta en constante evolución y bajo



**Figura 3.1.2:** Desarrollo Guiado por Pruebas es rojo, verde, refactor.

revisión constante, aunque no esta predeterminado. Esto es un *diseño emergente* a un nivel granular y es uno de los subproductos mas significativos de TDD.

Más que pensar en TDD como una práctica de pruebas, lo vemos como una técnica utilizada para entregar código de alta calidad a los testers, los que son responsables de las prácticas de pruebas formales.

Y aquí es donde los *Test* en TDD llega a tener problemas. Específicamente, esta es la idea de *unit testing*<sup>4</sup> y esto a menudo conduce a nuevos TDDers para verificar y asegurarse de que un objeto sea el objeto esperado, como ser un método *register()* almacene una colección de *Registers* y sea específicamente un *Array*.

Este tipo de detalle en un test crea una dependencia del test en la estructura interna del objeto que esta siendo probado. Esta dependencia significa, que si hay otro requerimiento que nos hace cambiar el objeto de Array a Hash el test fallará, a pesar de que el comportamiento no cambie. Esta fragilidad puede hacer conjuntos de pruebas mucho mas caras de mantener, y esta es la razón por la que muchos grupos de pruebas son ignorados o incluso descartados.

---

<sup>4</sup>unit testing, son pruebas unitarias

En resumen, si estos test internos que se hace a un objeto son contraproducente a largo plazo, ¿en que debemos centrarnos al escribir estos test por primera vez?

*TDD es una práctica para entregar código de calidad con un buen diseño que una práctica para realizar **pruebas***

### 3.2. BEHAVIOUR-DRIVEN DEVELOPMENT: EL SIGUIENTE PASO

El problema con los tests que se hacen a la estructura de los objetos es que estamos probando si *es* un objeto en lugar de lo que *hace* el objeto. Lo que *hace* un objeto es mucho más importante.

Lo mismo ocurre a nivel de aplicación. Los Stakeholders<sup>5</sup> no le dan importancia a que si los datos son persistentes o no en una base de datos relacional compatible con ANSI. A ellos les importa de que “*estén en la base de datos*”, por lo general a ellos les interesa de que este almacenado en *algún lugar* de donde ellos puedan recuperarlo.

#### 3.2.1. TODO ES COMPORTAMIENTO

BDD se enfoca completamente en el *comportamiento* en lugar de la estructura, y lo hace en todos los niveles de desarrollo. Ya sea que estemos hablando de un objeto que calcule la distancia entre dos ciudades, u otro objeto que delegue la búsqueda de servicios, o una interfaz de usuario que se provee para hacer feedback cuando ocurre una salida invalida, *todo es comportamiento*.

Una vez que reconocemos esto, cambiamos la forma en que pensamos acerca de la expulsión de código. Empezamos a pensar mas en la interacción entre las personas y el sistema, o entre objetos, de los que sabemos acerca de su estructura.

#### 3.2.2. CONSEGUIR LAS PALABRAS CORRECTAS

Creemos que la mayoría de los problemas que enfrentan los equipos de desarrollo de software son los problemas de comunicación. BDD tiene como objetivo ayudar a la comunicación mediante la simplificación del lenguaje que usamos para describir los escenarios en los que se utilizara el software: *Given* obtener datos de algún contexto, *When* cuando ocurre algún evento, *Then* entonces espero algún resultado.

---

<sup>5</sup>Stakeholders, los interesados del sistema

*Given, When, Then*, la triada BDD, son simples palabras que utilizaremos ya sea que estemos hablando del comportamiento de la aplicación o de un objeto. Estas palabras son fáciles de entender por los analistas de negocios, testers, y desarrolladores por igual. Estas palabras están incrustadas en el DSL<sup>6</sup> Gherkin.

### 3.2.3. GHERKIN

Gherkin es un DSL legible para gente no técnica, que permite definir el comportamiento del software sin detallar como está implementado, además de que nos permite documentar las funcionalidades a la vez que escribimos casos de prueba automáticos.

Otras ventajas que nos proporciona usar Gherkin:

- Fácil de entender
- Fácil de leer
- Fácil de parsear
- Fácil de discutir

Gherkin es un lenguaje que usa el indentado para definir la estructura, de manera que los saltos de línea dividen las diferentes declaraciones, la mayoría de las líneas empiezan con palabras clave. El parser divide el texto en *Features*, *Scenarios* y *Steps*, cuando pasas los casos de prueba, el parser busca un Step con ese nombre. Los Steps son los análogos de los métodos en Java o las funciones en Javascript.

La gramática de Gherkin consiste en pocas palabras claves que debe usarse cuando escribimos un archivo *.feature*:

- Feature
- Background
- Scenario
- Scenario outline
- Examples
- Given, When, Then, And, But (Para definir pasos)

---

<sup>6</sup>Domain Specific Language, Lenguaje de dominio específico

- | (Para definir tablas)
- """ (Para definir una cadena de varias lineas)
- # (Para usar comentarios)

Se puede escribir lo que uno quiere después de una palabra clave. Las palabras claves *Given*, *When*, *Then*, *And* y *But* indican pasos en un escenario, que usaremos para construir un DSL para un proyecto.

Cada *Feature* o *característica* se define en un archivo *.feature*, una característica normalmente consiste en una serie de *escenarios*, el texto entre *Feature* y *Scenario* permite definir libremente el contexto ya que este texto no será usado para ninguna funcionalidad en el código generado, es meramente descriptivo.

```
Feature: [nombre de la característica en general a probar]
  As [un actor/rol]
    In order to [algún beneficio]
      I want [una característica]

Scenario: ...
```

Nos ayuda a entender *como* que tipo de usuario es el que va a usar esta característica o funcionalidad, *para* que nos sirve esta funcionalidad y que *quiero* obtener.

#### 3.2.4. SCENARIO - ESQUEMA DEL ESCENARIO

Los escenarios son ejemplos concretos de como queremos que se comporte el software. Esta manera de hacerlo es mas explicita que algunas formas tradicionales para describir requerimientos y nos ayudan a plantear y responder preguntas que de lo contrario no podríamos hacerlas. Los escenarios nos permiten responder a preguntas, describiendo exactamente lo que debe suceder y en que circunstancias debe hacerse.



```
Feature: [nombre de la característica en general a probar]
  As [un actor/rol]
  In order to [algún beneficio]
  I want [una característica]

Scenario: [un ejemplo concreto de lo que se quiere probar]
```

Cuando se empieza a escribir una nueva *característica* o *feature*, por lo general es más fácil comenzar con un escenario que describe el “camino feliz” y más común. Una vez que haya terminado con eso, se puede agregar más escenarios que describan diferentes casos, por ejemplo:

```
Feature: [nombre de la característica en general a probar]
  As [un actor/rol]
  In order to [algún beneficio]
  I want [una característica]

Scenario: [un ejemplo concreto de lo que se quiere probar]

Scenario: [...]

Scenario: [...]

...
```

Ahora, para que cada *Scenario* tenga un comportamiento correcto necesita seguir varios pasos.

### 3.2.5. STEPS - PASOS

Cada escenario usa un numero arbitrario de *pasos* o *steps* para describir todo lo que pasa dentro de un escenario. Un paso es generalmente una simple linea de texto que empieza con una de las palabras claves: *Given*, *When*, *Then*, *And* y *But*

```
Feature: [nombre de la característica en general a probar]
  As [un actor/rol]
  In order to [algún beneficio]
  I want [una característica]

Scenario: [un ejemplo concreto de lo que se quiere probar]
  Given [una condición previa]
  When [una acción]
  Then [un resultado esperado]

Scenario: [...]
  Given [...]
  When [...]
  Then [...]

Scenario: [...]

Scenario: [...]
```

### 3.2.6. INTERNACIONALIZACIÓN

Las palabras reservadas de Gherkin fueron traducidas en varios idiomas como el español, eso significa que nosotros podemos escribir los *Features* en nuestro propio idioma o en el idioma del stakeholder. En los archivos *.feature* se debe escribir la siguiente línea en la cabecera del archivo.

```
# language: es
Característica: [...]
```

Para otros idiomas como el francés o portugués sería de la siguiente forma:

```
# language: fr
Fonctionnalité: [...]
```

```
# language: pt
Funcionalidade: [...]
```

Gherkin usa por defecto el idioma ingles, asi que no es necesario poner la cabecera si es que se va a escribir en ingles.

Con la cabecera ya puesta en español, podemos escribir los features de la siguiente manera:

```
# language: es

Característica: [nombre de la característica en general a probar]
  Como [un actor/rol]
  Para [algún beneficio]
  Quiero [una característica]

Esquema del escenario: [un ejemplo concreto a probar]
  Dado [una condición previa]
  Cuando [una acción]
  Entonces [un resultado esperado]

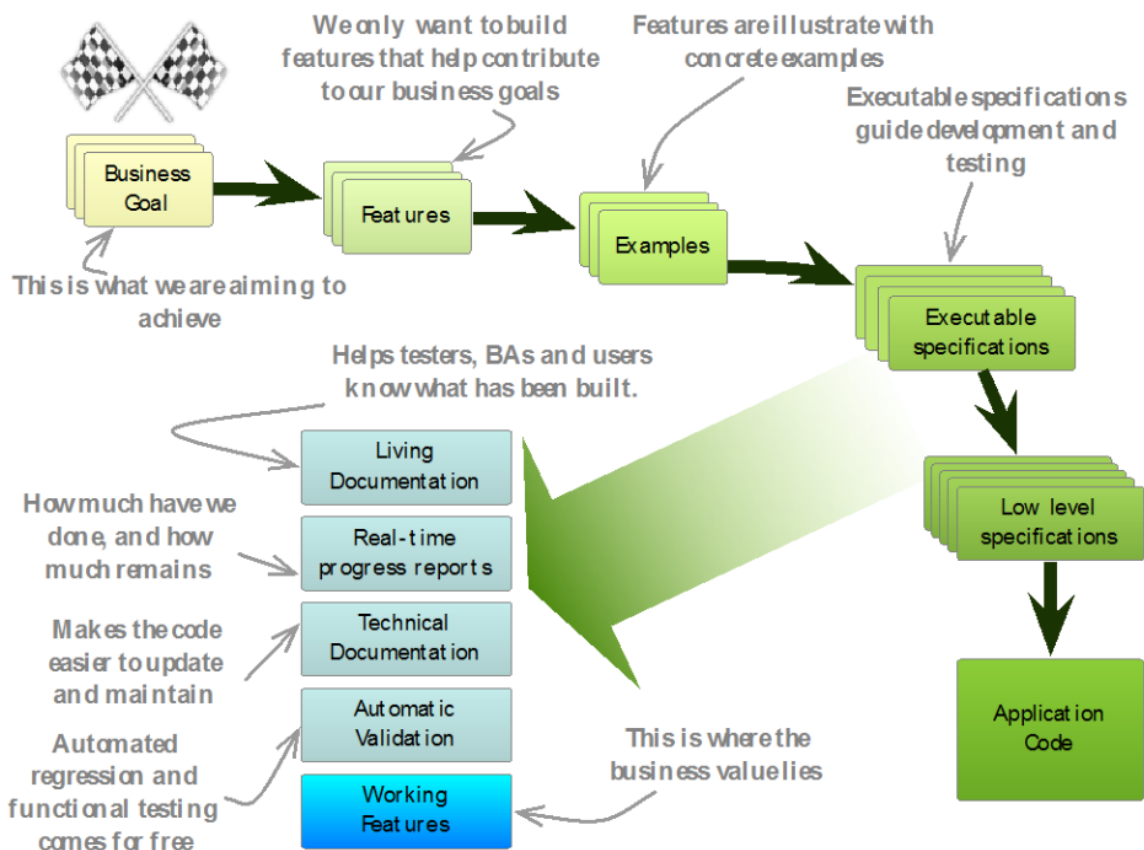
Esquema del escenario: [...]
...
Esquema del escenario: [...]
...
```

### 3.3. EL CICLO DE BDD

BDD se centra en la obtención de una comprensión clara del comportamiento del software deseado a través de la discusión con las partes interesadas.

Es una forma de crear comportamientos o funcionalidades que se puedan probar y sean automatizados que agreguen valor al momento de mostrar al cliente antes de que exista el código fuente. Evitan errores que pudieran existir basados en las funcionalidades o comportamientos del sistema y generan un conjunto de tests basados en esas funcionalidades.

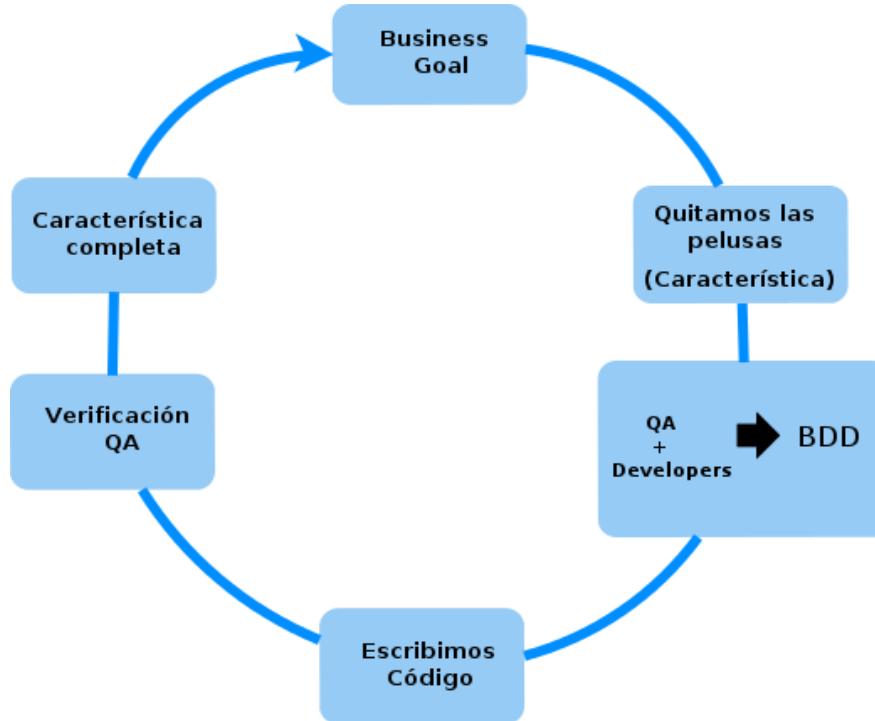
Según [Smart](#) define de una forma clara el funcionamiento básico de BDD:



**Figura 3.3.1:** Funcionamiento Básico de BDD, Fuente: John Ferguson Smart - BDD in Action

Hacer que pase un test es *muy diferente* a una funcionalidad conseguida.

Con lo expuesto anteriormente llegamos a un ciclo ágil de desarrollo:



**Figura 3.3.2:** El ciclo de vida ágil de BDD

Las historias o los *Business Goal* son los que dirigen nuestro desarrollo, estas historias son parte de nuestro desarrollo, son las funcionalidades que queremos conseguir.

Una vez que lo hemos implementado podemos comprobar de acuerdo con las especificaciones que se han escrito al principio.

En el siguiente capítulo veremos a detalle el funcionamiento de BDD con escenarios reales.

### 3.4. HERRAMIENTAS PARA BDD

Gherkin tiene varias opciones que interpretan su código escritos en los archivos *.feature* tales como: Cucumber, Lettuce, Freshen, Behave, etc.

Para este proyecto usamos Behave<sup>7</sup>.

---

<sup>7</sup>Behave, desarrollado por Benno Rice and Engel

### 3.4.1. BEHAVE

Behave es escribir BDD al estilo de Python, es decir behave utiliza los tests escritos en un estilo de lenguaje natural como es Gherkin para obtener el código inicial para hacer las pruebas que es código python y estas mismas son ejecutadas para sus pruebas por behave.

En el siguiente capítulo se explicará con más detalle como se escriben las funcionalidades y escenarios de prueba y como se los ejecuta con behave, tomando en cuenta escenarios reales.

# 4

## Desarrollo inicial del Sistema

EN Sistema Web que se desarrolló, se sistematizaron las siguientes tareas:

- El registro de los vehículos con las escaleras y el conductor.
- El registro de personas que ingresan a la empresa.
- El registro de mantenimiento a los vehículos.
- La emisión de reportes por vehículo, por parqueo, conductor y escaleras.
- Las notificaciones/alertas para el control del supervisor.

En este capítulo veremos como empleamos la metodología BDD en la fase inicial de desarrollo del Sistema Web.

### 4.1. ESCENARIOS DEL SISTEMA

Cada tarea a sistematizar llega a ser un comportamiento o funcionalidad para un determinado usuario o rol.

#### 4.1.1. REGISTRO DE VEHÍCULOS

En este primer escenario nos interesa saber el día, la hora, el parqueo, el conductor y las escaleras con las que sale y retorna.

Este primer *Feature* o *Característica* sera el *Registro de vehículos* y lo guardaremos en el archivo **registro.feature** con el siguiente contenido:

```
Característica: Registro de vehículos
  Como un guardia
  Quiero registrar las salidas y retornos de los vehículos
  Para registrar los datos

Esquema del escenario: Registrar vehículo de la empresa
  Dado que el vehículo con numero interno <interno> esta de <estado>
  Y sale/retorna del parqueo <parqueo>
  Y con el conductor de con ítem <num_item>
  Y tiene el kilometraje <km>
  Y con las escaleras <escaleras>
  Y en fecha <fecha>
  Y en horas <hora>
  Cuando}} registre al vehículo <interno> con su estado de <estado>
  Y los datos del vehículo <interno> sean validos
  Entonces guardo el registro del vehículo <interno>
  Pero que pasa si el conductor del vehículo <interno> de retorno es diferente
  Y las escaleras del vehículo <interno> de retorno no son las mismas
  Y el parqueo de retorno del vehículo <interno> no es el mismo
  Y si el registro del vehículo <interno> tiene observaciones debe notificarse al supervisor
```

Toda esta sintaxis ya vimos en el capítulo 3, pero ahora se presenta algo nuevo, hay palabras que están dentro de < y >, esto también es parte de la sintaxis de Gherkin, que es para el uso de tablas para los ejemplos.

Ejemplos: Datos de registro

	parqueo	interno	num_item	km	escaleras	fecha	hora	estado
	muyurina	123	543	10100	23 43 12	18/02/2014	08:12	salida
	km 0	321	432	20100	45 14 54	18/02/2014	08:34	salida
	muyurina	I-02	564	15210	78 46 11	18/02/2014	08:43	salida
	muyurina	I-04	101	10010	99 98 90	18/02/2014	08:50	salida
	muyurina	I-01	110	12110	90 78 11	18/02/2014	08:58	salida
	muyurina	321	432	20159	45 14	18/02/2014	10:05	entrada
	muyurina	321	432	20159	45 14 54	18/02/2014	10:50	salida
	taller	321	432	20249	45 14 54	18/02/2014	15:50	entrada
	taller	321	432	20249	45 14 54	18/02/2014	18:10	salida
	muyurina	123	505	10189	23 43 12	18/02/2014	18:52	entrada
	muyurina	I-01	110	12184	90 78 11	18/02/2014	19:38	entrada
	sucre	I-02	564	15256	78 46 11	18/02/2014	20:00	entrada
	muyurina	321	432	20293	45 14 54	18/02/2014	20:35	entrada
	muyurina	I-01	110	12184	90 78 11	19/02/2014	08:18	salida
	muyurina	I-04	1305	10000	99 98	19/02/2014	13:05	entrada
	muyurina	I-01	120	12110	90 78 11	20/02/2014	08:58	salida
	muyurina	I-01	110	12987	90 78 11	25/02/2014	08:14	entrada



En este comportamiento manejamos el rol de *Guardia* que llegaría a ser el Vigilante Industrial, el se encarga de registrar a los vehículos de la empresa al sistema. Este primer escenario es un caso ideal o como se dijo en el capítulo 3 es el *camino feliz*, los siguientes escenarios muestran otras variables que manejan los guardias al momento de registrar los vehículos ya sean de la empresa o no.

#### Esquema del escenario: Registrar vehículo alquilado

Dado que tiene la placa de control <placa>  
 Y esta de <estado> del parqueo <parqueo>  
 Y con el conductor con ítem <num\_item>  
 Y con las escaleras <escaleras>  
 Y en fecha <fecha>  
 Y en horas <hora>  
 Cuando registre al vehículo con placa de control <placa>  
 Y se validen los datos del vehículo alquilado  
 Entonces registro el vehículo con placa de control <placa>

#### Ejemplos: Datos de salida

parqueo	placa	num_item	escaleras	fecha	hora	estado
muyurina	2341 GAB	345	12 32 43	18/02/2014	08:00	salida
km 0	212 BEN	456	44 55 76	18/02/2014	08:00	salida
muyurina	212 BEN	456	44 55 76	18/02/2014	11:10	entrada
muyurina	212 BEN	456	44 55 76	18/02/2014	11:40	salida
muyurina	2341 GAB	345	12 32 43	18/02/2014	18:05	entrada
km 0	212 BEN	456	44 55 76	18/02/2014	19:10	entrada

#### Esquema del escenario: Registrar vehículos que se quedan en parqueo

Dado que que tiene el numero interno <num\_interno>  
 Y esta en el parqueo <parqueo>  
 Y en fecha <fecha>  
 Cuando registre el vehículo <num\_interno>  
 Entonces el vehículo se quedo en parqueo

#### Ejemplos: Datos de vehículos que no salieron

parqueo	num_interno	fecha
muyurina	I-02	8/05/2014

#### Esquema del escenario: Ingreso de vehículo externo

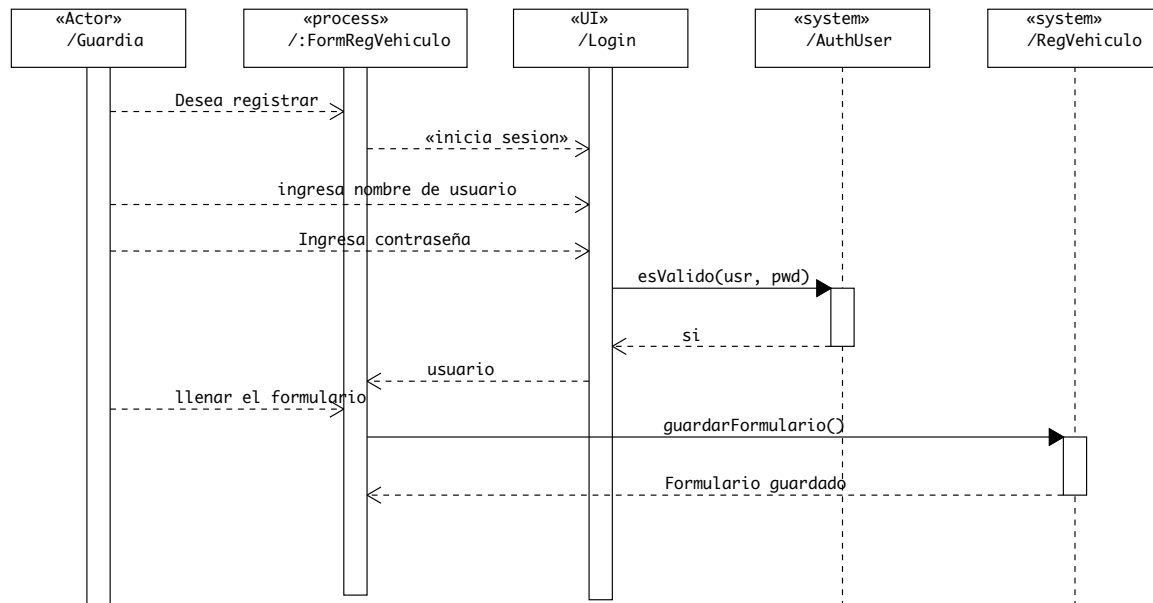
Dado que ingresa una persona con documento <documento> a las oficinas <oficina>  
 Y con numero de documento <num\_doc>  
 Y con el siguiente nombre <nombre>  
 Y proviene de la ciudad <ciudad>  
 Y con el siguiente motivo <motivo>  
 Y con placa de vehículo <placa>  
 Cuando registre a la persona <nombre> y el vehículo <placa>  
 Y los datos sean validos  
 Entonces registro a la persona <nombre> con el vehículo <placa>

#### Ejemplos:

oficina	documento	num_doc	nombre	ciudad	placa	motivo
muyurina	pasaporte	123456	juan	br	123 RRR	descarga de material
central	CI	654321	carl	lpz	321 GHF	descarga de material

Estos escenarios nos muestran las diferentes tareas que tienen que cumplir los guardias, como registrar vehículos que no son de la empresa, vehículos alquilados y vehículos de la empresa que no registran ninguna salida.

Para esta primera característica tendríamos un diagrama de secuencias de la siguiente manera:



**Figura 4.1.1:** Diagrama de secuencia para el Registro de vehículos

#### 4.1.2. REGISTRO DE INGRESO DE PERSONAS

Los guardias también tienen que estar atentos a toda *persona ajena* a la empresa que quiere ingresar a las oficinas. Tiene que pedir un documento que lo identifique y saber el motivo para lo que esta ingresando.

El archivo para este comportamiento sera *ingresos.feature*

**Característica:** Registro de personas

Como un guardia

Quiero registrar los ingresos y salidas de personas ajenas a la empresa

Para tener un registro de ingresos

**Esquema del escenario:** Registrar visita a oficinas

Dado que ingresa una persona a las oficinas <oficina>

Y con documento <documento>

Y con numero de documento <num\_doc>

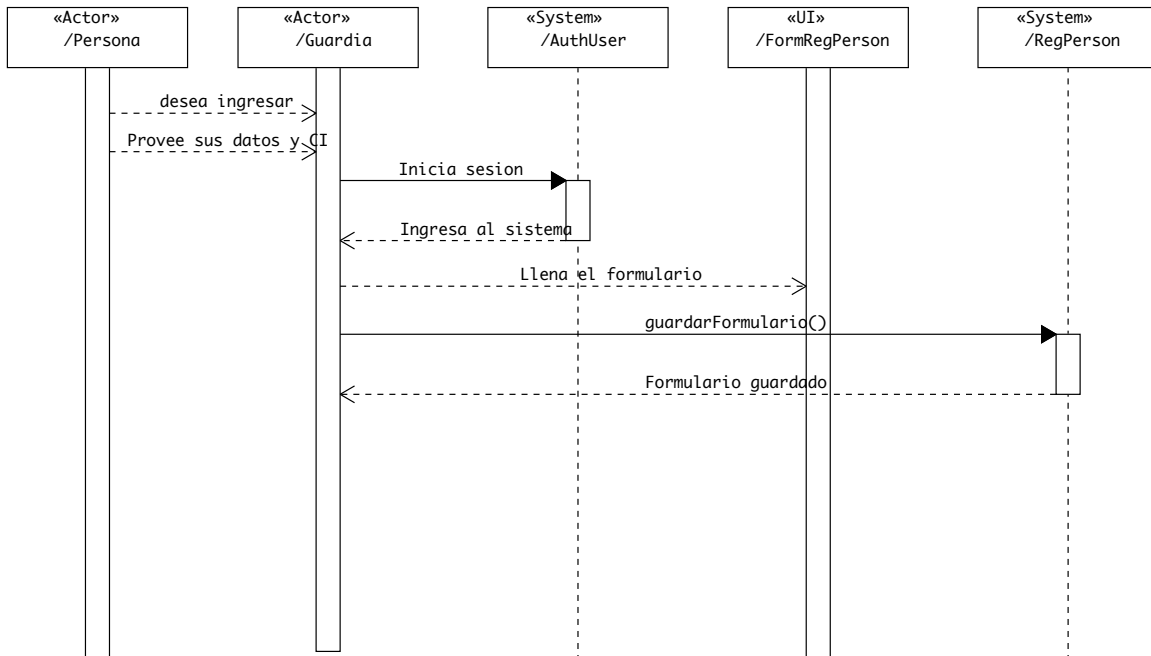
Y con el siguiente nombre <nombre>

Y proviene de la ciudad <ciudad>  
Y con el siguiente motivo <motivo>  
Cuando registre a la persona  
Y el <num\_doc> sea valido  
Entonces registro a la persona <nombre> con documento <num\_doc> que ingreso a la oficina <oficina>

Ejemplos:

oficina	documento	num_doc	nombre	ciudad	motivo
muyurina	pasaporte	123456	Marcelo Roca	br	Reclamos ADSL
central	CI	654321	Pedro Quispe	lpz	Interac TV

Para el registro de personas seguiría la siguiente secuencia.



**Figura 4.1.2:** Diagrama de secuencia para el Registro de Personas

#### 4.1.3. REGISTRO DE MANTENIMIENTO

Los vehículos siempre están en constante *mantenimiento*, pero también se vio la necesidad de tener un registro de que vehículos van al taller para su mantenimiento y también programar mantenimientos preventivos. Esta tarea la realiza el *Encargado de Vigilancia*.

El archivo para este comportamiento sera ***taller.feature***

Característica: Registro de mantenimiento a vehículos  
Como un Encargado de Vigilancia

Quiero registrar mantenimientos preventivos a los vehículos  
Para prevenir accidentes

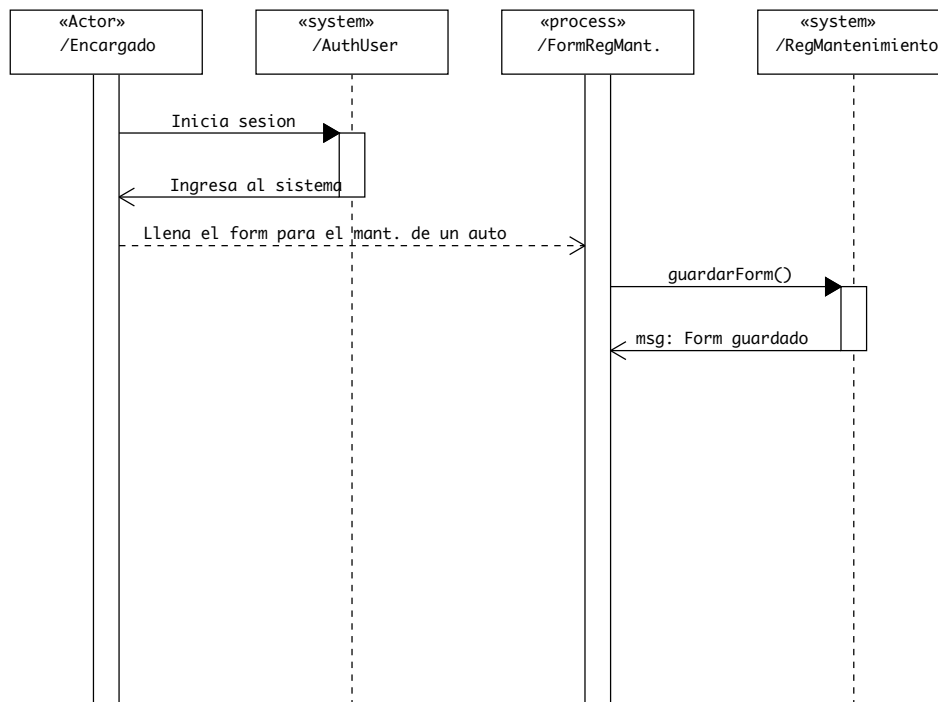
Esquema del escenario: Registrar o programar mantenimientos a vehículos

Dado que el vehículo <interno> del parqueo <parqueo>  
Y su ultimo mantenimiento fue en fecha <fecha>  
Y con un kilometraje de <km\_antiguo>  
Y con el siguiente motivo <motivo>  
Y su kilometraje actual es <km\_actual>  
Y requiere el siguiente mantenimiento <mantenimiento>  
Cuando registre a el mantenimiento  
Y los datos de registro sean validos  
Entonces registro al vehículo <interno> del parqueo <parqueo> para mantenimiento

Ejemplos:

parqueo	interno	fecha	km_antiguo	motivo	km_actual	mantenimiento
muyurina	167	12/01/2014	167944	cambio de aceite	187983	cambio de aceite
sucre	54	25/04/2014	110456	chequeo de motor	150876	revisión de motor

El diagrama de secuencia seria el siguiente:



**Figura 4.1.3:** Diagrama de secuencia para programar mantenimiento

#### 4.1.4. REPORTES

Los *reportes* es algo fundamental en el Sistema Web que se desarrolló, porque ayuda a hacer los diferentes seguimientos de cada parqueo de los vehículos de la

empresa. Esta es una tarea que solo lo puede hacer el *Encargado de Vigilancia*.

El archivo para este comportamiento sera ***reportes.feature***

Característica: Pagina de reportes

Como un Encargado de Vigilancia

Quiero sacar reportes de vehículos

Para control interno

Esquema del escenario: Sacar reportes de parqueos de un día

Dado que necesito el reporte del parqueo <parqueo>

Y solo de la fecha <fecha>

Y lo quiero en formato <formato>

Cuando ingreso los datos

Entonces saco el reporte

Ejemplos:

	parqueo		fecha		formato	
	muyurina		10/06/2014		pdf	
	sucre		15/07/2014		excel	

Esquema del escenario: Sacar reportes de parqueos de un rango de fechas

Dado que necesito el reporte del parqueo <parqueo>

Y con fecha inicial <fecha\_inicial>

Y con fecha final <fecha\_final>

Y lo quiero en formato <formato>

Cuando ingreso los datos

Entonces saco el reporte

Ejemplos:

	parqueo		fecha_inicial		fecha_final		formato	
	muyurina		10/03/2014		26/06/2014		pdf	
	sucre		15/07/2014		26/06/2014		excel	

La gran ventaja que tiene hacer estos escenarios en un lenguaje que entiende el cliente es que puede dar su opinión de acuerdo a que variables quiere que también se contemplen. Por ejemplo, en los anteriores escenarios para los reportes el cliente vio la necesidad de tener también reportes que filtren por ítem de conductor y por el número interno del vehículo. Dado estas aclaraciones con el cliente el escenario quedaría de la siguiente manera:

Esquema del escenario: Sacar reportes de parqueos

Dado que necesito el reporte del parqueo <parqueo>

Y con fecha inicial <fecha\_inicial>

Y con fecha final <fecha\_final>

Y con conductor <item\_conductor>

Y del vehículo <interno>

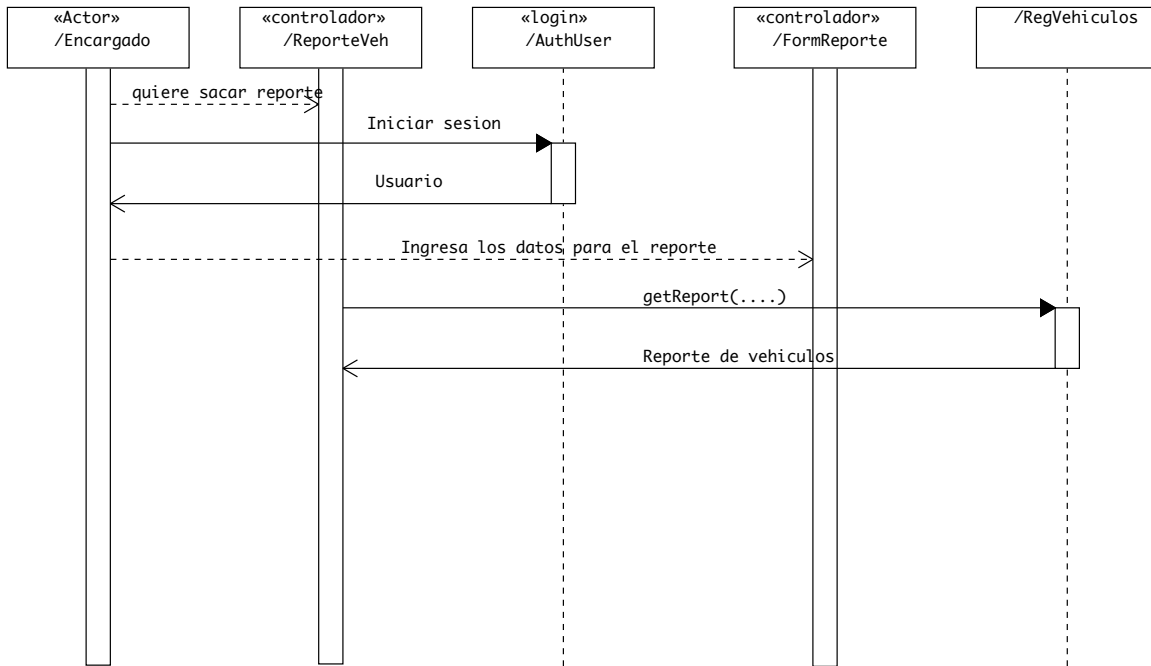
Y lo quiero en formato <formato>

Cuando ingreso los datos

Entonces saco el reporte

Ejemplos:

parqueo	fecha_inicial	fecha_final	item_conductor	interno	formato	
muyurina	10/03/2014	26/06/2014	505	167	pdf	
sucre	15/07/2014	26/06/2014	543	I-02	excel	



**Figura 4.1.4:** Diagrama de secuencia para la emisión de reportes

#### 4.1.5. NOTIFICACIONES

Las *notificaciones* ayuda al *Encargado de Vigilancia* a tomar ciertas medidas de seguridad en la empresa, mas que todo respecto a los vehículos de la empresa y los conductores.

Las notificaciones que se manejan están medidos por *bajo*, *medio* y *alto* solo para estas alertas:

- Un vehículo se queda en un parqueo diferente al que esta asignado - *medio*
- Un vehículo no llega a ningún parqueo - *alto*
- La licencia de conducir de un conductor ya esta por vencer - *alto*

El archivo para este comportamiento sera ***notificaciones.feature***

Característica: Personalizar notificaciones

Como un Encargado de Vigilancia

Quiero dar valor a las notificaciones

Para tomar medidas de seguridad

Esquema del escenario: Cargar notificaciones

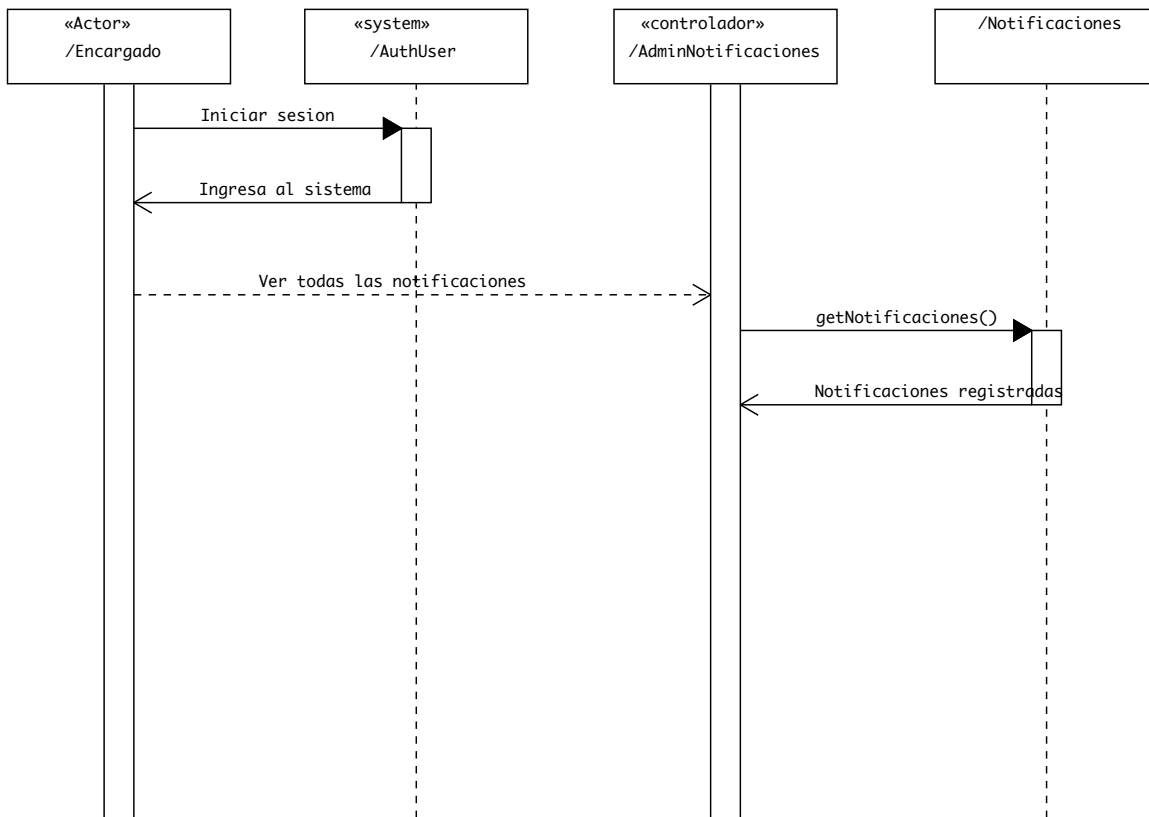
Dado que manejo varias notificaciones

Y con valores diferentes

Cuando asigno valor a las notificaciones

Entonces el sistema debe mandar alertas al encargado

El diagrama de secuencia seria el siguiente:



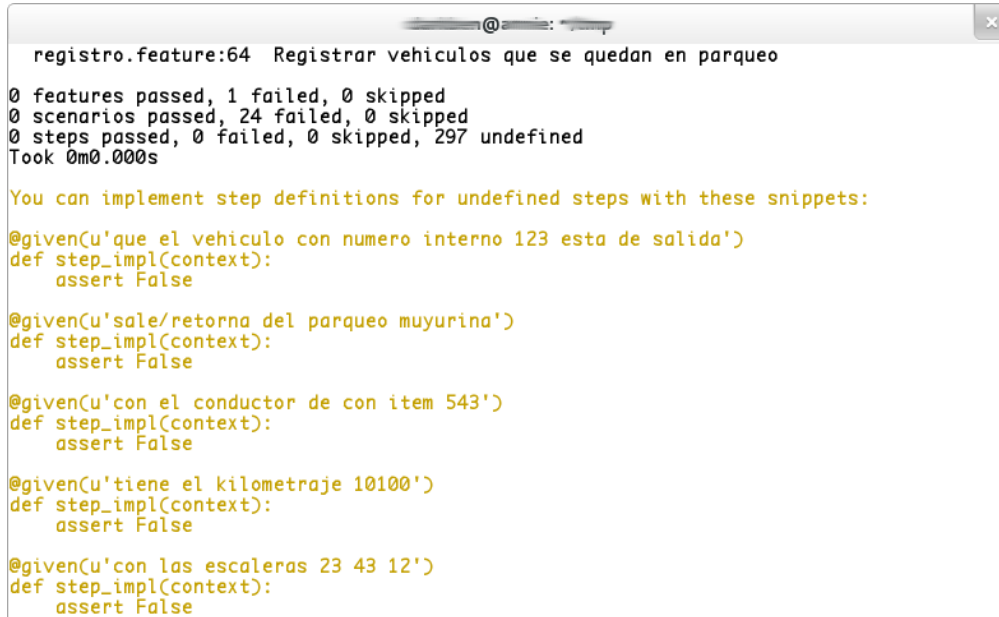
**Figura 4.1.5:** Diagrama de secuencia para la administración de notificaciones

Hasta este punto tenemos claro que es lo que tiene que hacer el sistema, que funcionalidades debe cumplir, lo que veremos a continuación es como podemos darle un valor agregado a todo los escenarios que escribimos con la ayuda de *Behave*

## 4.2. CÓDIGO GENERADO PARA LOS TEST

*Behave* es un interprete de Gherkin que lee todas las especificaciones que escribimos y la forma de ejecutarlo es de la siguiente manera<sup>1</sup>:

```
$ behave registro.feature
```



```
registro.feature:64 Registrar vehiculos que se quedan en parqueo
0 features passed, 1 failed, 0 skipped
0 scenarios passed, 24 failed, 0 skipped
0 steps passed, 0 failed, 0 skipped, 297 undefined
Took 0m0.000s

You can implement step definitions for undefined steps with these snippets:

@given(u'que el vehiculo con numero interno 123 esta de salida')
def step_impl(context):
    assert False

@given(u'sale/retorna del parqueo muyurina')
def step_impl(context):
    assert False

@given(u'con el conductor de con item 543')
def step_impl(context):
    assert False

@given(u'tiene el kilometraje 10100')
def step_impl(context):
    assert False

@given(u'con las escaleras 23 43 12')
def step_impl(context):
    assert False
```

**Figura 4.2.1:** Ejecución de behave

Todas las letras que nos muestra en color naranja son *pasos* o *steps* sin definir y lo que nos muestra es un ejemplo de como podemos escribir estos pasos. En el Anexo B de los *Test de Ejecución* encontrará este código.

Los archivos *.feature* tienen que estar en un directorio llamado **features** y el código escrito en python tiene que estar en otro directorio llamado **steps**, como puede ver a continuación.

Esta es la forma correcta de organizar los features y los steps para Behave, ademas en este directorio encontramos dos archivos nuevos que son:

---

<sup>1</sup>En el Anexo A se mostrará como instalar y configurar las diferentes herramientas



```
darkben@annie: ~/Projects/me/comteco/migration/flosite/features
annie:features darkben$ tree .
.
|-- behave.ini
|-- environment.py
|-- guardia.feature
|-- mantenimiento.feature
|-- notificaciones.feature
|-- registro.feature
`-- steps
    |-- step_guardia.py
    |-- step_mantenimiento.py
    |-- step_notificaciones.py
    `-- step_registro.py

1 directory, 10 files
annie:features darkben$
```

**Figura 4.2.2:** Organización del directorio *features*

```
behave.ini
environment.py
```

En **behave.ini** escribimos variables de entorno que usara Behave al momento de interpretar los archivos *.features*

```
[behave]
lang = es
```

En este caso solo definimos el lenguaje en el que están escritos los escenarios.

En **environment.py** inicializamos variables que usaremos al momento de ejecutar los test.

```
import logging

def before_all(context):
    context.register_car = None
    context.parqueo_out = None
    context.km_out = None
```

```

context.escaleras_out = None
context.item_out = None
context.date_out = None
context.time_out = None
context.dict_cars = {}
context.warning = False

if not context.config.log_capture:
    logging.basicConfig(level=logging.DEBUG)

```

En los test que ejecutamos para este proyecto necesitamos que estén inicializadas todas estas variables antes de todo.

Veamos como ejemplo la ejecución del *registro.feature*:

```
$ behave registro.feature
```

```

(dj6)annie:features darkben$ behave registro.feature
Característica: Pagina de registro # registro.feature:2
  Como un guardia
    Quiero registrar las salidas y retornos de los vehiculos
    Para almacenar los datos
    Esquema del escenario: Registrar vehiculo de la empresa
      Dado que el vehiculo con numero interno 123 esta de salida
      Y sale/retorna del parqueo muyurina
      Y con el conductor de con item 543
      Y tiene el kilometraje 10100
      Y con las escaleras 23 43 12
      Y en fecha 18/02/2014
      Y en horas 08:12
      Cuando registre al vehiculo 123 con su estado de salida
      Y los datos del vehiculo 123 sean validos
      Entonces guardo el registro del vehiculo 123
      Pero que pasa si el conductor del vehiculo 123 de retorno es diferente
        Assertion Failed: Steps must be unicode.

      Y las escaleras del vehiculo 123 de retorno no son las mismas
      Y el parqueo de retorno del vehiculo 123 no es el mismo
      Y si el registro del vehiculo 123 tiene observaciones debe notificarse al supervisor

```

**Figura 4.2.3:** Ejecución con Behave escenario con error

En esta parte de la ejecución los colores nos interesan:

- Verde. Se ejecuta el paso o *step* sin errores.
- Rojo. Ocurrió un error al ejecutar el paso.

- Azul. Pasos que no se ejecutaron.
- Naranja. Pasos no definidos.

En el escenario anterior hay un error, pero mas que un error es una alerta o futura notificación en el sistema, ya que el vehículo ingresa al parqueo pero la persona que ingresa con el vehículo no es la misma que lo saco. El sistema debe ser capaz de agarrar esa alerta al momento de registrar y crear una notificación para el encargado de Vigilancia.

En el siguiente escenario se ejecuta cada paso sin error:

```
Esquema del escenario: Registrar vehiculo de la empresa
  Dado que el vehiculo con numero interno 321 esta de entrada
  Y sale/retorna del parqueo muyurina
  Y con el conductor de con item 432
  Y tiene el kilometraje 20159
  Y con las escaleras 45 14
  Y en fecha 18/02/2014
  Y en horas 10:05
  Cuando registre al vehiculo 321 con su estado de entrada
  Y los datos del vehiculo 321 sean validos
  Entonces guardo el registro del vehiculo 321
  Pero que pasa si el conductor del vehiculo 321 de retorno es diferente
  Y las escaleras del vehiculo 321 de retorno no son las mismas
  Y el parqueo de retorno del vehiculo 321 no es el mismo
  Y si el registro del vehiculo 321 tiene observaciones debe notificarse al supervisor
```

**Figura 4.2.4:** Ejecución con Behave, escenario sin error

En esta fase inicial del desarrollo del producto tenemos claro de todas las funcionalidades que espera ver el cliente en el sistema. Con todo lo desarrollado hasta el momento y los test escritos en *Behave* se tiene como un primer producto entregable, donde mostramos al cliente como pasan los test de acuerdo a las especificaciones.

A continuación veremos como se hizo el modelo de la Base de Datos teniendo en mente todo lo que hicimos hasta este punto. Hablando en términos de BDD hasta este punto del desarrollo es trabajo de los testers y desarrolladores, de aquí para adelante es solo trabajo de los desarrolladores.

### 4.3. MODELO DEL SISTEMA

Según la definición original de BDD expuesta por Dan North<sup>2</sup> nos dice:

*BDD is a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, agile methodology. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters.*

BDD es pensar de afuera hacia adentro(*outside-in*), es decir, una vez que ya tenemos bien pensadas y trabajadas las funcionalidades, eso nos ayuda a tener una idea más clara de lo que tenemos que hacer, como ser los modelos a implementar y los roles que son necesarios crear.

#### 4.3.1. ACL - LISTA DE CONTROL DE ACCESO

Según las características podemos identificar dos roles: *Encargado de Vigilancia* y *Guardia*, que son los que trabajaran con el sistema. Con ACL podemos definir que puede y que no puede hacer cada rol, como vemos a continuación.

	Index	Show	New	Edit	Destroy
Encargado de Vigilancia	★	★	★	★	★
Guardia	★	★	★		

**Figura 4.3.1:** ACL - Lista de Control de Acceso

En la figura anterior se muestra en las columnas las acciones sobre un recurso y en las filas se tiene a los roles definidos en el sistema, en cada intersección existe un valor de *falso* o *verdadero* que indica si el rol tiene permiso de realizar esta acción o si no la tiene.

Esto es de acuerdo a las acciones que realizaran en el sistema:

---

<sup>2</sup>Dan North, principal impulsor en el desarrollo de BDD, <http://dantnorth.net> consultado el 7/07/2014

- como *guardia* debe ser capaz de registrar a los vehículos, es decir crear nuevos registros.
- como *guardia* debe ser capaz de ver todos los registros guardados.
- como *guardia* debe ser capaz de ver los reportes.
- como *encargado de vigilancia* debe ser capaz de editar registros.
- como *encargado de vigilancia* debe ser capaz de borrar registros.
- como *encargado de vigilancia* debe ser capaz de crear usuarios.

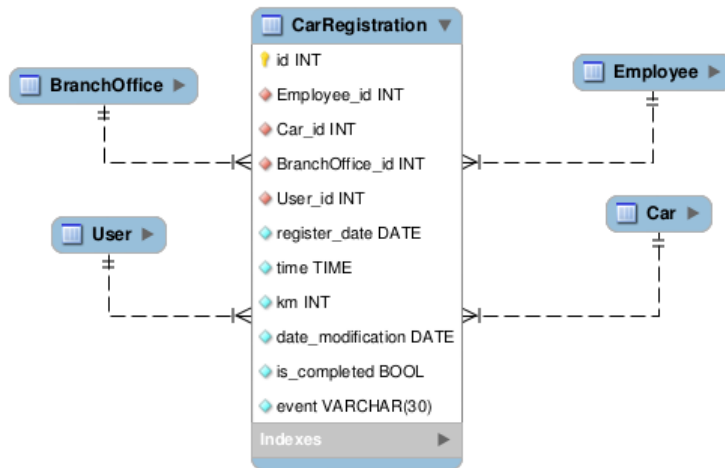
Este enfoque fue el más flexible que se encontró y es el que se implemento en la aplicación.

Guiándonos por los escenarios que vimos en la sección 4.1 nosotros podemos ir modelando con una idea mas clara de que actores o entidades necesitaremos para cada característica.

#### 4.3.2. REGISTRO DE VEHÍCULOS

En base a nuestra primera característica que es el registro de vehículos y los escenarios que la componen tendremos las siguientes entidades para un modelo ER.

- User. Representa al usuario que usa el sistema, para este escenario el usuario es directamente el *guardia*.
- Employee. Es el empleado o funcionario que sale o retorna con el vehículo.
- BranchOffice. La sucursal de donde sale o ingresa el vehículo.
- Car. El Vehículo que es registrado.
- CarRegistration. La tabla donde se registra todos las salidas y retornos de los vehículos.

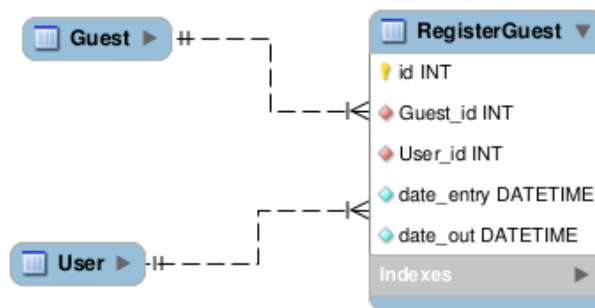


**Figura 4.3.2:** Modelo ER para el registro de vehículos

#### 4.3.3. REGISTRAR INGRESOS A OFICINAS

En base al escenario de *ingreso de personas a la empresa* tenemos las siguientes entidades con su modelo ER respectivo:

- User. Representa al usuario que usa el sistema, para este escenario el usuario es directamente el *guardia*.
- Guest. Es la persona que esta ingresando a la empresa.
- RegisterGuest. La tabla donde se registran todos los ingresos.

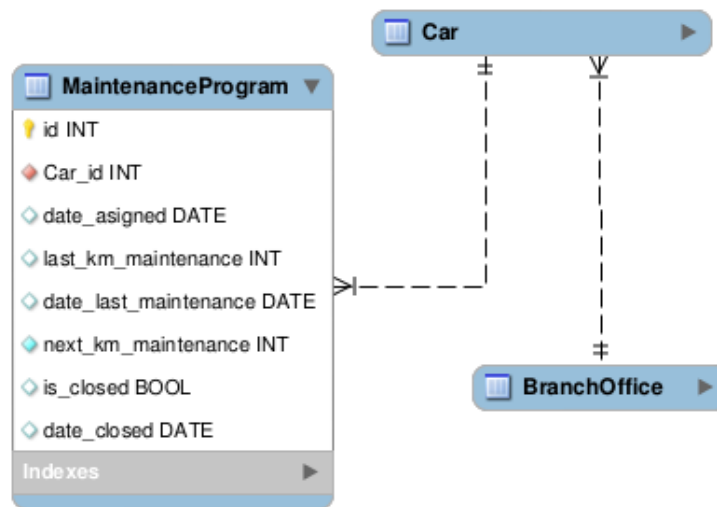


**Figura 4.3.3:** Modelo ER para el registro de visitas

#### 4.3.4. PROGRAMAR MANTENIMIENTO

Para programar mantenimiento de los vehículos se requieren las siguientes entidades:

- Car. El vehículo que es registrado.
- BranchOffice. La sucursal a la que pertenece el vehículo.
- MaintenanceProgram. La tabla donde se registra el mantenimiento programado.



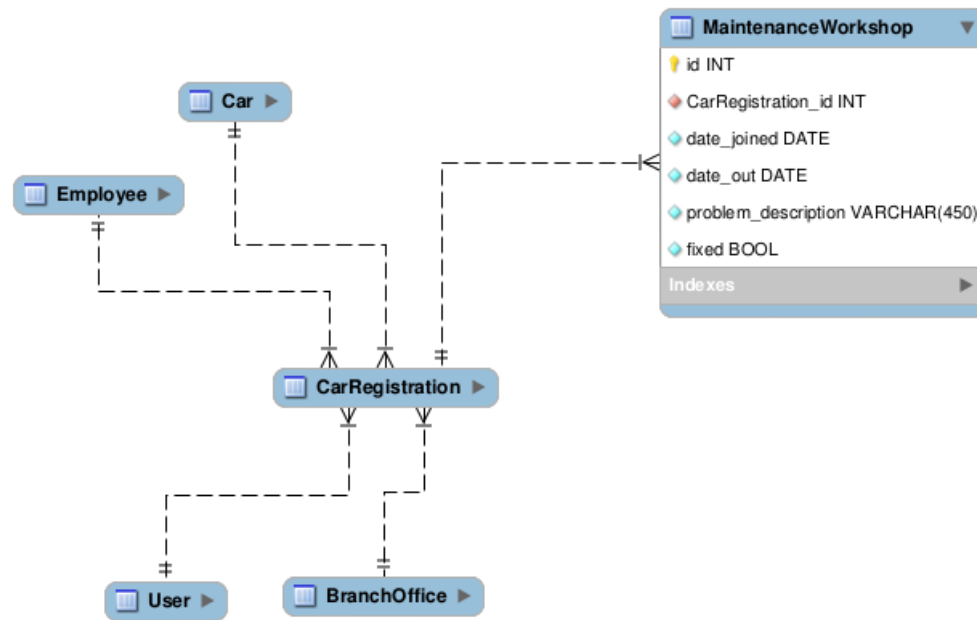
**Figura 4.3.4:** Modelo ER para programar mantenimiento

#### 4.3.5. REGISTRAR SALIDA A TALLER

Las entidades que están involucradas son las mismas que las de registro de ingreso y salida de vehículos excepto que en este escenario tenemos una entidad más, que es la entidad *MaintenanceWorkshop* o taller que es donde se registra cada ingreso de un vehículo al taller de la empresa para su reparación o mantenimiento.

- User. Representa al usuario que usa el sistema, para este escenario el usuario es directamente el *guardia*.
- Employee. Es el empleado o funcionario que sale o retorna con el vehículo.

- BranchOffice. La sucursal de donde sale o ingresa el vehículo.
- Car. El Vehículo que es registrado.
- CarRegistration. La tabla donde se registra todos las salidas y retornos de los vehículos.
- MaintenanceWorkshop. Tabla donde se registra el ingreso de los vehículos al taller de la empresa para su mantenimiento o reparación.



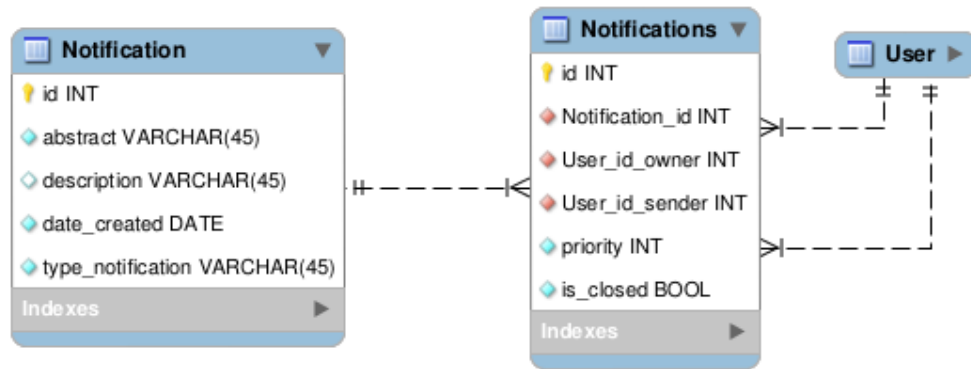
**Figura 4.3.5:** Modelo ER para el registro de salida a taller

#### 4.3.6. ADMINISTRAR NOTIFICACIONES

Las notificaciones tienen la lógica de quien envía *sender* o la crea y para quien es la notificación *owner*.

- User. El usuario que envía la notificación o al que le corresponde.
- Notifications. Las notificaciones con sus respectivos dueños y con la prioridad que le corresponde.
- Notification. Las notificaciones creadas.





**Figura 4.3.6:** Modelo ER para el manejo de notificaciones

Aquí terminamos este desarrollo inicial del sistema, con los siguientes logros:

- Funcionalidades escritas con BDD y probadas con Behave, mostrando así como se ejecuta cada funcionalidad.
- Modelo ER de la Base de Datos

En el siguiente capítulo veremos la parte del desarrollo web y el uso del framework Django.

# 5

## Implementación del Sistema Web

PARA la implementación del sistema web se eligió trabajar con el framework Django<sup>1</sup>.

Django fue desarrollado en un ambiente ágil de redacción y se diseñó para que las tareas comunes de desarrollo web fueran rápidas y simples. Esta escrito en python y usa el paradigma conocido como Model Template View - MTV

### 5.0.7. MODEL TEMPLATE VIEW - MTV

En la interpretación que hace Django al patrón MVC, el *view.py* describe los datos que se presentan al usuario. Esto no es necesariamente como se ven los datos. La *vista* indica qué datos puede ver el usuario no como los ve. Es una distinción sutil.

Así, para Django una “*View*” es la función de devolución en python para una particular URL que esta definida en *urls.py*, debido a que la función de devolución describe que datos son presentados.

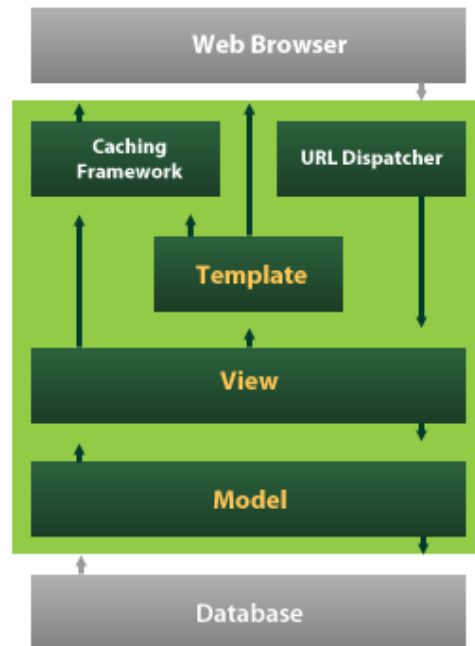
Por otra parte Django hace un separacion del contenido de la presentacion que es donde los *templates* o plantillas entran. En Django una “*vista*” describe que datos

---

<sup>1</sup>Django Foundation, <http://djangoproject.com>, consultado en Julio de 2014

son presentados, pero una vista normalmente delega a un *template*, el cual describe como los datos son presentados.

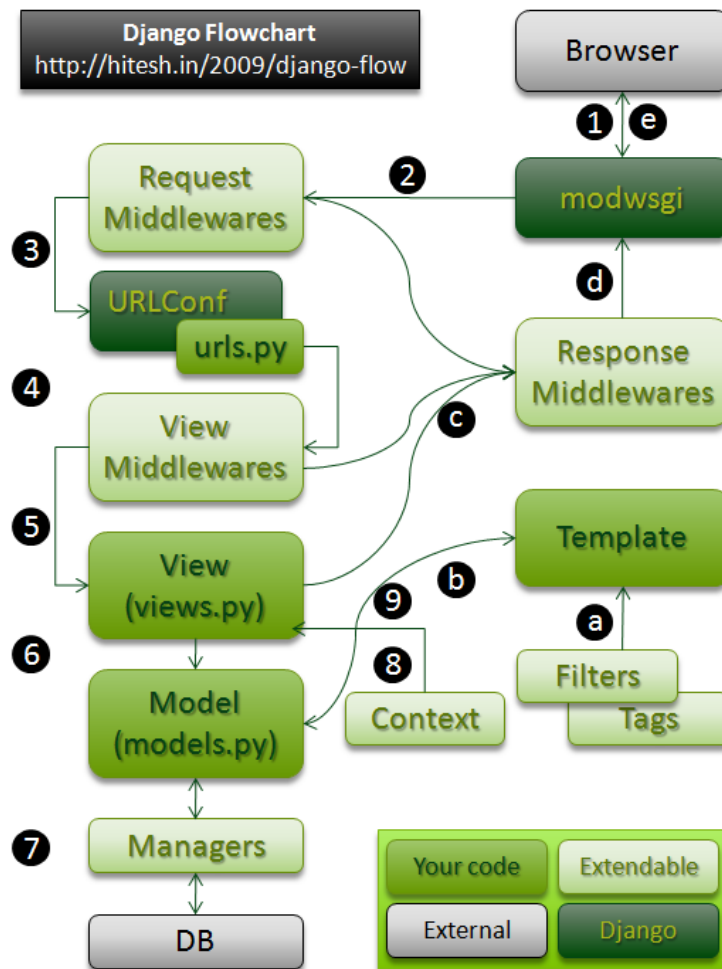
La figura a continuación muestra como interactuan el Modelo con la Vista y Template.



**Figura 5.0.1:** Model View Template, Fuente: <http://djangoproject.org>

Ahora llegamos a la pregunta ¿dónde viene a encajar el “controlador”? En el caso de Django, es probable que sea el Framework en si. La maquina que envía una solicitud a la vista apropiada, de acuerdo con la configuración de los URL de Django y las que definimos en nuestro proyecto que se encuentran en “*urls.py*”.

Para entender a detalle este flujo que sigue Django tenemos la siguiente figura.



**Figura 5.0.2:** Una vista general del flujo de Django, Fuente: <http://hitesh.in/2009/django-flow/>

1. El usuario hace una solicitud de una pagina.
2. La solicitud llega a un *Request Middleware* el cual lo manipula o responde a la solicitud.
3. El *URLConf* busca la vista relacionada usando los *urls.py*
4. *View Middleware* es llamado, que manipula o responde a la solicitud.
5. La función *view* es invocada.
6. La vista puede opcionalmente tener acceso a los datos a través de los modelos.
7. Todas las interacciones de modelo-a-DB se hacen a través de un *manager*.

8. Las vistas podrían usar un *context* especial si es necesario.
9. El context se pasa al *template* para la presentación.
  - a Los templates usan *Filters* y *Tags* para la salida de la presentación.
  - b La salida devuelta a la vista.
  - c *HttpResponse* envía a *Response Middlerwares*.
  - d Cada una de las respuestas de los *Middlewares* puede enriquecer la respuesta o devolver completamente una nueva respuesta.
  - e La respuesta es enviada la navegador del usuario.

## 5.1. ESTRUCTURA DEL PROYECTO

Ya con una vista general de como funciona Django vamos a ver como se estructura un proyecto en Django.

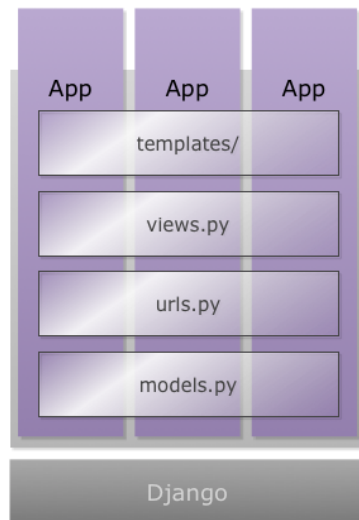
```
flosite
|-- manage.py
|-- flosite (carpeta principal)
|   |-- __init__.py
|   |-- settings.py (archivo de Configuración del proyecto)
|   |-- urls.py (urls que direcciona a las urls de las apps)
|   `-- wsgi.py
`-- app_one
|   |-- templates/ (templates para el uso de este app)
|   |-- __init__.py
|   |-- models.py
|   |-- views.py
|   `-- urls.py (url que direcciona a las views de esta app)
`-- app_two
|   |-- templates/
|   |-- __init__.py
|   |-- models.py
|   |-- views.py
|   `-- urls.py
```

```

`-- templates (Templates para el uso general del proyecto)
|   |-- base.html
|   |-- index.html
`-- staticfiles
    |-- css/
    |-- js/

```

Nuestro proyecto *flosite* cuenta con ocho aplicaciones, como ser: *account*, *branchoffice*, *core*, *guest*, *maintenance*, *notifications*, *register*, *staff*. Algo que es importante mencionar es que desde la versión 1.4 de Django los templates se manejan diferente, ya que se tiene una carpeta *templates* para el uso general del proyecto y cada aplicación maneja su propia carpeta de *templates*.



**Figura 5.1.1:** Estructura de una aplicación de un proyecto Django

El archivo *flosite/urls.py* es muy importante que este bien configurado, ya que este se encarga de delegar las peticiones que se hacen a las aplicaciones correspondientes

```
$ less flosite/urls.py
```

```

from django.conf.urls import patterns, include, url
from django.views.generic import TemplateView
from django.conf import settings
from django.conf.urls.static import static
from django.contrib.staticfiles.urls import staticfiles_urlpatterns

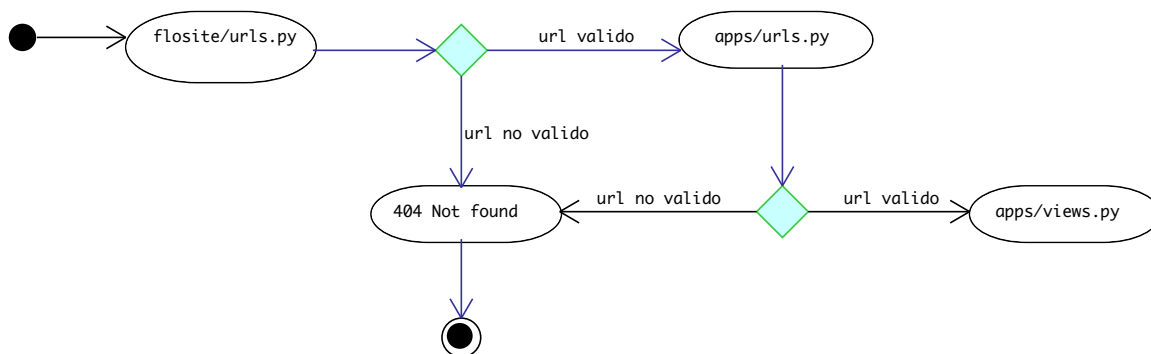
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^$', TemplateView.as_view(template_name='index.html'), name="home"),
    url(r'^accounts/', include('userena.urls')),
    url(r'^messages/', include('userena.contrib.umessages.urls')),
    url(r'^staff/', include('staff.urls')),
    url(r'^branchoffice/', include('branchoffice.urls')),
    url(r'^notifications/', include('notifications.urls')),
    url(r'^maintenance/', include('maintenance.urls')),
    url(r'^guest/', include('guest.urls')),
    url(r'^register/', include('register.urls')),
    url(r'^admin/', include(admin.site.urls)),
) + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

urlpatterns += staticfiles_urlpatterns()

```

Gráficamente tendríamos lo siguiente:



**Figura 5.1.2:** flosite/urls.py

## 5.2. CONFIGURACIÓN DEL PROYECTO

La configuración del proyecto se encuentra en *flosite/settings.py*, no mostraremos todo el archivo de configuración solo resaltaremos las partes más importantes.

### 5.2.1. EL ARCHIVO SETTINGS.PY

Al crear un proyecto con Django<sup>2</sup> se crea un archivo `settings.py` donde uno tiene que modificar el archivo con los datos correspondientes.

```
$ less flosite/settings.py
```

```
DATABASES = {
    'default': {
        # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'nombre_base_de_datos',
        'USER': '#####',
        'PASSWORD': '#####',
        'HOST': '',
        'PORT': '', # Set to empty string for default.
    }
}
```

Para instalar las aplicaciones que hemos creado en el archivo *settings.py* se tiene otra sección específica para esto:

```
INSTALLED_APPS = (
    # default
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # ....

    # project flosite
    'accounts',
    'staff',
)
```

---

<sup>2</sup>Vea el apéndice de Instalación para más detalles



```
'guest',
'branchoffice',
'guest',
'register',
'core',
'notifications',
'maintenance',
```

Todas las aplicaciones que se mencionan en esta sección son las que se instalan al momento de hacer un *syncdb*.<sup>3</sup>

```
$ python manage.py syncdb
```

### 5.3. APLICACIONES EXTRAS PARA EL SISTEMA

Teniendo en cuenta las herramientas y bondades que ofrece Django, para el proyecto usamos las siguientes aplicaciones ya desarrolladas.

```
INSTALLED_APPS = (
    # ....

    #extras
    'bootstrap_toolkit',
    'bootstrap3_datetime',
    'pagination',
    'debug_toolbar',
    'userena',

    # ....
)
```

- `bootstrap_toolkit`: nos ofrece todo el diseño web y paginas de estilo necesarias.
- `bootstrap3_datetime`: nos permite el manejo de calendarios en los forms.
- `pagination`: Esta herramienta es muy importante para la paginación de tablas.

---

<sup>3</sup>Vea el apéndice de Instalación para mas detalles

- debug\_toolbar: Una herramienta útil para debug.
- userena: Para el manejo de usuarios y sus perfiles.



**Figura 5.3.1:** Página de inicio del Sistema Web

## 5.4. FORMULARIOS PARA LOS REGISTROS

Para el sistema tenemos tres formularios: *Registro de vehículos*, *Registro de Personas*, *Reportes*

### 5.4.1. REGISTRO DE VEHÍCULOS

Los formularios para los registro se encuentra en *register/forms.py*

```
$ less register/forms.py
```

```
from django import forms
```

```
class RegisterKmForm(forms.Form):
    r_automotive = forms.CharField(widget=forms.TextInput(attrs={...}))
    r_km_revert = forms.BooleanField(required=False)
    r_date      = forms.DateField(widget=BootstrapDateInput(attrs={...}))
    r_hour      = forms.TimeField(widget=forms.TextInput(attrs={...}))
```

```

r_km      = forms.IntegerField(label=_('output_km'), widget=...)
r_driver  = forms.CharField(widget=forms.TextInput(attrs={...}))
r_ladder  = forms.CharField(required=False)
r_obs     = forms.CharField(widget=forms.Textarea, required=False)

# validators
def ...

```

A estos formularios se los llama desde *register/views.py* y lo carga en el template *register/templates/register\_form.html* para tener el siguiente formulario:

El código de *views.py* para registro de vehículos sería el siguiente:

```

from django.shortcuts import render, ...
from register.forms import RegisterKmForm, ...

@login_required
@csrf_protect
@permission_required_or_403('register.add_register_automotive')
def form_car(request):
    ....
    if request.method == 'POST':
        form = RegisterKmForm(request.POST) # Esta linea llama al form
        if form.is_valid():
            ....
    else:
        form = RegisterKmForm()
    return render(request, 'register_form.html', {'form': form})

```

La ultima linea es la que se encarga principalmente de renderizar el template con el formulario.

```

return render(request, 'register_form.html', {'form': form})

```

Y con todo, tendríamos la siguiente vista:

Control de acceso : Registro - Iceweasel

192.168.1.1/register/form\_car/

Registro de Vehiculos Registro de Personas URAS Reportes benjamin

OPTIONS  
Ver registros  
Incompletos

### Registro de ingreso y salida de vehiculos

Fecha de la planilla a registrar  
21 de agosto de 2014

Cambiar Fecha

Fecha de registro  
13/08/2014 ☐ Mantener esta fecha de registro

Agosto 2014

Do	Lu	Ma	Mi	Ju	Vi	Sa
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Cambiar de chofer

Observaciones

Guardar

**Figura 5.4.1:** Formulario de Registro de Vehículo

#### 5.4.2. REGISTRO DE PERSONAS

El formulario para el registro de personas se encuentra también en *register/forms.py* y sería el siguiente:

```
class RegisterPersonForm(forms.Form):
    LOCALE_CHOICES = (
        ('cba', 'Cochabamba'),
        ('lpz', 'La Paz'),
        ('scz', 'Santa Cruz'),
        ('pnd', 'Pando'),
        ('ben', 'Beni'),
        ('oru', 'Oruro'),
        ('pts', 'Potosi'),
        ('tja', 'Tarija'),
        ('scr', 'Chuquisaca'),
        ('---', _('none')),
    )
```

```

GENDER_CHOICES = (
    ('H', 'Hombre'),
    ('M', 'Mujer')
)
identity_document = forms.CharField(required=True, widget=...)
city = forms.CharField(required=True, widget=forms.Select(...))
first_name = forms.CharField(required=True)
last_name = forms.CharField(required=True)
gender = forms.CharField(required=True, widget=forms.Select(...))
reason = forms.CharField(required=True, widget=forms.Textarea)

```

La llamada desde el *views.py*

```

from register.forms import RegisterPersonForm

@login_required
@csrf_protect
@permission_required_or_403('register.add_register_person')
def form_register_person(request):
    if request.method == 'POST':
        form = RegisterPersonForm(request.POST)
        if form.is_valid():
            ...
            return HttpResponseRedirect('/register/persons')
    else:
        form = RegisterPersonForm()
    return render(request, 'form_register_person.html', {'form': form})

```

Esta vista llama al template *form\_register\_person.html*

Registro de Vehiculos Registro de Personas URAS Reportes benjamin

OPTIONS  
 Personas Registradas del día  
 Todas las Personas Registradas

### Registro de ingreso de personas

Numero de CI:

De:

Nombre:

Apellido:

Sexo:

Motivo de ingreso:

© 2012 <Corporacion Comiteco>

**Figura 5.4.2:** Formulario para el Registro de Personas

### 5.4.3. REPORTES

El formulario para los reportes se encuentra *report/forms.py*

```
class ReportForm(forms.Form):
    generate_pdf = forms.BooleanField(required=False)
    generate_excel = forms.BooleanField(required=False)
    parking = forms.ModelChoiceField(queryset=LocaleParking.objects.all(),
                                     empty_label="(seleccione un parqueo)")
    today = forms.DateField(label='today', widget=BootstrapDateInput(...))
    date_begin = forms.DateField(widget=BootstrapDateInput(...))
    date_end = forms.DateField(widget=BootstrapDateInput(format="%d/%m/%Y"), ...)
    internal_num_car = forms.CharField(max_length=10, required=False)
    item_driver = forms.CharField(max_length=10, required=False)
```

La llamada desde el vista es la siguiente y se encuentra en *report/views.py*

```
@login_required
def home(request):
    if 'csrfmiddlewaretoken' in request.GET:
        form = ReportForm(request.GET)
```

```

        if form.is_valid():
            .....
    else:
        return render_to_response('report/list.html', 'report': report.order_by(...))
    else:
        form = ReportForm()

    return render(request, 'report/report.html', 'form': form)

```

Esta vista llama al template *report/report.html*

Control de acceso : Registro - Iceweasel

Iceweasel Control de acceso : Registro

192.168.1.11/report/

Registro de Vehiculos Registro de Personas URAS Reportes benjamin

OPTIONS

[Buscar escalera](#)

[Reporte de URAs](#)

Introduzca los datos para emitir el reporte de Vehiculos

Parqueo

(seleccione un parqueo)

Introduzca una fecha

Introduzca un rango de busqueda

Fecha de inicio

Fecha de fin

13/08/2014

Busquedas por vehiculo o conductor

Numero interno del Vehiculo

Item del Conductor

☐ Crear documento PDF

☐ Crear documento Excel

Crear reporte

**Figura 5.4.3:** Formulario para Reportes

# 6

## Pruebas del Sistema

EN el capítulo tres vimos como la metodología BDD es una herramienta útil que fomenta la colaboración entre desarrolladores, testers y clientes. Entendimos como BDD es una técnica de desarrollo ágil de software.

En el capítulo cuatro vimos la aplicación de BDD en el proyecto e hicimos los escenarios respectivos para cumplir los requerimientos de funcionalidad del sistema.

- Registro de Vehículos
- Registro de Ingreso de personas
- Registro de mantenimiento
- Reportes
- Notificaciones

En este capítulo mostraremos como esos escenarios que ya definimos nos sirven para hacer las pruebas finales.

Los escenarios y el código que desarrollamos para que pasen los tests eran solo una muestra de como el sistema iba a cumplir todos los requerimientos de funcionalidad.



Hasta este punto del proyecto ya se tiene el sistema completo. Y ahora es cuando usaremos esos mismos escenarios para mostrar como el sistema cumple con las funcionalidades exigidas.

Para realizar estas pruebas necesitaremos de dos herramientas mas en el entorno virtual de desarrollo.

- `django-behave`<sup>1</sup>
- `splinter`<sup>2</sup>

## 6.1. DJANGO-BEHAVE

Es un TestRunner de Django para el modulo de Behave BDD, ya en el capítulo tres y cuatro vimos como trabaja *behave* con BDD.

### 6.1.1. INSTALACIÓN

Para la instalación primero entraremos a nuestro entorno virtual de desarrollo y usando *pip*<sup>3</sup> lo instalaremos.

```
project$ source ~/env/bin/active
(env)project$ pip install django-behave
```

### 6.1.2. CONFIGURACIÓN

Una vez instalado `django-behave` pasamos a hacer la configuración en el archivo *settings.py* y hacemos los cambios como se muestra a continuación.

```
INSTALLED_APPS = (
    ..
    'django_behave',
)

TEST_RUNNER = 'django_behave.runner.DjangoBehaveTestSuiteRunner'
```

---

<sup>1</sup>Django-Behave, desarrollado por [Willmer](#)

<sup>2</sup>Splinter, desarrollado por [CobraTeam](#)

<sup>3</sup>PIP, es un manejador de paquetes de Python

## 6.2. SPLINTER

Splinter<sup>4</sup> es una herramienta Open Source para hacer pruebas en aplicaciones web usando Python. Permite automatizar acciones dentro del navegador, como visitar URLs e interactuar con los items.

### 6.2.1. INSTALACIÓN

Para instalarlo usaremos *pip* como en los casos anteriores y siempre trabajando dentro de nuestro entorno virtual:

```
(env)project$ pip install splinter
```

Enhorabuena, ya tenemos instalado *django-behave* y *splinter*, ahora la forma de usarlo es de la siguiente manera:

```
(env)project$ python manage.py test app
```

Como ya tenemos configurado django-behave en *settings.py* el archivo *manage.py* reconoce la orden *test* y lo que viene después es el nombre de la aplicación que en el ejemplo solo se lo llama *app*.

Pero *splinter* no se ejecuta de manera mágica, necesitamos decirle a los *features* de BDD de cada aplicación de nuestro proyecto que debe usar splinter al momento de ejecutar los tests.

## 6.3. CONFIGURACIÓN DE LOS ESCENARIOS

En el capítulo cuatro pusimos todos los escenarios en una misma carpeta *features* y cuando lo ejecutábamos con *behave* recorría todos los escenarios.

Pero ahora, como ya tenemos el sistema completo con sus aplicaciones respectivas para cada funcionalidad vamos a llevar los escenarios a la aplicación que le corresponde, veamos el caso de *accounts*.

```
flosite
|-- manage.py
|-- flosite (carpeta principal)
|   |-- ...
```

---

<sup>4</sup>Splinter, <http://splinter.cobrateam.info/en/latest/>, consultado en noviembre de 2014

```

|-- accounts/
| |-- features
| | |-- behave.ini
| | |-- environment.py
| | |-- login.feature
| | |-- `-- steps
| |         |-- step_login.py
| |-- __init__.py
| |-- models.py
| |-- urls.py
| |-- `-- views.py
|-- ...

```

En la aplicación *accounts* ahora tenemos la carpeta *features* y es ahí donde están los archivos de Configuración y su escenario *login.feature*, veamos *behave.ini*:

```

[behave]
lang = es

```

En este archivo le decimos de manera explícita que los escenarios están escritos en *Español*.

El archivo *environment.py* es el que mas nos interesa ahora, porque es ahí donde le decimos a *django-behave* que use splinter al momento realizar los tests.

```

from splinter.browser import Browser

def before_all(context):
    context.browser = Browser()

def after_all(context):
    context.browser.quit()
    context.browser = None

```

El método *before\_all*, se encarga de que antes de cualquier cosa inicialice el Browser o el navegador.

El método *after\_all*, se encarga de que después que realice todos los pasos cierre el navegador.

## 6.4. ESCENARIO ACCOUNT

Como ya mencionamos, trabajaremos con *accounts* para ver como se ejecuta el test con splinter paso a paso. Veamos el archivo *login.features*:

```
# language: es
```

```
Característica: Pagina principal
```

```
    Como un guardia
```

```
    Quiero ingresar al sistema
```

```
    Para registrar ingresos
```

```
Esquema del escenario: Iniciar sesión
```

```
    Dado que ingreso al sistema con el url "http://127.0.0.1:8000/"
```

```
    Y voy a la opcion "Ingresar"
```

```
    Y entro con mi nombre de usuario <username>
```

```
    Y mi contraseña <password>
```

```
    Cuando oprima el boton "Ingresa"
```

```
    Entonces ingreso al sistema y leo el mensaje "Bienvenido"
```

```
Ejemplos:
```

```
    | username | password |
```

```
    | benjamin | asdf    |
```

A diferencia de los escenarios que ya vimos, aquí le decimos que tiene que entrar a una dirección URL.

Una vez hecho esto en la carpeta **steps** escribimos el código necesario para este escenario en el archivo *step\_login.py*.

```
# encoding: utf-8
```

```
from behave import given, when, then
```

```
@given(u'que ingreso al sistema con el url "{url}"')
```

```
def impl(context, url):
```

```
    br = context.browser
```

```
    br.visit(url)
```

```
@given(u'voy a la opcion "{option}"')
```

```
def impl(context, option):
```

```
    link = context.browser.find_link_by_text(option).first
```

```
    assert link
```

```
    link.click()
```

```

@given(u'entro con mi nombre de usuario {usr}')
def impl(context, usr):
    context.browser.fill('identification', usr)
    context.user = usr
    assert usr

@given(u'mi contraseña {pwd}')
def impl(context, pwd):
    context.browser.fill('password', pwd)
    context.password = pwd
    assert pwd

@when(u'oprima el boton "{btn}"')
def impl(context, btn):
    button = context.browser.find_by_value(btn).first
    assert button
    button.click()

@then(u'ingreso al sistema y leo el mensaje "{msg}"')
def impl(context, msg):
    assert msg in context.browser.html

```

Cada método recibe como atributo a *context* y como en el archivo *environment.py* ya lo inicializamos con splinter, solo tenemos que llamar a sus métodos e ir recorriendo la pagina siguiendo los pasos escritos en *login.features*.

Una vez escrito y configurado todo esto ejecutamos el test:

```
(env)project$ python manage.py test accounts
```

Cuando se ejecuta el test llama a *splinter* que a la vez ejecuta un navegador de pruebas donde recorre los pasos que hemos definido en los escenarios.

Esto podemos verlo en el siguiente vídeo que es de mi autoría puesto en youtube.com con el siguiente título *BDD, django-behave and splinter*<sup>5</sup> en la URL: <http://youtu.be/KT6DCFfCFnY>

---

<sup>5</sup>BDD, django-behave and splinter: publicado el 2 de diciembre de 2014

# 7

## Puesta en producción

PARA la puesta en producción del sistema es preciso tomar en cuenta otras herramientas como el Sistema Operativo donde se ejecutara el sistema, el Servidor Web que usaremos para desplegar el proyecto, etc.

### 7.1. SERVIDOR DEBIAN LINUX

Debian es una distribución de Linux, un Sistema Operativo Libre con acceso en línea a los diferentes repositorios que tiene.

Se eligió Debian por el soporte estable que tiene para servidores y por lo practicó que es al momento de instalar los paquetes de software.

La versión que usaremos para el proyecto es Debian 7.7 que tiene el nombre de *Wheezy*.

### 7.2. SERVIDOR WEB NGINX

Nginx es un Servidor Web ligero y Libre, además de eso es de alto rendimiento. Es conocido por su estabilidad, gran conjunto de características, configuración

simple, y bajo consumo de recursos. También cuenta con un buen soporte de para FastCGI. Nginx y FastCGI<sup>1</sup> es una buena elección al momento de desarrollar con Django y es lo que usaremos para el proyecto, aunque no es el único camino.

La forma de instalar en Debian es la siguiente:

```
# aptitude install nginx
```

Y la versión que instalará y la que usaremos es la 1.2.1

### 7.3. BASE DE DATOS MYSQL

MySQL es un sistema de Base de Datos estable, rápido, multi-usuario y multi-hilo. La principal meta de MySQL es ser rápido, robusto y fácil de usar.

La forma de instalar es la siguiente:

```
# aptitude install mysql-server
```

Esto nos instalará la versión 5.5 de MySQL.

### 7.4. INSTALACIÓN Y CONFIGURACIÓN DEL SISTEMA

Para no tener conflictos de versiones de paquetes de software en el servidor crearemos un entorno virtual para el proyecto donde ahí instalaremos todos los paquetes que son requeridos para el funcionamiento del sistema.

Para ello necesitamos instalar en el servidor *virtualenv*, esto nos permitirá crear el entorno virtual que necesitamos para el proyecto, pero antes debemos instalar *python-pip* y una vez hecho esto con *pip* instalaremos *virtualenv*.

```
# aptitude install python-pip
```

```
# pip install virtualenv
```

- *pip* - Es un manejador de paquetes de software para Python
- *virtualenv* - Es una herramienta para mantener las dependencias de las versiones de los paquetes de software requeridas por los diferentes proyectos y separarlas.

---

<sup>1</sup>FastCGI es un protocolo para interconectar con un Servidor Web

Ahora creamos nuestro entorno virtual llamado *env* y lo activamos. Y esto lo hacemos como un usuario normal ya no como *root*.

```
user:~$ virtualenv env
user:~$ source env/bin/activate
(env)user:~$
```

En Debian y en otras distribuciones Linux se tiene */var/www/* para los sitios Web y es ahí donde copiaremos el sistema */var/www/flosite/*.

Pero antes de instalar el sistema crearemos la Base de Datos y un usuario con privilegios.

```
mysql> create database projectdb;
mysql> create user userdb identified by 'xxxxxx';
mysql> grant all privileges on projectdb.* to userdb;
```

Ingresamos a la carpeta del proyecto pero primero activamos el entorno virtual que ya creamos anteriormente.

```
user:~$ source env/bin/activate
(env)user:~$ cd /var/www/flosite/
(env)user:/var/www/flosite$
```

Para facilitar la instalación tenemos el siguiente archivo *requirements/project.txt*, ahí se tiene todas las dependencias del proyecto y lo usamos de la siguiente manera:

```
(env)user:/var/www/flosite$ pip install -r requirements/project.txt
```

Hasta este punto ya tenemos todo lo necesario para que el sistema se ejecute sin problemas, ahora vamos a sincronizar el proyecto con la Base de Datos para la creación de las tablas.

Primero configuramos *settings.py* para la Base de Datos.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'projectdb',
        'USER': 'userdb',
        'PASSWORD': 'xxxxxxx',
```



```

        'HOST': '',
        'PORT': '',
    }
}

```

Una vez editado y guardado *settings.py* sincronizamos el proyecto con la Base de Datos y usamos la instrucción *syncdb* para ello.

```
(env)user:/var/www/flosite$ python manage.py syncdb
```

Y para terminar levantamos el sistema:

```
(env)user:/var/www/flosite$ python manage.py runserver
```

En el navegador del servidor ingresamos al URL <http://127.0.0.1:8000> y efectivamente el sistema esta ejecutándose, pero no pueden ingresar desde otras maquinas, como ya contamos con una IP pública hacemos lo siguiente:

```
(env)user:/var/www/flosite$ python manage.py runserver 0.0.0.0:8000
```

Ahora todos pueden ingresar al sistema usando el IP del servidor como URL <http://xxx.xxx.xxx.xxx:8000>

Pero ejecutar el sistema de esta forma es incomodo y ademas que tenemos que ejecutarlo manualmente. En la siguiente sección veremos como automatizar esta tarea en el servidor.

## 7.5. CONFIGURACIÓN EN EL SERVIDOR

Bien, ahora es el turno de Nginx y Debian para el paso final de la puesta en producción del sistema.

### 7.5.1. NGINX

Vamos a los archivos de configuración de Nginx en */etc/nginx/* y en *sites-available* creamos el archivo *flosite.conf*

```

root:~# cd /etc/nginx/sites-available/
root:/etc/nginx/sites-available# touch flosite.conf

```

En *flosite.conf* copiamos las siguientes lineas.

```

server {
    listen 80;
    server_name IP.de.nuestro.servidor;
    access_log /var/log/nginx/flosite.access.log;
    error_log /var/log/nginx/flosite.error.log;

    location /static/ { # STATIC_URL
        alias /var/www/flosite/staticfiles/; # STATIC_ROOT
        expires 30d;
    }

    location /static/admin/ { # STATIC_ADMIN
        alias /home/user/env/lib/python2.7/site-packages/django/.../admin/;
        expires 30d;
    }

    location /media/ { # MEDIA_URL
        alias /var/www/flosite/media/; # MEDIA_ROOT
        expires 30d;
    }

    location / {
        include fastcgi_params;
        fastcgi_pass 0.0.0.0:8080;
        fastcgi_split_path_info ^()(.*)$;
        fastcgi_read_timeout 20m;
    }
}

```

Ahora creamos un enlace simbólico en la carpeta *sites-enabled*

```
root:/etc/nginx# ln -s sites-available/flosite.conf sites-enabled/
```

Reiniciamos Nginx para que reconozca la nueva configuración que hemos cargado.

```
# service nginx restart
```

Pero aun nos falta algo, si se fija en la sección de *location* necesitamos pasarle un *fastcgi* a nuestra configuración de Nginx, para esto ejecutamos el siguiente comando:

```
(env)user:/var/www/flosite$ python manage.py runfcgi host=0.0.0.0 port=8080
(env)user:/var/www/flosite$
```

Esto crea el *fastcgi* que necesita la configuración que creamos en Nginx.

Si vamos al navegador e ingresamos con nuestra IP, efectivamente esta funcionando <http://ip.de.nuestro.servidor/>

Pero aun hay un comando que hay que escribir manualmente. Ahora es el turno de Debian Linux.

### 7.5.2. CONFIGURACIÓN EN EL SERVIDOR DEBIAN LINUX

Para automatizar la tarea de levantar el sistema al momento de iniciar el servidor necesitamos usar *update-rc.d* que es el comando usado para decidir la secuencia y el runlevel de ejecución del script que crearemos.

Creamos nuestro script *flosite* en */etc/init.d/*

```
# touch /etc/init.d/flosite
```

Ahora copiamos las lineas a continuación en el archivo creado.

```
#!/bin/bash
#
### BEGIN INIT INFO
# Provides:          flosite
# Required-Start:    mysql
# Required-Stop:     mysql
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Inicia el sistema de flosite
# Description:       Este script fue escrito por Benjamin Perez
### END INIT INFO

# Este script es para levantar el sistema de seguridad industrial
echo "This part always gets executed"

case "$1" in
    start)
        echo "Starting flosite"
```

```

    echo "Please wait . . . ."
    source /home/user/env/bin/activate
    cd /var/www/flosite
    python manage.py runfcgi host=0.0.0.0 port=8080
    deactivate
    echo "Done"
    ;;
stop)
    echo "Stopping flosite"
    echo "Please wait . . ."
    # TODO
    echo "Done"
    ;;
restart)
    echo "Restarting flosite"
    echo "Please wait . . . ."
    # TODO
    ;;
*)
    echo "Usage: /etc/init.d/flobasic {start|stop|restart}"
    exit 1
    ;;
esac

exit 0

```

En este script algo que es muy importante son estas lineas:

```

# Required-Start:    mysql
# Required-Stop:     mysql

```

Con estas lineas le decimos que nuestro script se ejecute o se detenga una vez que ya inicio o se detuvo MySQL. Esto es importante por que si el script se ejecuta antes de MySQL nos dará error y no podremos ingresar al sistema.

Ahora le damos permisos de ejecución:

```

# chmod +x /etc/init.d/flosite

```

Finalmente ejecutamos *update-rc.d* para que el script se ejecute automáticamente al iniciar el servidor.

```
# update-rc.d /etc/init.d/flosite defaults
```

Al usar la opción *defaults* *update-rc.d* creará enlaces para iniciar el servicio en los runlevels 2345 y detendrá el servicio en los runlevels 016.

Los *runlevels* básicamente se refieren a los diferentes niveles que tiene un Servidor Linux para iniciar y parar servicios.

Con esto ya tenemos un servidor listo para ejecutar automáticamente el proyecto.

# 8

## Conclusiones y recomendaciones

### 8.1. CONCLUSIONES

El sistema se pudo terminar satisfactoriamente. Se logró alcanzar las expectativas que se pusieron desde un inicio, cumpliendo todas las funcionalidades que se necesitan para el control de los vehículos como de las personas que ingresan a las diferentes oficinas.

Algunas conclusiones puntuales:

- El uso de una metodología ágil como es BDD, fue excelente al momento de entender las funcionalidades exigidas por el usuario final.
- El uso de Django que es un Framework Web de alto nivel escrito en Python, ayudó al momento de escribir modelos y obtener un buen rendimiento del sistema.
- Para las pruebas finales el uso de Splinter fue muy útil para ver como las funcionalidades escritas en los escenarios se cumplían satisfactoriamente ya en el sistema.
- El uso de herramientas libres ayudó a no tener problemas de licencias, ya que

todas están basadas en licencias libres como son GPL, BSD, etc.

- El tener conocimientos de Sistemas de Control de Versiones como es Git fue muy útil al momento de escribir código, para tener un historial de como fue creciendo el sistema y tener respaldo del mismo en diferentes versiones del código.
- Personalmente como programador llegue a la conclusión de que Linux es un buen entorno de desarrollo, al momento de instalar y configurar herramientas, ya sean en mi máquina personal como en el servidor. También al momento de escribir código, porque no tengo que depender de IDEs de desarrollo muy grandes como son Eclipse, Netbeans, PyCharm, etc. Sino que tan solo usando VIM con los plugins que tiene para Python, Git y otros es excelente. Además de lo versátil que es Linux.

## 8.2. RECOMENDACIONES

Las recomendaciones mas importantes al concluir el sistema serian las siguientes:

- Para el buen uso del sistema se recomienda leer previamente el manual de usuario que se encuentra en el **Anexo C** y que se esta entregando digitalmente.
- Siempre es recomendable realizar periódicamente copias de respaldo de la Base de Datos para evitar perdida de información.



## Instalación y configuración de herramientas



## A.1. ENTORNO VIRTUAL

Para crearnos un entorno virtual de desarrollo que no afecte en las dependencias de versiones de otros programas usaremos *virtualenv* y para esto seguimos los siguientes pasos como *root*.

```
# aptitude install python-pip
# pip install virtualenv
```

- *pip* - Es un manejador de paquetes de software para Python
- *virtualenv* - Es una herramienta para mantener las dependencias requeridas por los diferentes proyectos y separarlas.

Ahora creamos nuestro entorno virtual llamado *env* y lo activamos. Y esto lo hacemos como un usuario normal ya no como *root*.

```
user:~$ virtualenv env
user:~$ source env/bin/activate
(env)user:~$
```

Note que delante de nuestro nombre de usuario aparece el nombre de nuestro entorno virtual: *(env)*

## A.2. BEHAVE

Para instalar *behave* lo haremos usando *pip*, pero lo haremos dentro de nuestro entorno virtual de desarrollo.

```
(env)user:~$ pip install behave
```

Con esto ya tenemos instalado *behave* y lo podemos usar de la siguiente manera:

```
(env)user:~/project/features$ behave registro.feature
```

## A.3. DJANGO-BEHAVE

Para la instalación primero entraremos a nuestro entorno virtual de desarrollo y usando *pip* lo instalaremos.

```
user:~ $ source env/bin/active
(env)user:~ $ pip install django-behave
```

### A.3.1. CONFIGURACIÓN

Una vez instalado django-behave pasamos a hacer la configuración en el archivo *settings.py* y hacemos los cambios como se muestra a continuación.

```
INSTALLED_APPS = (  
    ..  
    'django_behave',  
)  
  
TEST_RUNNER = 'django_behave.runner.DjangoBehaveTestSuiteRunner'
```

### A.4. SPLINTER

Para instalarlo usaremos *pip* como en los casos anteriores y siempre trabajando dentro de nuestro entorno virtual:

```
(env)user:~ $ pip install splinter
```

El archivo *environment.py* que se encuentra en la carpeta *features* de cada *app* del proyecto es donde escribimos el siguiente código, porque es ahí donde le decimos a *django-behave* que use splinter al momento realizar los tests.

```
from splinter.browser import Browser  
  
def before_all(context):  
    context.browser = Browser()  
  
def after_all(context):  
    context.browser.quit()  
    context.browser = None
```

La forma de usarlo es de la siguiente manera:

```
(env)project$ python manage.py test app
```

# B

Tests de Ejecución

En el capítulo 4 vimos los escenarios de funcionalidad del sistema, en este anexo mostraremos los tests escritos en Python que validan esas funcionalidades.

## B.1. STEPS - ANTES DEL DESARROLLO DEL SISTEMA

El código que se muestra a continuación valida los escenarios escritos antes de que el sistema fuera desarrollado para mostrar como el sistema cumplirá con las funcionalidades requeridas.

```
# encoding: utf-8

from behave import given, when, then
from hamcrest import assert_that, equal_to

class LoginUser(object):
    def __init__(self, uname=None, pwd=None, button=None):
        self.uname = uname
        self.pwd = pwd
        self.button = button

    def login(self):
        assert self.uname is not None
        assert self.pwd is not None
        assert self.button is not None

        if self.button == 'iniciar_sesion':
            return 'Iniciar sesion'
        else:
            return 'Error al ingresar'

class Person(object):
    def __init__(self, type_doc=None, num_doc=None,
                 name=None, city=None, reason=None):
        self.type_doc = type_doc
        self.num_doc = num_doc
        self.name = name
        self.city = city
        self.reason = reason

    def register_person(self):
        assert self.type_doc is not None
        assert self.num_doc is not None
        assert self.name is not None
        assert self.city is not None
        assert self.reason is not None

        return 'Persona registrada'

class RegisterCar(object):
    def __init__(self, internal_number=None, parqueo_out=None,
                 parqueo_in=None, item_out=None, item_in=None,
                 km_out=None, km_in=None, escaleras_out=None, escaleras_in=None,
                 date_out=None, date_in=None, time_out=None, time_in=None):
        self.internal_number = internal_number
        self.parqueo_out = parqueo_out
        self.parqueo_in = parqueo_in
        self.item_out = item_out
        self.item_in = item_in
        self.km_out = km_out
        self.km_in = km_in
        self.escaleras_out = escaleras_out
        self.escaleras_in = escaleras_in
        self.date_out = date_out
        self.date_in = date_in
```

```

        self.time_out = time_out
        self.time_in = time_in
        self.reg_out = False
        self.reg_in = False

    def validar_salida(self):
        assert self.internal_number is not None
        assert self.parqueo_out is not None
        assert self.item_out is not None
        assert self.km_out is not None
        assert self.escaleras_out is not None
        assert self.date_out is not None
        assert self.time_out is not None

        self.reg_out = True

        return 'Registro guardado'

# #####
# Steps: Ingreso al sistema
# #####

@given(u'que he introducido mi nombre de usuario {username}')
def step_impl(context, username):
    context.username = username

@given(u'mi contraseña {password}')
def step_impl(context, password):
    context.password = password

@when(u'oprima el boton {button}')
def step_impl(context, button):
    context.button = button
    context.login = LoginUser(context.username,
                              context.password, context.button)

@then(u'debo ingresar al sistema')
def step_impl(context):
    assert_that('Iniciar sesion', equal_to(context.login.login()))

# #####
# Steps: Registro de visitas
# #####

@given(u'que ingresa una persona con documento {type_doc}')
def step_impl(context, type_doc):
    context.type_doc = type_doc
@given(u'con numero de documento {num_doc}')
def step_impl(context, num_doc):
    context.num_doc = num_doc

@given(u'con el siguiente nombre {name}')
def step_impl(context, name):
    context.name = name

@given(u'proviene de la ciudad {city}')
def step_impl(context, city):
    context.city = city

@given(u'con el siguiente motivo {reason}')
def step_impl(context, reason):
    context.reason = reason

@when(u'registre a la persona')
def step_impl(context):
    context.person = Person(context.type_doc,
                             context.num_doc,

```

```

        context.name,
        context.city,
        context.reason)

@when(u'el {num_doc} sea valido')
def step_impl(context, num_doc):
    assert_that(num_doc, equal_to(context.num_doc))

@then(u'registro a la persona {name} con documento {num_doc}')
```

```

def step_impl(context, name, num_doc):
    assert_that('Persona registrada',
                equal_to(context.person.register_person()))

# #####
# Steps: Registro de salida de vehiculos
# #####
```

```

@given(u'que sale el vehiculo con numero interno {internal_number}')
```

```

def step_impl(context, internal_number):
    context.internal_number = internal_number
@given(u'sale del parqueo {parqueo_out}')
```

```

def step_impl(context, parqueo_out):
    context.parqueo_out = parqueo_out

@given(u'con el conductor de salida con item {item_out}')
```

```

def step_impl(context, item_out):
    context.item_out = item_out

@given(u'con kilometraje de salida {km_out}')
```

```

def step_impl(context, km_out):
    context.km_out = km_out

@given(u'con las escaleras {escaleras_out} de salida')
```

```

def step_impl(context, escaleras_out):
    context.escaleras_out = escaleras_out

@given(u'con fecha {date_out} de salida')
```

```

def step_impl(context, date_out):
    context.date_out = date_out

@given(u'con hora {time_out} de salida')
```

```

def step_impl(context, time_out):
    context.time_out = time_out

@when(u'registre salida del vehiculo {internal_number}')
```

```

def step_impl(context, internal_number):
    context.register_car = RegisterCar(internal_number=context.internal_number,
                                       parqueo_out=context.parqueo_out,
                                       item_out=context.item_out,
                                       km_out=context.km_out,
                                       escaleras_out=context.escaleras_out,
                                       date_out=context.date_out,
                                       time_out=context.time_out)

    context.car_list = []
@when(u'los datos de salida sean validos')
```

```

def step_impl(context):
    assert_that('Registro guardado',
                equal_to(context.register_car.validar_salida()))

# #####
# Steps: Registro de retorno de vehiculos
# #####
```

```

@given(u'que retorna el vehiculo con numero interno {internal_number}')
```

```

def step_impl(context, internal_number):
    for c in context.car_list:
        if c.internal_number == internal_number:
```

```

        context.car_tmp = c

    @given(u'retorna al parqueo {parqueo_in}')
    def step_impl(context, parqueo_in):
        context.car_tmp.parqueo_in = parqueo_in

    @given(u'con el conductor de retorno con item {item_in}')
    def step_impl(context, item_in):
        context.car_tmp.item_in = item_in

    @given(u'con kilometraje de retorno {km_in}')
    def step_impl(context, km_in):
        context.car_tmp.km_in = km_in

    @given(u'con las escaleras {escaleras_in} de retorno')
    def step_impl(context, escaleras_in):
        context.car_tmp.escaleras_in = escaleras_in

    @given(u'con fecha {date_in} de retorno')
    def step_impl(context, date_in):
        context.car_tmp.date_in = date_in

    @given(u'con hora {time_in} de retorno')
    def step_impl(context, time_in):
        context.car_tmp.time_in = time_in
    @when(u'registre retorno del vehiculo {internal_number}')
    def step_impl(context, internal_number):
        assert False

    @when(u'los datos del kilometraje sean correctos')
    def step_impl(context):
        assert False

    @when(u'las escaleras de retorno sean las mismas')
    def step_impl(context):
        assert False

    @when(u'el conductor sea el mismo')
    def step_impl(context):
        assert False

    # #####
    # Steps para ambos: salida y retorno
    # #####

    @when(u'se validen los datos de salida')
    def step_impl(context):
        assert_that('Registro guardado',
                    equal_to(context.register_car.validar_salida()))

    @then(u'registro al vehiculo {internal_number}')
    def step_impl(context, internal_number):
        assert context.register_car.reg_out
        context.car_list.append(context.register_car)

```

Para que el código anterior funcione correctamente es necesario inicializar algunas variables en *environment.py*

```

import logging

def before_all(context):
    context.register_car = None
    context.parqueo_out = None
    context.km_out = None

```

```

context.escaleras_out = None
context.item_out = None
context.date_out = None
context.time_out = None
context.dict_cars = {}
context.warning = False

if not context.config.log_capture:
    logging.basicConfig(level=logging.DEBUG)

```

## B.2. STEPS - CON EL SISTEMA CONCLUIDO

Una vez concluido el Sistema usamos los mismos escenarios que se plantearon en el Capitulo 4 para las pruebas finales, con la diferencia de que ahora lo probaremos en el sistema.

```

# encoding: utf-8

from behave import given, when, then

# #####
# Steps: Para el ingreso al Sistema
# #####

@given(u'que ingreso al sistema con el url "{url}"')
def impl(context, url):
    br = context.browser
    br.visit(url)

@given(u'voy a la opcion "{option}"')
def impl(context, option):
    link = context.browser.find_link_by_text(option).first
    assert link
    link.click()
    # assert option in context.browser.html
    # msg = context.browser.find_link_by_partial_text(option).first
    # assert msg

@given(u'entro con mi nombre de usuario {usr}')
def impl(context, usr):
    context.browser.fill('identification', usr)
    context.user = usr
    assert usr

@given(u'mi contraseña {pwd}')
def impl(context, pwd):
    context.browser.fill('password', pwd)
    context.password = pwd
    assert pwd

@when(u'oprima el boton "{btn}"')
def impl(context, btn):
    button = context.browser.find_by_value(btn).first
    assert button
    button.click()

@then(u'ingreso al sistema y leo el mensaje "{msg}"')
def impl(context, msg):
    assert msg in context.browser.html

# #####
# Steps: Para el registro de Vehiculos
# #####

```



```

@given(u'que ingreso al sistema con el url "{url}")
def impl(context, url):
    br = context.browser
    br.visit(url)

@given(u'voy a la opcion "{option}")
def impl(context, option):
    link = context.browser.find_link_by_text(option).first
    assert link
    link.click()
    # assert option in context.browser.html
    # msg = context.browser.find_link_by_partial_text(option).first
    # assert msg

@given(u'entro con mi nombre de usuario {usr}')
def impl(context, usr):
    context.browser.fill('identification', usr)
    context.user = usr
    assert usr

@given(u'mi contraseña {pwd}')
def impl(context, pwd):
    context.browser.fill('password', pwd)
    context.password = pwd
    assert pwd

@when(u'oprima el boton "{btn}")
def impl(context, btn):
    button = context.browser.find_by_value(btn).first
    assert button
    button.click()

@then(u'ingreso al sistema y leo el mensaje "{msg}")
def impl(context, msg):
    assert msg in context.browser.html

@given(u'que vamos a la pagina de registro "{url}")
def impl(context, url):
    br = context.browser
    br.visit(url)

@given(u'registra el vehiculo con numero interno {car} esta de {status}')
def impl(context, car, status):
    context.browser.fill('car', car)
    context.car = car
    context.status = status

@given(u'sale/retorna del parqueo {parking}')
def impl(context, parking):
    context.parking = parking

@given(u'con el conductor con item {driver}')
def impl(context, driver):
    context.browser.fill('employee', '')
    context.browser.fill('employee', driver)
    context.driver = driver

@given(u'tiene el kilometraje {km}')
def impl(context, km):
    if context.status == 'salida':
        pass
    else:
        current_km = context.browser.find_by_name('km').first.value
        km = int(current_km) + 25
        context.browser.fill('km', '')
        context.browser.fill('km', km)

```

```

@given(u'con las escaleras {ladders}')
def impl(context, ladders):
    context.browser.fill('ladders', '')
    context.browser.fill('ladders', ladders)

@given(u'en fecha {date}')
def impl(context, date):
    link = context.browser.find_by_id('change_date').first
    assert link
    link.click()
    context.browser.fill('register_date', '')
    context.browser.fill('register_date', date)

@given(u'en horas {time}')
def impl(context, time):
    context.browser.fill('time', '')
    context.browser.fill('time', time)

@when(u'registre al vehiculo {car} con su estado de {status}')
def impl(context, car, status):
    button = context.browser.find_by_value("Registrar").first
    assert button
    button.click()

@when(u'lea el mensaje "{msg}"')
def impl(context, msg):
    assert msg in context.browser.html

@then(u'guardo el registro del vehiculo {car} correctamente')
def impl(context, car):
    pass

```

Para que este código funcione correctamente es necesario también inicializar algunas variables en *environment.py*

```

from splinter.browser import Browser

def before_all(context):
    context.browser = Browser()

def after_all(context):
    context.browser.quit()
    context.browser = None

```



# Manual de Usuario

# Bibliografía

Richard Jones Benno Rice and Jens Engel. Behave, Diciembre 2014. URL <https://pythonhosted.org/behave/>.

CobraTeam. Splinter, Diciembre 2014. URL <http://splinter.cobrateam.info/en/latest/>.

Django Software Foundation. The web framework for perfectionists with deadlines, Diciembre 2014. URL <https://www.djangoproject.com/>.

John Ferguson Smart. *BDD in Action*. Manning Publications, 2014.

Rachel Willmer. A django testrunner for the behave bdd module, Diciembre 2014. URL <https://github.com/django-behave/django-behave>.