

---

# **django-spese Documentation**

*Release 0.1*

**Idfa**

**Apr 16, 2017**



<b>1</b>	<b>Contents:</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	User Guide . . . . .	5
1.3	Veicoli . . . . .	14
1.4	Changelog . . . . .	18
1.5	License . . . . .	19
<b>2</b>	<b>Indices and tables</b>	<b>21</b>



This is django-spese's documentation.

django-spese is a simple personal expenses register. It is an application for Django.

The source code of this project is hosted on [github.com](https://github.com).



---

**Contents:**

---

## Installation

To install *django-spese*, you'll need some prerequisite:

- first of all you must know how type commands in console mode in your system;
- then, it's important you have some knowledge about how create/configure a Django project and app(s);
- and, last but not least, you'll need to have a copy of **Python** version 3.5, or newer, already installed in your system.

Futhermore I take for granted you know how to use virtualenv and you'll use it to create a Django project to test this app.

Hereafter I describe how install from a distribution. You can download it from [this tar.gz](#) , or from [this zip archive](#) if you prefer a zipped version.

---

**Note:** If you are a developer, maybe you'd like to clone from the [project source repository](#) using [git](#) as version control software.

---

## Creating a Django project to host the application

If you have your project to host *django-spese*, use it, and go to *Configuring the project*.

Otherwise create a base project using virtualenv as follows:

```
> mkdir progetto_servizi
> cd progetto_servizi
> virtualenv env                                (this loads a copy of the system's s_
↳python)
> source env/bin/activate                       (or, in Windows, _
↳env\Scripts\activate)
> pip install path/to/django-spese-0.1.tar.gz  (this loads django-spese and its
↳taggit, ...)                                dependencies: django, django-
```

then and create the django project:

```
> django-admin startproject servizi
```

## Configuring the project

Add *django-spese* and *taggit* to your `INSTALLED_APPS` in `setting.py`. Like this:

```
INSTALLED_APPS = [  
    ...  
    'spese',  
    'taggit',  
]
```

And, again in `settings.py` double check the presence of:

- `django.contrib.sessions.middleware.SessionMiddleware` and
- `django.contrib.messages.middleware.MessageMiddleware`

in `MIDDLEWARE_CLASSES = [ ... ]`

Include the *django-spese* `URLconf` in your project `urls.py`. Like this:

```
from django.conf.urls import include  
...  
url(r'^spese/', include('spese.urls', namespace='spese')),
```

Provide the django login machinery in your project: that is a `template/login.html` template, in your project `url.py` add:

```
from django.contrib.auth import views as auth_views  
...  
url('^login/$', auth_views.login, {'template_name': 'login.html',}, name='login'),  
url(r'^logout/$', auth_views.logout, {'next_page': '/login'}, name='logout'),
```

and in `setting.py` add:

```
LOGIN_REDIRECT_URL = '/' # It means home view
```

You can copy a `login.html` example from `.../env/Lib/site-packages/spese/templates/example/*` (in windows use backslashes)

Provide a `template/base.html` template in your project. In `base.html` the block content marks where *django-spese* is going to write its contents:

```
{% block content %}  
{% endblock %}
```

You can copy a `base.html` examples from `.../env/Lib/site-packages/spese/templates/example/*` (in windows use backslashes)

## Creating the database

Run `python manage.py migrate` to create the *django-spese* models and adding a minimal dataset: *user1*, *user2*, *transfer\_funds* tag, and *cache* source for *user1* and *user2*.

Run `python manage.py createsuperuser` to create a *superuser*.

Start the development server (`python manage.py runserver`)



## Refining the database contents

Visit <http://127.0.0.1:8000/admin/> .

Login as *superuser* to add/change/delete DB base items: sources, tags, users (... and expenses. But furnish a user interface to accomplish this task is one target of spese app).

---

**Note:** you'll need the *Admin* app enabled

---

---

**Note:** As superuser, at least, reset the users *user1* and *user2* passwords at known values.

It might be a good idea to change the user names to something more meaningful too.

---

## Enjoi

Visit <http://127.0.0.1:8000/spese/> , login as a user and enjoi the app.

## User Guide

Here we are going to talk about how use *django-spese* to register our personal expenses.

To understand better the available operations, we need know the base concepts underlining this application. So first of all we'll talk of *General concepts*.

Then we'll see the *Administration* operations.

And finally we'll see the details about *Using it* , where *it* means the application, of course.

Let's start.

### General concepts

Well, we said *personal* and *expense*. Two terms to think about.

Thinking to *personal* we get the **first concept**: every expense *own* someone.

Who is this *one*? The logged user.

Hence, to use *django-spese* we need login using a `username` and a `password`. When we do it, the application will bind every new expense to us.

Follow the **second concept**: I, logged user, can add and work on *my* expenses; but I cannot do something on somebody's else expenses. Even I cannot see them.

Clarified the *personal* term implications, let's see the *expense* term.

An expense means *money* to buy something. Where is money from?

Here we are at the **third concept**: in *django-spese* a source of money is called (bank) *account*.

An *account* could be a wallet, or a debit card, or a bank account or something else. Every expense is bound to an *account* from where money is kept to fulfil it.

Be aware do not confuse this account concept with idea of user. In *django-spese* when we speak about *account* we mean bank account, **not** user account.

As opposed from everyday world, here an *account* hasn't limit: we can draw from it how much money as we wish. Or add to it.

Add? Yes, add. Why we could not add money instead of remove it? **Forth concept:** write positive numbers to add money to an *account*, and negative numbers to subtract money from it. *django-money* don't know how to subtract money, it knows only how to add it :)

**Fifth concept:** an *account* is bound to one, or more, *user(s)*.

This means that every *user* could have one or more *account(s)*. But is also possible an *account* could be **shared** between two or more *users*. (Yes, I know. This is a very strange concept. Only the few married men can understand it :-)

**Sixth concept:** every expense could be classified using one or more *tag(s)*.

For example, maybe I wish classify my holidays expenses using the tag *freetime*, and the work expenses assigning them the *work* tag.

A last concept and we're done. **Seventh concept:** it is possible transfer money from one *account* to another. We call this kind of operation: *transfer funds*.

Using *transfer funds* we can save time. We could subtract (remember: use a negative number) from an *account* and add the same (positive) number to another *account*. But it's a waste of time and it's even a bit error prone: it isn't so difficult write one of the two numbers with one more, or less, digit.

And, without *transfer funds*, if we need to change an amount, we must remember to change it in two (unrelated) expenses. *transfer funds* links the two expenses, and if we change one, *django-spese* will change accordingly the other.

Since a picture is worth a thousand words, to summarize, please look at the figure below, which sum up the relationships between the exposed concepts.

As we can see, the most complex relations are between *account* and *user*, and between *tag* and *expense*. Technically speaking, these are m:n relationships. Every *account* could belong to more *users* and vice versa. Similarly for *tag* and *expense*.

The relationships *user / expense* and *source /expense* are much simpler: one *user* own more *expenses*, not viceversa. So between *source* and *expense*.

Oh, I haven't quoted the *work cost type*. Have I? Please, be patient. Over the seventh I tend to confuse between ordinals. So ... I explain it now. In case you are a professional, may be you wish to register work's expenses. If so, you can classify this kind of costs assigning an appropriate type. *WCTYPE* means Work Cost Type and register these types. And *PercentDeduction* register what percent can be deduct from income tax calculation about a type.

In case you are asking yourself why use records to register a percent deduction associated to a work cost type, the answer is: time. The percentage can vary on time passing, and we can register different values on different time intervals. Yes, incoming tax calculation isn't a simple matter in Italy.

Oh! I was forgetting. In case of transfer funds, you cannot assign work cost type to the operation. Work cost type is worth about money flow to/from the extern boundary of our accounts. Not about internal movements between accounts.

Ok. Now our global knowledge about *django-spese* is complete. We can start play around.

## Administration

*django-spese* administrators are in charge to supply a simple but complete environment to application users.

To accomplish this target you use the URI <http://127.0.0.1:8000/admin/> and login using the administrator's username and password.

Base administration window looks as below, without the ellipses:

Red ellipses show the most interesting entities for our duties:

- *user*,
- *account*,
- *work cost type*,
- *percent deduction*,
- and *tag*.

Green ellipses show the shortcuts to add and change the target records.

User interface is immediate, so I don't explain it in detail. Only, I wish underline two points.

First. It isn't possible to know an existing user password. We can only reset it to a known value. To do so, select *user* entity, from the next users list select the desired user, and then use link underlined with red ellipse in the picture below.

Second. When working on *accounts*, remember: an account could be shared between different *users*. So the pertinent window looks like below:

Here red ellipse remember us the possibility to bind a single account to more users.

It's important bind the account to the correct user, and be aware to share accounts that are truly shared between different users. I.e. let's to keep again the previous picture. There we have *wallet* shared between *user1* and *user2*. This means that this two users have the **same wallet**: see it as a shared wallet!

If I wish model a situation where *user1* and *user2* have different wallets, I must create them (for sake of example let's say *wallet1* and *wallet2*) and assign either of them to a single user (to complete the example: bind *wallet1* to *user1* and *wallet2* to *user2*).

A last word about *tags*. These are the folders used to classify our expense. So I urge you to create a tag set limited in size, that fit well with your necessities. It's a difficult matter change classification criterions while running :)

## Using it

And now, provided of user's username and password, finally we can login visiting <http://127.0.0.1:8000/spese/>.

Home page welcome us, showing the list of our expenses:

Hereafter we call this window as *home*, even if the true home is the project home. But we need to focus on our application, so we call *home* this one.

In the previous figure we highlight the presence of two different menus: the project menu and the the *django-spese* menu.

Project menu depends on your project. The previous figure shows the one you obtain from the *django-spese* repository. It's there only as a demo to host our application.

We are concerned about the *django-spese* menu. At *home* we have this voices:

- *add* adding us a new expense;
- *transfer funds* to realize a transfer of money from one source to another;

- *reports* to show a summary about our tracked accounts and tags;
- *→csv* to export the showed expenses list to a csv file.

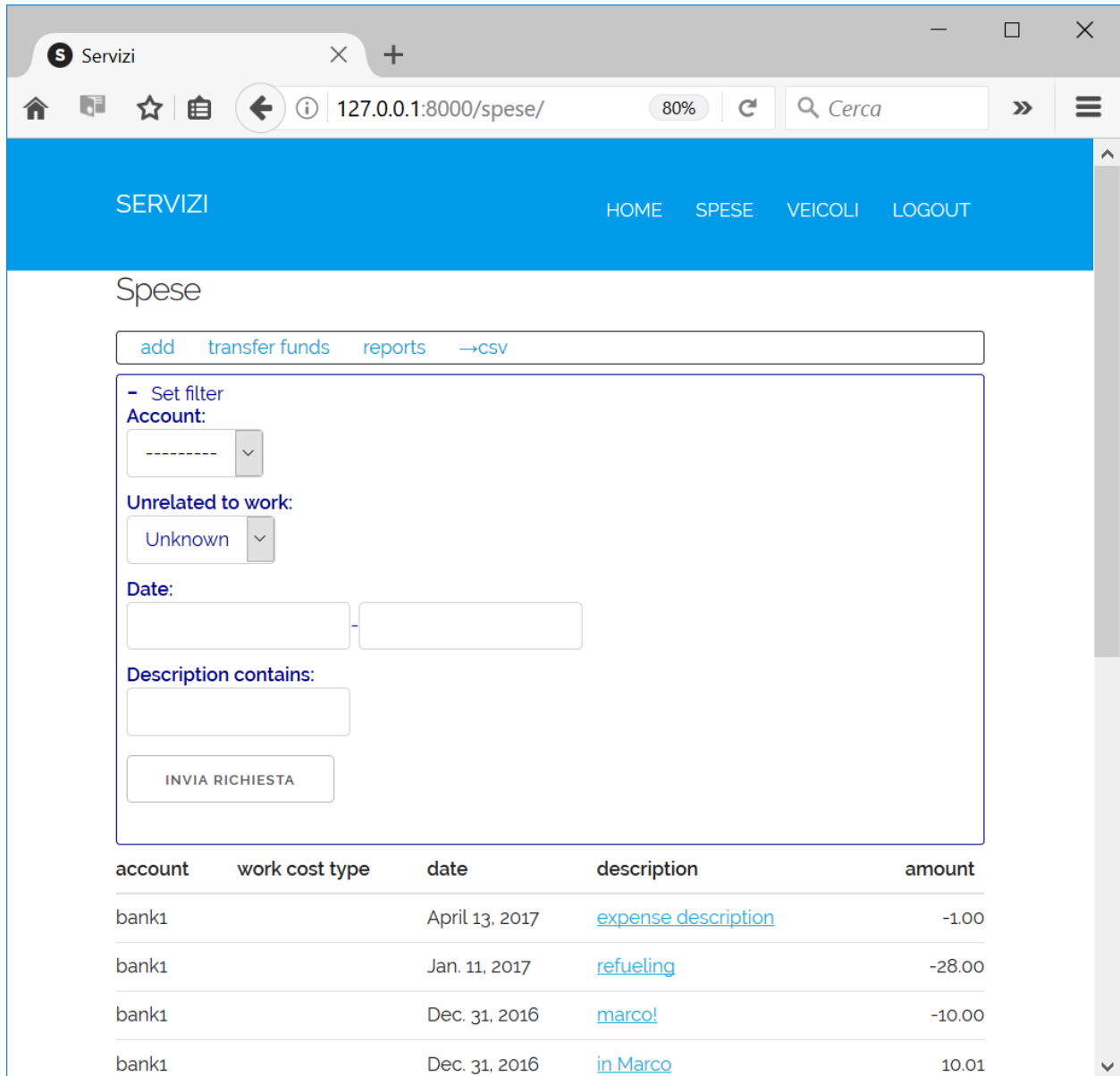
If we click on an expense description, we'll get its detail:

and from there we can:

- *add* to insert a new expense (this is the same menu voice from home);
- *change* to edit the expense characteristics;
- *toggle* to change the amount sign of the expense;
- *delete* to remove the expense.

In case of a transfer between funds, its details view show us even a link to the associated transfer. In next picture this link is highlighted by the green oval.

If we wish to focus on a limited expenses list, we can use the *filter section*, located just below the *django-spese* menu. If we click on + *Set filter* the application will expand the filter section:



The screenshot shows a web browser window with the URL `127.0.0.1:8000/spese/`. The page title is "SERVIZI" and the navigation menu includes "HOME", "SPESE", "VEICOLI", and "LOGOUT". The main content area is titled "Spese" and contains a filter section with the following fields:

- Buttons: `add`, `transfer funds`, `reports`, `→csv`
- Account: A dropdown menu with a dashed line as a placeholder.
- Unrelated to work: A dropdown menu with "Unknown" selected.
- Date: Two adjacent date input fields.
- Description contains: A text input field.
- Button: `INVIA RICHIESTA`

Below the filter section is a table of expenses:

account	work cost type	date	description	amount
bank1		April 13, 2017	<a href="#">expense description</a>	-1.00
bank1		Jan. 11, 2017	<a href="#">refueling</a>	-28.00
bank1		Dec. 31, 2016	<a href="#">marco!</a>	-10.00
bank1		Dec. 31, 2016	<a href="#">in Marco</a>	10.01

In this section we can choose to select our expenses:

- by *account*, selecting it using the first selection list;
- by relation with work, using the second selection list, titled *unrelated to work*; this list has three choices: *unknown* selects expenses of every type, *yes* selects only expenses with *work cost type* value unset, and *no* selects only expenses with a valid work cost type value set;
- by a date interval, using the two date fields;
- and, finally, by a piece of description, writing it in the last field of the section.

Filtering, all choosed values are used contemporary. I.e. if we choose account and description, this is exactly what we obtain: a list of expenses that satisfy both the criterions.

After we have compiled the filter fields, we need to click on the *transmit request* button to make effective our selections.

Hereafter I depict an example of use of filter, where we request to show only expenses:

- about *cache* account,
- **and** *not* related with work,
- **and** occurred after 1st october 2016,
- **and** with slice of word *sp* in description.

The screenshot shows a web browser window with the URL `127.0.0.1:8000/spese/?accou`. The page title is "SERVIZI" and the navigation menu includes "HOME", "SPESE", "VEICOLI", and "LOGOUT". The main content area is titled "Spese" and contains a filter section and a table of expenses.

The filter section is titled "Set filter" and includes the following options:

- Account:** cache
- Unrelated to work:** No
- Date:** 2016-10-01
- Description contains:** sp

Below the filter section is a button labeled "INVIA RICHIESTA".

The table below shows the results of the filter:

account	work cost type	date	description	amount
cache	auto/telef.	Nov. 2, 2016	<a href="#">spesa 2</a>	-20.00

The footer of the page contains links for "ABOUT", "CONTACT", "PRIVACY POLICY", and "TERMS", along with a copyright notice: "© 2016, 2017. SERVIZI. ALL RIGHTS RESERVED. 51D507A".

Be aware that clicking `+ Set filter` and `- Set filter` we open, or close, the filter section. **But** values we set in it are valid anyway. To reset filter (by now) we must clear every value in it manually.

### Adding expenses

At *home*, selecting the *django-spese* menu voice *add* we obtain a form to input an entirely new expense:

Here we can select the desired *account*, write the date, description and amount. And we can choose between the shown tags to categorize our expense.

When we are done, we can save and return to *home*, using the *save* button. Or we can save and add again a new expense, using the *save & continue* button.

Using *save & continue* give us the current form already completed with the previous fields values. This is comfortable in case of entry of more expenses regarding same account, close dates, same tags ...

To leave the form without creating a new expense, simply use the browser's *back* button, or visit the *home* URL.

## Changing expense

As we said, at *home*, selecting an expense, we obtain the expense *detail* window.

Here we can choose the *change* menu voice getting a form to change the expense characteristics:

In this form we can change whatever we wish. To save changes, we must push the *save* button, moving us to detail again. Or we can choose the *save & continue* button, that keep us on the current change form: just in case we wish change more fields values but one at time.

## Toggling expense

From the *detail* window, we can choose the *toggle* menu voice.

If we hit this voice, the application change the sign of the amount of the displayed expense.

This action is immediatly shown. We can observe:

- a (hopefully) confirming message after the spese's menu area;
- the new amount of the expense, with the same quantity, but opposite sign.

## Deleting expense

In the expense *detail* window there is another, very dangerous, menu voice: *delete*.

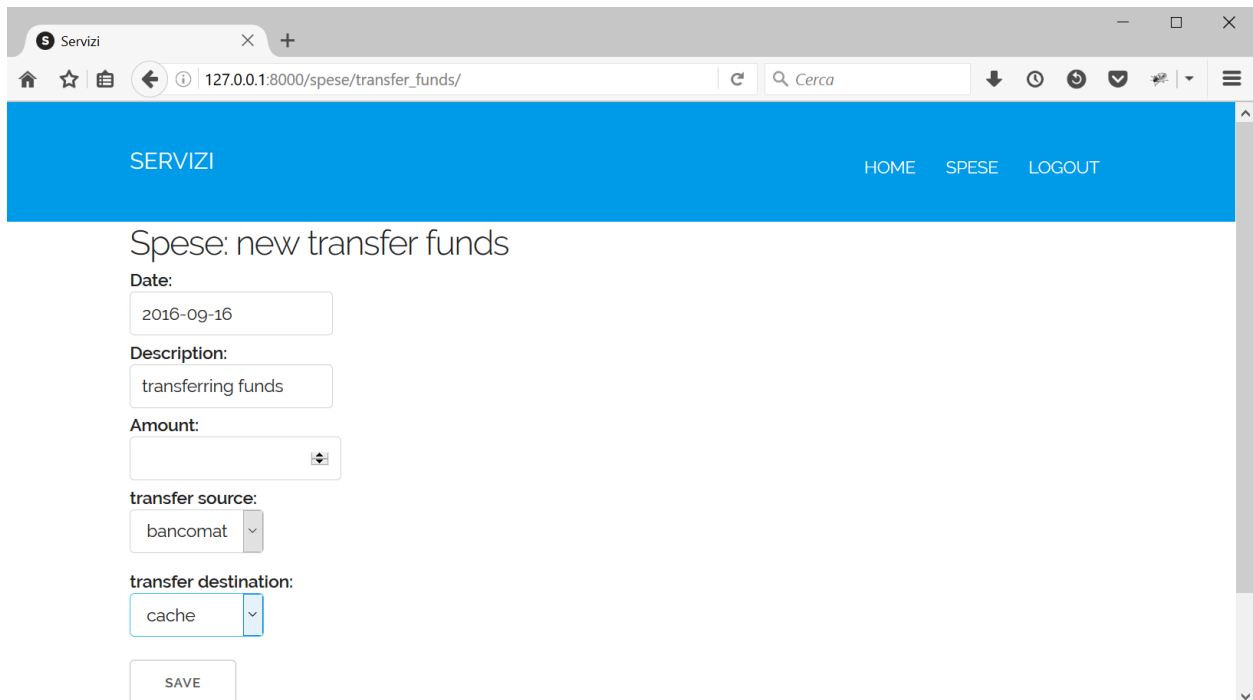
As we can imagine, this voice remove the showed expense.

**Warning:** What you *don't know* is that, by now, this operation **don't ask for confirmation**.

If we choose to delete the current expense this is what we *immediatly* obtain.

## Transfer funds

At *home*, selecting the *django-spese* menu voice *transfer funds* we obtain the shortcut to transfer money from an *account* to another:



The screenshot shows a web browser window with the URL `127.0.0.1:8000/spese/transfer_funds/`. The page title is "SERVIZI" and the navigation menu includes "HOME", "SPESE", and "LOGOUT". The main content area is titled "Spese: new transfer funds" and contains the following form fields:

- Date:**
- Description:**
- Amount:**
- transfer source:**
- transfer destination:**
- SAVE** button

As usual: we must compile the form with the appropriate values. Then choosing the *save* button we get the desired operation: the *amount* is subtracted from *transfer source* and added to *transfer destination*.

## Reports

At *home*, selecting the *django-spese* menu voice *reports* we obtain a summary about the current situation of the observed accounts, tags, and work cost types:



## Spese: Reports

account	ext-in	int-in	int-out	ext-out	balance
bank1	100.00	0	-39.50	-45.00	15.50
cache	0	39.50	0	-30.50	9.00
<i>totals</i>	<i>100.00</i>			<i>-75.50</i>	<i>24.50</i>

tag	ext-in	int-in	int-out	ext-out	balance
without tags	100.00	39.50	-39.50	-45.50	54.50
vacanze	0	0	0	0	0
marco	0	0	0	0	0
<i>totals</i>	<i>100.00</i>			<i>-45.50</i>	<i>54.50</i>

work cost type		in	out	balance
auto/telef.		0	-20.00	-20.00
contributi		0	0	0
<i>totals</i>		<i>0</i>	<i>-20.00</i>	<i>-20.00</i>

Just a note. Remember that *reports* works using the filter set in *spese* home. So reports are homogeneous with expenses list.

Now, we spend a word about the overall layout of these reports.

First of all: the window has three tables:

- about *accounts*;
- about *tags*;
- and about *work cost types*.

Every row of *accounts* and *tags* is divided in six columns. From left to right:

- the item name;
- the sum of the income *from the extern*, i.e. not from transfer funds operations;
- the sum of *internal* income, i.e. from transfer funds operations;
- the sum of *internal* outcome, i.e. transfer funds operations to other accounts;
- the sum of outcome *to the extern*, i.e. not transfer funds to other accounts;
- and finally the sum of the previous quantities: the balance of the row.

In case of work cost types, are missing the columns about transfer funds because here we observe money flow from/to our system boundaries.

The last row of every table shows the relative column grand total, with the exceptions of the transfer funds columns that aren't calculated.

→**csv**

Clicking this menu voice we export the showed expenses list to a file named *expense\_list.csv* located in the download directory of our web browser.

## Veicoli

From version 0.4+ *django-spese* has an ancillary app: *django-veicoli*.

This is a simple extension to expenses database to record data specific to vehicles expenses, such:

- what vehicle is about;
- what type of expense (refuelling, maintenance, ...);
- at what distance (km or mi) it happened;
- a unit cost (regarding refuelling, eventually needed to calculate unit consumption).

## Installation

When you install *django-spese*, you'll get *django-veicoli* installed too.

So, if you need it, you have only to configure it.

In `setting.py` of your project, add *django-veicoli* to your `INSTALLED_APPS`. Like this:

```
INSTALLED_APPS = [  
    ...  
    'veicoli',  
]
```

Then, include the *django-veicoli* URLconf in your project `urls.py`. Like this:

```
from django.conf.urls import include  
...  
url(r'^veicoli/', include('veicoli.urls', namespace='veicoli')),
```

With *django-veicoli* set in your project, when you are going to create the database you'll get even the *django-veicoli*'s database.

## General concepts

Here we must consider that *django-veicoli* is a *django-spese*'s extension.

So the vehicle's expense will be registered in *django-spese* database.


Because now we are senior database analyst :-), we start directly to show the *django-veicoli* schema:

We observe:

- every event links to one and only one expense; in expense there is the description, the amount and the other characteristics of the expense we are already accustomed to;
- and every event links to a vehicle and to a type (VEvent).

## Administration

If we have correctly configured our project to use *django-veicoli*, when we login to the URI <http://127.0.0.1:8000/admin/> using the administrator's username and password, the administrator's home will show the *django-veicoli* tables too:

VEICOLI		
Events	+ Add	 Change
VEvents	+ Add	 Change
VTypes	+ Add	 Change
Vehicles	+ Add	 Change

From where to fill:

- *VTypes*, for example: car, motorcycle, ...;
- *VEvents*, let's say: gasoline, maintenance, administrative, ...;
- *Vehicles*: the mythical 313, and so on.

## Using it

Visiting <http://127.0.0.1:8000/veicoli/> we get:

that is vehicle's home page.

It lists all events, regarding the logged user, about vehicles. Here we can:

- to add a new event using the *add* voice of vehicle menu;
- and to show details about an event with a click on its description.

## Adding event

At vehicle home, selecting the *django-veicoli* menu voice *add* we obtain a form to input a new event:

## Veicoli: new event

Vehicle:

313 

Event type:

benzina 

Km:

11 

Unit\_cost:

1 

Account:

bank1 

Work cost type:

auto/telef. 

Date:

2016-12-11

Description:

gasoline refuelling

Amount:

-56 

Type of expense:

 vacanze  marco

---

This form has two sections.

The first section permits us to select the vehicle, the event type, the distance (labeled as Kilometer; use the distance unit you prefer, but be homogeneous) and the unit cost of the following expense amount.

The second section is about the usual characteristics of an expense.

In fact it is the same windows used to register a new expense.

And even the behaviour of the *save* and *save & continue* buttons are the same.

As usual, to leave the form without creating a new event, use the browser's *back* button, or visit the *home* URL.

## Displaying event

From *home*, clicking on an event description, we obtain its *detail* window.

---

SERVIZI HOME SPESE VEICOLI LOGOUT

Veicoli: show event detail

[add](#) [change](#)

Vehicle: 313

Event type: rifornimento di benzina

km: 11

unit cost: 1.500

---

Payed by: cache

on: Dec. 4, 2016

rifornimento di benzina chg

amount: -10.50

ABOUT CONTACT PRIVACY POLICY TERMS© 2016. SERVIZI. ALL RIGHTS RESERVED.

Here we have two menu voices:

- *new* to create a new event; this is similar to the same voice in the vehicle *home* menu;
- *change* to modify the current event.

### Changing event

If we choose the previous *change* menu voice, we get a form to change the event characteristics:

## Veicoli: edit expense detail

Vehicle:

Event type:

Km:

Unit\_cost:

Account:

Work cost type:

Date:

Description:

Amount:

Type of expense:

 vacanze  marco 

As in case of new event this form has two sections: the first one specific for vehicle events, and the second for a whatever expense.

In this form we can change what we wish.

As in case of changing expenses, to save changes, we must push the *save* button, moving us to detail again. Or we can choose the *save & continue* button, that keep us on the current change form.

## Changelog

### Version 0.5

- fixed bug #3;
- using django-filter to select on expenses list by *account*, *date*, *work\_cost\_type* presence and partial *description*;
- exporting expenses list to csv file;
- in transfer fund detail, added reference to other expense;

- user manual updated;
- added version in status bar.

## Version 0.4

- Documents updated;
- Added *Work Cost Types*;
- Now tranfer funds are related;
- Added a first version of expense reports (by account, tags, work cost type);
- Added a first version of *django-veicoli* app;
- In case of operation regarding more tables, using explicit transaction management.

## Version 0.3

- Added expense amount toggle functionality
- Added log facility

## Version 0.2

- Added documentation

## Version 0.1

- Initial release.
- WEB User interface to list, add, change, delete expenses.

## License

Django-spese is licensed under the MIT License:

Copyright (c) 2016 luciano de falco alfano

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.





---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`