Merlot Documentation

Release 0.2

Merlot Documentation Team

March 18, 2012

CONTENTS

Merlot is a web-based project management software written in Python using the Grok web development framework.

Merlot is under development, but it currently allows you to:

- Create projects and organize projects in tasks
- Associate projects to clients
- Manage projects and tasks statuses
- Track the time worked in tasks and generate reports based on time logs
- Have a set of favorite tasks on a per user basis
- Quickly log time in a task via the *quick log feature*

You can follow the *Installing Merlot* guide to get your own Merlot instance up and running.

The project is hosted in Google Code at: http://code.google.com/p/merlot/. There you will find screenshots, links to the issue tracker, mailing list and other resources.

CHAPTER

ONE

CONTENTS

1.1 Installing Merlot

This document will guide you through the steps of installing Merlot. This guide was written for an Ubuntu 10.10 system, there may be some differences with other systems.

1.1.1 Install the required system packages

Before even getting the source code, we need to make sure you have all the system level dependencies installed. The following command will take care of it:

```
$ sudo apt-get install mercurial python-virtualenv python-dev libxslt-dev libxml2-dev python-pip
```

1.1.2 Create and build the buildout

We are approaching a buildout-based installation.

The first thing you need to do is to install the MerlotTemplates package, which provides a PasteScript template to create a buildout that sets up Merlot:

```
$ pip install MerlotTemplates
```

Now you are ready to create the buildout:

```
$ paster create -t merlot_buildout merlot
```

Provide the Merlot version to be used when the question is prompted. Then create a virtual environment inside the buildout directory that you've just generated:

```
$ cd merlot
$ virtualenv --python=/usr/bin/python2.6 --no-site-packages .
$ source bin/activate
```

Now you can run buildout:

```
$ python bootstrap.py
$ buildout
```

And you are ready to start Merlot:

```
$ merlot fg
```

This will start the server on port 8080 with basic authentication in front.

1.1.3 First steps with Merlot

Once the server is up, point your browser to http://localhost:8080/ and authenticate using *admin* for both user name and password. This will take you to an administration screen. There you can create a Merlot application.

The first thing you will want to do is to add *users* and *clients*. Projects will be later associated to clients.

Once you've added at least one client, you can proceed to add a *project*, that is the main concept of Merlot.

1.2 Getting started with development

This document will guide you through the steps of getting a development environment up and running. This guide was written for an Ubuntu 10.10 system, there may be some differences with other systems.

1.2.1 Install the required system packages

Before even getting the source code, we need to make sure you have all the system level dependencies installed. The following command will take care of it:

```
$ sudo apt-get install mercurial python-virtualenv python-dev libxslt-dev libxml2-dev
```

1.2.2 Get the source and build the development environment

The first thing you need to do is to clone the Mercurial repository:

```
$ hg clone https://merlot.googlecode.com/hg/ merlot
```

Then change to the working directory, create a virtualenv right there and activate it:

```
$ cd merlot
$ virtualenv --python=/usr/bin/python2.6 --no-site-packages .
$ source bin/activate
```

Now we build the *buildout*:

```
$ python bootstrap.py
$ buildout
```

This usually takes a while, as the buildout command downloads all the requirements for Merlot.

Finally, we can start the server, which will run in port 8080:

```
$ merlot fg
```

1.2.3 First steps to play with the system

Once the server is up, point your browser to http://localhost:8080/ and authenticate using *admin:admin* to access the Grok administration screen. There you can create a Merlot application and start playing with it.

The first thing you will want to do is to add *users* and *clients*. Projects will be later associated to clients.

Once you've added at least one client, you can proceed to add a *project*, which is what this is all about.

1.2.4 Run the automated tests

To run Merlot automated tests, just run the following command from the working directory:

```
$ bin/test
```

You can generate a coverage report with the following command:

```
$ bin/test -s merlot --coverage=coverage
```

The report files will be generated in HTML format in the *coverage* directory.

1.2.5 Build the documentation

To build the Merlot documentation, just run the following command from the working directory:

```
$ bin/sphinx-build docs-source/ docs
```

The documentation will be created in the *docs* directory in HTML format.

1.3 Automated tests in doctest style

The application is covered by tests that are written using the *doctest* style. In this section you will find all those tests.

1.3.1 Setup

The *app* variable is binded in out tests setup to a brand new created Merlot object. The Merlot object has a title:

```
>>> app.title
u'Merlot \u2014 Project Management Software'
```

When the application is created, containers for projects, clients and users get also created:

```
>>> 'projects' in app
True
>>> 'clients' in app
True
>>> 'users' in app
True
>>> project_container = app['projects']
>>> client_container = app['clients']
>>> userfolder = app['users']
```

Let's check that *projects_container* and *clients_container* are what we expect them to be:

```
>>> project_container.title
u'Projects'
>>> client_container.title
u'Clients'
>>> from merlot.interfaces import IProjectContainer, IClientContainer
>>> IProjectContainer.providedBy(project_container)
True
>>> IClientContainer.providedBy(client_container)
True
```

There are a few local utilities registered in the Merlot application. We have local utilities to manage users and authentication:

```
>>> from zope.component import getUtility
>>> from zope.app.authentication.interfaces import IAuthenticatorPlugin
>>> from zope.app.security.interfaces import IAuthentication
>>> users = getUtility(IAuthenticatorPlugin, 'users', context=app)
>>> auth = getUtility(IAuthentication, context=app)
```

Also, an integer IDs local utility is registered. This allow us to create an integer ID for each object and later look up objects by their IDs:

```
>>> from zope.intid.interfaces import IIntIds
>>> intids = getUtility(IIntIds, name='intids', context=app)
```

The last local utility registered is a relation catalog utility that allow us to keep track of relations between objects:

```
>>> from zc.relation.interfaces import ICatalog
>>> rel_catalog = getUtility(ICatalog, context=app)
```

A global utility is registered to resolve to an object given a path and vice versa:

```
>>> from z3c.objpath.interfaces import IObjectPath
>>> object_path = getUtility(IObjectPath)
>>> object_path.path(app['projects'])
u'/app/projects'
>>> object_path.resolve(u'/app/projects') == app['projects']
True
```

1.3.2 Starred Tasks

Starred tasks are tasks that the user selects to appear allways on top on his or her dashboard. They are implemented as annotations on the user account model. That is, the objects that store the users information (the model implements merlot.interfaces.IAccount). The annotations are managed via an adapter.

We need to create an account object:

```
>>> from merlot.auth import Account
>>> user_account = Account('testuser', 'secret', u'Test User')
```

Now we can adapt the account object and manipulate a list of integer values which are intended to be tasks integer IDs that you can get by using the integer IDs local utility:

```
>>> from merlot.interfaces import IStarredTasks
>>> starred_tasks = IStarredTasks(user_account)
>>> starred_tasks.getStarredTasks()
[]
>>> starred_tasks.addStarredTask(23)
>>> starred_tasks.addStarredTask(44523)
>>> starred_tasks.getStarredTasks()
[23, 44523]
```

And we can delete items:

```
>>> starred_tasks.removeStarredTask(44523)
>>> starred_tasks.getStarredTasks()
[23]
```

If we try to delete an item that doesn't exist, nothing happens:

```
>>> starred_tasks.removeStarredTask(344)
>>> starred_tasks.getStarredTasks()
[23]
```

1.3.3 Catalog

We create a project:

```
>>> from datetime import datetime
>>> from merlot.project import Project
>>> project = Project()
>>> project.title = u'Testing'
>>> project.start_date = datetime(2010, 10, 10).date()
>>> project.id = 'test'
>>> app['test'] = project
```

We make a catalog query to find the project we've just added:

```
>>> from zope.component import getUtility
>>> from zope.catalog.interfaces import ICatalog
>>> catalog = getUtility(ICatalog, context=app)
>>> result = catalog.searchResults(title='Testing')
>>> len(result)
1
>>> list(result)[0].title
u'Testing'
>>> list(result)[0].start_date
datetime.date(2010, 10, 10)
```

Let's test the *project* and *task* indexes. To do so, let's create a task a log inside that task:

```
>>> from merlot.project import Task, Log
>>> task = Task()
>>> task.title = 'A sample task'
>>> task.description = ''
>>> task.id = 'a-sample-task'
>>> app['test'][task.id] = task
>>> log = Log()
>>> log.description = 'Bla, bla'
>>> app['test'][task.id]['1'] = log
```

Let's see what's in the *project* and *task* indexes for the log we've just created according to the catalog:

```
>>> result = catalog.searchResults(description='bla')
>>> len(result)
1
>>> logi = list(result)[0]
>>> logi.description
'Bla, bla'
```

Well, let's see what's the intid associated with our project and task:

```
>>> from zope.intid.interfaces import IIntIds
>>> intids = getUtility(IIntIds, name='intids', context=app)
>>> project_id = intids.getId(app['test'])
>>> task_id = intids.getId(app['test']['a-sample-task'])
```

And finally we can compare the actual intids with the values indexed by the catalog:

```
>>> project_id == logi.project()
True
>>> task_id == logi.task()
True
```

1.3.4 Users, Authentication & Authorization Functional Tests

In Merlot there are only two roles: authenticated and anonymous. If you are authenticated you have full access to the system; if you are not logged in, you have no permissions.

We need to create a user to play with later. First of all we access the site as admin:

```
>>> from zope.app.wsgi.testlayer import Browser
>>> browser = Browser()
>>> browser.handleErrors = False
>>> browser.addHeader('Authorization', 'Basic admin:admin')
>>> browser.open('http://localhost/app')
>>> 'Logged in as: Manager' in browser.contents
True
```

Now we create a new user. To do so, we click on the *Users* tab and then on the *Add new User* link:

```
>>> browser.getLink('Users').click()
>>> 'There are currently no users.' in browser.contents
True
>>> browser.getLink('Add new User').click()
```

We fill the *add user form*:

```
>>> browser.getControl(name="form.id").value = u'user'
>>> browser.getControl(name="form.real_name").value = u'Testing User'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl(name="form.confirm_password").value = u'secret'
```

Submit the form and check that the changes were saved:

```
>>> browser.getControl("Add user").click()
>>> 'User added' in browser.contents
True
>>> 'Testing User' in browser.contents
True
```

We are now ready to start testing permissions. First of all, let's log out from the site to test that we can't access any pages:

```
>>> browser = Browser()
>>> browser.open('http://localhost/app')
>>> browser.url
'http://localhost/app/@@login?camefrom=http%3A%2F%2Flocalhost%2Fapp%2F%40%40index'
```

We can't access the projects container:

```
>>> browser.open('http://localhost/app/projects')
>>> browser.url
'http://localhost/app/@@login?camefrom=http%3A%2F%2Flocalhost%2Fapp%2Fprojects%2F%40%40index'
```

We can't access the reports either:

```
>>> browser.open('http://localhost/app/@@logs-report')
>>> browser.url
'http://localhost/app/@@login?camefrom=http%3A%2F%2Flocalhost%2Fapp%2F%40%40logs-report'

>>> browser.open('http://localhost/app/@@tasks-report')
>>> browser.url
'http://localhost/app/@@login?camefrom=http%3A%2F%2Flocalhost%2Fapp%2F%40%40tasks-report'
```

We can't access the clients container:

```
>>> browser.open('http://localhost/app/clients')
>>> browser.url
'http://localhost/app/@@login?camefrom=http%3A%2F%2Flocalhost%2Fapp%2Fclients%2F%40%40index'
```

Now we authenticate using the user we created using the login form:

```
>>> browser.open('http://localhost/app')
>>> browser.getControl(name="form.username").value = u'user'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl("Login").click()
>>> 'You are logged in.' in browser.contents
True
```

And we can access everything. For example, we can access the projects container:

```
>>> browser.getLink('Projects').click()
>>> browser.url
'http://localhost/app/projects'
```

We can also access the reports:

```
>>> browser.open('http://localhost/app/@@logs-report')
>>> browser.url
'http://localhost/app/@@logs-report'

>>> browser.open('http://localhost/app/@@tasks-report')
>>> browser.url
'http://localhost/app/@@tasks-report'
```

We can also access the clients container:

```
>>> browser.open('http://localhost/app/clients')
>>> browser.url
'http://localhost/app/clients'
```

Let's go to the *user folder* and create a new user:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('Add new User').click()
```

If we try too set a user name with strange characters, the form submission will fail:

```
>>> browser.getControl(name="form.id").value = u'jdoe/'
>>> browser.getControl(name="form.real_name").value = u'John Doe'
>>> browser.getControl(name="form.password").value = u'easy'
>>> browser.getControl(name="form.confirm_password").value = u'easy'
>>> browser.getControl("Add user").click()
>>> 'Invalid user name, only characters in [a-z0-9] are allowed' in \
...     browser.contents
True
```

Let's also check that no user were created in the ZODB by checking that the only existing user is the one we created at the beginning:

```
>>> users = app['users']
>>> len(users.values())
1
>>> users.values()[0].id
'user'
```

Let's fix the user name in the form and see what happens if we enter different values in the *password* and *confirm password* fields:

```
>>> browser.getControl(name="form.id").value = u'jdoe'
>>> browser.getControl(name="form.real_name").value = u'John Doe'
>>> browser.getControl(name="form.password").value = u'something'
>>> browser.getControl(name="form.confirm_password").value = u'different'
>>> browser.getControl("Add user").click()
>>> 'Passwords does not match' in browser.contents
True
```

Let's finally fill the form properly and create the user:

```
>>> browser.getControl(name="form.password").value = u'something'
>>> browser.getControl(name="form.confirm_password").value = u'something'
>>> browser.getControl("Add user").click()
>>> 'User added' in browser.contents
True
>>> 'John Doe' in browser.contents
True
```

And the user is now persisted:

```
>>> len(users.values())
2
>>> 'jdoe' in [u.id for u in users.values()]
True
```

We can now edit the user we've just added:

```
>>> browser.getLink('edit', index=0).click()
>>> 'jdoe' in browser.contents
True
```

There is a *username* field in the edit form, but its value can't be changed. We don't allow user IDs to change as they are used to reference users in other parts of the system:

```
>>> try:
...     browser.getControl(name='form.id').value = 'changed'
... except AttributeError as detail:
...     detail
AttributeError("control 'form.id' is readonly",)
```

Let's change the *real name* to something else and save the changes:

```
>>> browser.getControl(name='form.real_name').value = u'Something Else'
>>> browser.getControl('Save').click()
>>> 'Changes saved' in browser.contents
True
```

We got redirected to the container user folder:

```
>>> browser.url
'http://localhost/app/users'
```

And the change is in place:

```
>>> 'Something Else' in browser.contents
True
```

Let's check that the password for the user *Something Else* was no modified. So we logout:

```
>>> browser = Browser()
```

And we use the login form to login into the site:

```
>>> browser.open('http://localhost/app')
>>> browser.getControl(name="form.username").value = u'jdoe'
>>> browser.getControl(name="form.password").value = u'something'
>>> browser.getControl("Login").click()
>>> 'You are logged in.' in browser.contents
True
```

Let's change the password of the user we first created:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('edit', index=1).click()
>>> 'Testing User' in browser.contents
True
```

Once again, if we enter different values for the *password* and *confirm password* fields, we get a validation error:

```
>>> browser.getControl(name="form.password").value = u'super'
>>> browser.getControl(name="form.confirm_password").value = u'super2'
>>> browser.getControl('Save').click()
>>> 'Passwords does not match' in browser.contents
True
>>> browser.url
'http://localhost/app/users/user/edit'
```

So, let's change the password for real:

```
>>> browser.getControl(name="form.password").value = u'super'
>>> browser.getControl(name="form.confirm_password").value = u'super'
>>> browser.getControl('Save').click()
>>> 'Changes saved' in browser.contents
True
>>> browser.url
'http://localhost/app/users'
```

Now let's try to change our own password:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('edit', index=0).click()
>>> 'jdoe' in browser.contents
True
>>> browser.getControl(name="form.password").value = u'supersecret'
>>> browser.getControl(name="form.confirm_password").value = u'supersecret'
>>> browser.getControl('Save').click()
>>> 'Changes saved' in browser.contents
True
```

As our credentials changed, we are kicked off the site:

```
>>> browser.url
'http://localhost/app/@@login?camefrom=http%3A%2F%2Flocalhost%2Fapp%2Fusers%2F%40%40index'
```

The old credentials are no longer valid:

```
>>> browser.getControl(name="form.username").value = u'jdoe'
>>> browser.getControl(name="form.password").value = u'something'
>>> browser.getControl("Login").click()
>>> 'Invalid username and/or password' in browser.contents
True
```

Let's login back using the new password:

```
>>> browser.getControl(name="form.username").value = u'jdoe'
>>> browser.getControl(name="form.password").value = u'supersecret'
>>> browser.getControl("Login").click()
>>> 'You are logged in.' in browser.contents
True
```

Now let's delete the user *Testing User*:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('delete', index=1).click()
>>> 'Are you sure you want to delete the "user" item?' in browser.contents
True
```

We can cancel the deletion, in that case, the user won't be deleted and we will get redirected to the user listing:

```
>>> browser.getControl('Cancel').click()
>>> browser.url
'http://localhost/app/users'
>>> 'Testing User' in browser.contents
True
```

Well, let's delete the user for real now:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('delete', index=1).click()
>>> 'Are you sure you want to delete the "user" item?' in browser.contents
True
>>> browser.getControl('Delete').click()
>>> 'User deleted.' in browser.contents
True
```

And let's logout from the site:

```
>>> browser.getLink('Logout').click()
>>> browser.url.startswith('http://localhost/app/@@login')
True
```

1.3.5 Client Model Functional Tests

We need to create a user to play with later. First of all we access the site as admin:

```
>>> from zope.app.wsgi.testlayer import Browser
>>> browser = Browser()
>>> browser.handleErrors = False
>>> browser.addHeader('Authorization', 'Basic admin:admin')
>>> browser.open('http://localhost/app')
```

```
>>> 'Logged in as: Manager' in browser.contents
True
```

Now we create a new user that we will use through this document:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('Add new User').click()
>>> browser.getControl(name="form.id").value = u'user'
>>> browser.getControl(name="form.real_name").value = u'Testing User'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl(name="form.confirm_password").value = u'secret'
>>> browser.getControl("Add user").click()
>>> 'User added' in browser.contents
True
>>> 'Testing User' in browser.contents
True
```

Let's log in with the user we've just created:

```
>>> browser = Browser()
>>> browser.open('http://localhost/app')
>>> browser.getControl(name="form.username").value = u'user'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl("Login").click()
>>> 'You are logged in.' in browser.contents
True
```

Let's create a *client*. First we click to *Clients* tab:

```
>>> browser.getLink('Clients').click()
>>> 'There are currently no clients.' in browser.contents
True
```

And we add a client:

```
>>> browser.getLink('Add new Client').click()
>>> browser.getControl(name='form.title').value = u'Acme'
>>> browser.getControl(name='form.type').value = ('NGO',)
>>> browser.getControl("Add client").click()
>>> 'Client added' in browser.contents
True
>>> 'There are currently no clients.' in browser.contents
False
```

We edit the client we've just created:

```
>>> browser.getLink('edit').click()
>>> browser.url
'http://localhost/app/clients/1/edit'
>>> browser.getControl(name='form.title').value = u'Some Company'
>>> browser.getControl(name='form.type').value = ('Company',)
>>> browser.getControl('Save').click()
>>> 'Changes saved' in browser.contents
True
```

Let's now create a new project associated with the client we created:

```
>>> browser.getLink('Projects').click()
>>> browser.getLink('Add new Project').click()
>>> browser.getControl(name="form.title").value = u'Project'
>>> browser.getControl(name='form.client').value = ('/app/clients/1',)
```

```
>>> browser.getControl('Add project').click()
>>> 'Project added' in browser.contents
True
```

If we try to delete the client, we will not be able to because it has a client associated:

```
>>> browser.getLink('Clients').click()
>>> browser.getLink('delete').click()
>>> 'Are you sure you want to delete the "Some Company" item?' in \
...     browser.contents
True
>>> browser.getControl('Delete').click()
>>> 'This client cannot be deleted because it has projects associated' in \
...     browser.contents
True
```

So, let's delete the project and try again:

```
>>> browser.getLink('Projects').click()
>>> browser.getLink('delete').click()
>>> 'Are you sure you want to delete the "Project" item?' in \
...     browser.contents
True
>>> browser.getControl('Delete').click()
>>> 'Project deleted' in browser.contents
True
```

Now we go back and try to delete the client:

```
>>> browser.getLink('Clients').click()
>>> browser.getLink('delete').click()
>>> 'Are you sure you want to delete the "Some Company" item?' in \
...     browser.contents
True
>>> browser.getControl('Delete').click()
>>> 'Client deleted' in browser.contents
True
```

1.3.6 Simple Project Functional Tests

We need to create a user to play with later. First of all we access the site as admin:

```
>>> from zope.app.wsgi.testlayer import Browser
>>> browser = Browser()
>>> browser.handleErrors = False
>>> browser.addHeader('Authorization', 'Basic admin:admin')
>>> browser.open('http://localhost/app')
>>> 'Logged in as: Manager' in browser.contents
True
```

Now we create a new user:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('Add new User').click()
>>> browser.getControl(name="form.id").value = u'user'
>>> browser.getControl(name="form.real_name").value = u'Testing User'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl(name="form.confirm_password").value = u'secret'
>>> browser.getControl("Add user").click()
```

```
>>> 'User added' in browser.contents
True
>>> 'Testing User' in browser.contents
True
```

We are now ready to start testing projects. Let's log in with the user we've just created:

```
>>> browser = Browser()
>>> browser.open('http://localhost/app')
>>> browser.getControl(name="form.username").value = u'user'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl("Login").click()
>>> 'You are logged in.' in browser.contents
True
```

Since we must associate projects to clients, let's start by creating a client:

```
>>> browser.getLink('Clients').click()
>>> browser.getLink('Add new Client').click()
>>> browser.getControl(name="form.title").value = u'Acme'
>>> browser.getControl("Add client").click()
>>> 'Client added' in browser.contents
True
```

Now we can create a project. Let's try to submit the add project form after filling just the title and check that we get proper validation errors to fill required fields:

```
>>> browser.getLink('Projects').click()
>>> browser.getLink('Add new Project').click()
>>> browser.getControl(name="form.title").value = u'Develop a web app'
>>> browser.getControl("Add project").click()
>>> 'Required input is missing' in browser.contents
True
```

The project status defaults to In progress:

```
>>> browser.getControl(name='form.status').value
['In progress']
```

Also, the start date is automatically set to today:

```
>>> from datetime import datetime, timedelta
>>> from merlot.lib import DATE_FORMAT
>>> today = datetime.today().strftime(DATE_FORMAT)
>>> browser.getControl(name='form.start_date').value == today
True
```

If we try to set an end date prior to the start date, we will also get a validation error:

```
>>> yesterday = datetime.today() - timedelta(1)
>>> yesterday = yesterday.strftime(DATE_FORMAT)
>>> browser.getControl(name='form.end_date').value = yesterday
>>> browser.getControl("Add project").click()
>>> 'Start date must preceed end date' in browser.contents
True
```

So let's set the end date to tomorrow and fill the client field to finally add the project:

```
>>> tomorrow = datetime.today() + timedelta(1)
>>> tomorrow = tomorrow.strftime(DATE_FORMAT)
>>> browser.getControl(name='form.end_date').value = tomorrow
```

```
>>> browser.getControl(name='form.client').value = ('/app/clients/1',)
>>> browser.getControl("Add project").click()
>>> 'Project added' in browser.contents
True
```

And we were redirected to the created project view. Notice how the URL was generated based on the title we used:

```
>>> browser.url
'http://localhost/app/projects/develop-a-web-app'
>>> 'Project view: Develop a web app' in browser.contents
True
```

Projects can contain tasks, so let's create a task, but first let's verify that the *priority* field defaults to *Normal*, that the *status* field defaults to *In progress* and that *start_date* default to the current date:

```
>>> browser.getLink('Add new Task').click()
>>> browser.getControl(name='form.priority').value
['Normal']
>>> browser.getControl(name='form.status').value
['In progress']
>>> browser.getControl(name='form.start_date').value == today
True
```

No we will fill some fields and submit the form. Once again, if we set an end date prior to the start date, we get a validation error:

```
>>> browser.getControl(name="form.title").value = u'Define requirements'
>>> browser.getControl(name='form.end_date').value = yesterday
>>> browser.getControl("Add task").click()
>>> 'Start date must preceed end date' in browser.contents
True
```

Let's set the end date to tomorrow and add the task:

```
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl("Add task").click()
>>> 'Task added' in browser.contents
True
```

We are still in the project view:

```
>>> browser.url
'http://localhost/app/projects/develop-a-web-app'
>>> 'Project view: Develop a web app' in browser.contents
True
```

Let's quickly add another task:

```
>>> browser.getLink('Add new Task').click()
>>> browser.getControl(name="form.title").value = u'Testing'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl("Add task").click()
>>> 'Task added' in browser.contents
True
```

We can delete a task from the project view:

```
>>> browser.getLink('delete', index=2).click()
>>> 'Are you sure you want to delete the "Testing" item?' in \
...     browser.contents
True
```

```
>>> browser.getControl('Delete').click()
>>> 'Task deleted.' in browser.contents
True
```

And we are still in the project view:

```
>>> browser.url
'http://localhost/app/projects/develop-a-web-app'
>>> 'Project view: Develop a web app' in browser.contents
True
```

In order to track the time that a task takes, you can associate time logs to them. Let's go the the task view, and there we can add a log:

```
>>> browser.getLink('Define requirements').click()
>>> 'Task view: Define requirements' in browser.contents
True
>>> browser.getControl(name='form.description').value = u'Write document'
>>> browser.getControl(name='form.date').value == today
True
>>> browser.getControl(name='form.hours').value = u'6'
>>> browser.getControl(name='form.remaining').value = u'2.4'
>>> browser.getControl('Add log').click()
>>> 'Log added' in browser.contents
True
>>> 'Write document' in browser.contents
True
```

We are still in the task view:

```
>>> 'Task view: Define requirements' in browser.contents
True
```

The remaining hours set when adding a log updates the remaining hours field in the task:

```
>>> from decimal import Decimal
>>> task = app['projects']['develop-a-web-app']['define-requirements']
>>> task.remaining == Decimal('2.4')
True
```

Let's check that there are some required fields to add a log by submitting the form without filling any field:

```
>>> browser.getControl('Add log').click()
>>> 'Required input is missing' in browser.contents
True
```

Let's mark the current task as starred, but before, let's check what are the current starred tasks for the authenticated user:

```
>>> from merlot.interfaces import IStarredTasks
>>> from zope.component import getUtility
>>> from zope.app.authentication.interfaces import IAuthenticatorPlugin
>>> from zope.intid.interfaces import IIntIds
>>> user = app['users']['user']
>>> starred_tasks = IStarredTasks(user)
>>> starred_tasks.getStarredTasks()
[]
```

Now we mark the task as starred:

```
>>> browser.getLink(url=('http://localhost/app/projects/develop-a-web-app/'
...                      'define-requirements/toggle-starred')).click()
```

Now the task is marked as starred for the current user:

```
>>> intids = getUtility(IIntIds, name='intids', context=app)
>>> intid = intids.getId(task)
>>> starred_tasks.getStarredTasks() == [intid]
True

>>> link = browser.getLink(url=('http://localhost/app/projects/'
...                             'develop-a-web-app/define-requirements/'
...                             'toggle-starred'))
>>> link.attrs['class'] == 'starred-selected'
True
```

Let's quickly create another task and mark it as starred:

```
>>> browser.getLink('Develop a web app').click()
>>> browser.getLink('Add new Task').click()
>>> browser.getControl(name="form.title").value = u'New task'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl("Add task").click()
>>> 'Task added' in browser.contents
True
>>> browser.getLink('New task').click()
>>> browser.getLink(url=('http://localhost/app/projects/develop-a-web-app/'
...                      'new-task/toggle-starred')).click()
```

Let's check that it is actually marked as starred for the authenticated user:

```
>>> newtask = app['projects']['develop-a-web-app']['new-task']
>>> newtask_intid = intids.getId(newtask)
>>> starred_tasks.getStarredTasks() == [intid, newtask_intid]
True
```

Let's now edit the first task and change the hours estimate to 10:

```
>>> browser.getLink('Develop a web app').click()
>>> browser.getLink('Define requirements').click()
>>> browser.getLink('Edit').click()
>>> browser.getControl(name='form.estimate').value = '10'
>>> browser.getControl('Save').click()
>>> 'Changes saved' in browser.contents
True
```

The changes persisted:

```
>>> task.estimate == Decimal(10)
True
```

Logs can also be edited:

```
>>> browser.getLink('edit', index=1).click()
>>> browser.getControl(name='form.description').value = 'New description'
>>> browser.getControl('Save').click()
>>> 'Changes saved' in browser.contents
True
>>> 'New description' in browser.contents
True
```

```
>>> 'Write document' in browser.contents
False
```

If a task is deleted, it will be automatically removed from all users' starred tasks lists. Lets delete one of the tasks and check that it's also removed from the starred tasks list of the authenticated user:

```
>>> browser.getLink('Delete').click()
>>> 'Are you sure you want to delete the "Define requirements" item?' in \
...     browser.contents
True
>>> browser.getControl('Delete').click()
>>> 'Task deleted' in browser.contents
True
>>> starred_tasks.getStarredTasks() == [newtask_intid]
True
```

Moreover, if we delete the project that contains an starred task, then that task is also removed from all users' starred tasks lists. Let's delete the project and test this:

```
>>> browser.getLink('Delete').click()
>>> 'Are you sure you want to delete the "Develop a web app" item?' in \
...     browser.contents
True
>>> browser.getControl('Delete').click()
>>> 'Project deleted' in browser.contents
True
>>> starred_tasks.getStarredTasks()
[]
```

Let's now create a new project:

```
>>> browser.getLink('Add new Project').click()
>>> browser.getControl(name="form.title").value = u'Project'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl(name='form.client').value = ('/app/clients/1',)
>>> browser.getControl("Add project").click()
>>> 'Project added' in browser.contents
True
```

Let's create another project with the same title and check that the IDs don't clash:

```
>>> browser.getLink('Projects').click()
>>> browser.getLink('Add new Project').click()
>>> browser.getControl(name="form.title").value = u'Project'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl(name='form.client').value = ('/app/clients/1',)
>>> browser.getControl("Add project").click()
>>> 'Project added' in browser.contents
True
>>> browser.url
'http://localhost/app/projects/project1'
```

Let's edit the current project by changing the title and start date:

```
>>> browser.getLink('Edit').click()
>>> browser.getControl(name="form.title").value = u'Project 2'
>>> browser.getControl(name='form.start_date').value = yesterday
>>> browser.getControl('Save').click()
>>> 'Changes saved' in browser.contents
True
```

```
>>> browser.url
'http://localhost/app/projects/project1'
```

And let's check that the changes persisted:

```
>>> project1 = app['projects']['project1']
>>> project1.title
u'Project 2'
>>> project1.start_date == datetime.today().date() - timedelta(1)
True
```

1.3.7 Reports Functional Tests

There are currently two reports available, a *Logs report* and a *Tasks report*. Read on to learn about what those are about.

We need to create a user to play with later. First of all we access the site as admin:

```
>>> from zope.app.wsgi.testlayer import Browser
>>> browser = Browser()
>>> browser.handleErrors = False
>>> browser.addHeader('Authorization', 'Basic admin:admin')
>>> browser.open('http://localhost/app')
>>> 'Logged in as: Manager' in browser.contents
True
```

Now we create a new user:

```
>>> browser.getLink('Users').click()
>>> browser.getLink('Add new User').click()
>>> browser.getControl(name="form.id").value = u'user'
>>> browser.getControl(name="form.real_name").value = u'Testing User'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl(name="form.confirm_password").value = u'secret'
>>> browser.getControl("Add user").click()
>>> 'User added' in browser.contents
True
>>> 'Testing User' in browser.contents
True
```

Let's log in with the user we've just created:

```
>>> browser = Browser()
>>> browser.open('http://localhost/app')
>>> browser.getControl(name="form.username").value = u'user'
>>> browser.getControl(name="form.password").value = u'secret'
>>> browser.getControl("Login").click()
>>> 'You are logged in.' in browser.contents
True
```

Since we must associate projects to clients, let's start by creating a client:

```
>>> browser.getLink('Clients').click()
>>> browser.getLink('Add new Client').click()
>>> browser.getControl(name="form.title").value = u'Acme'
>>> browser.getControl("Add client").click()
>>> 'Client added' in browser.contents
True
```

Let's now create two new projects, but first let's set some formatted strings with dates:

```
>>> from datetime import datetime, timedelta
>>> from merlot.lib import DATE_FORMAT
>>> today = datetime.today().strftime(DATE_FORMAT)
>>> yesterday = datetime.today() - timedelta(1)
>>> yesterday = yesterday.strftime(DATE_FORMAT)
>>> tomorrow = datetime.today() + timedelta(1)
>>> tomorrow = tomorrow.strftime(DATE_FORMAT)

>>> browser.getLink('Projects').click()
>>> browser.getLink('Add new Project').click()
>>> browser.getControl(name="form.title").value = u'Project One'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl(name='form.client').value = ('/app/clients/1',)
>>> browser.getControl("Add project").click()
>>> 'Project added' in browser.contents
True

>>> browser.getLink('Projects').click()
>>> browser.getLink('Add new Project').click()
>>> browser.getControl(name="form.title").value = u'Project Two'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl(name='form.client').value = ('/app/clients/1',)
>>> browser.getControl("Add project").click()
>>> 'Project added' in browser.contents
True
```

Let's create one task in each of the projects we created. We create a task in *Project Two*:

```
>>> browser.getLink('Add new Task').click()
>>> browser.getControl(name='form.start_date').value == today
True
>>> browser.getControl(name="form.title").value = u'Define requirements'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl("Add task").click()
>>> 'Task added' in browser.contents
True
```

And we also create a task in *Project One*:

```
>>> browser.getLink('Projects').click()
>>> browser.getLink('Project One').click()
>>> browser.getLink('Add new Task').click()
>>> browser.getControl(name="form.title").value = u'A simple task'
>>> browser.getControl(name='form.end_date').value = tomorrow
>>> browser.getControl("Add task").click()
>>> 'Task added' in browser.contents
True
```

Let's now log some time in both tasks. First we add a couple of logs in the task we've just created:

```
>>> browser.getLink('A simple task').click()
>>> 'Task view: A simple task' in browser.contents
True
>>> browser.getControl(name='form.description').value = u'Write document'
>>> browser.getControl(name='form.date').value = yesterday
>>> browser.getControl(name='form.hours').value = u'6'
>>> browser.getControl(name='form.remaining').value = u'2.4'
>>> browser.getControl('Add log').click()
```

```
>>> 'Log added' in browser.contents
True
>>> browser.getControl(name='form.description').value = u'Close this task'
>>> browser.getControl(name='form.date').value = today
>>> browser.getControl(name='form.hours').value = u'3'
>>> browser.getControl(name='form.remaining').value = u'0'
>>> browser.getControl('Add log').click()
>>> 'Log added' in browser.contents
True
```

And now we add a couple of logs in the task we created for *Project Two*:

```
>>> browser.getLink('Projects').click()
>>> browser.getLink('Project Two').click()
>>> browser.getLink('Define requirements').click()
>>> 'Task view: Define requirements' in browser.contents
True
>>> browser.getControl(name='form.description').value = u'Meeting with Joe'
>>> browser.getControl(name='form.hours').value = u'4'
>>> browser.getControl(name='form.remaining').value = u'2'
>>> browser.getControl('Add log').click()
>>> 'Log added' in browser.contents
True
>>> browser.getControl(name='form.description').value = u'Finish document'
>>> browser.getControl(name='form.hours').value = u'3'
>>> browser.getControl(name='form.date').value = tomorrow
>>> browser.getControl(name='form.remaining').value = u'0'
>>> browser.getControl('Add log').click()
>>> 'Log added' in browser.contents
True
```

We are now ready to run the *Logs report*. This report queries the log entries in a range of dates filtering by project and user. The results are presented in a flat table.

So, to get to the report screen, we click on the *Reports* tab and then on the *Logs report* link:

```
>>> browser.getLink('Reports').click()
>>> browser.getLink('Logs report').click()
>>> 'Run logs report' in browser.contents
True
```

The *from* and *to* dates are set to today:

```
>>> browser.getControl(name='form.from_date').value == today
True
>>> browser.getControl(name='form.to_date').value == today
True
```

All users and all projects are selected by default:

```
>>> browser.getControl(name='form.project_or_client').value
['all']
>>> browser.getControl(name='form.user').value
['all']
```

So, if we run the report with those options, we should get only today's logs:

```
>>> browser.getControl('Submit').click()
>>> 'Write document' in browser.contents
False
```

```
>>> 'Close this task' in browser.contents
True
>>> 'Meeting with Joe' in browser.contents
True
>>> 'Finish document' in browser.contents
False
```

If we set the *from* date to yesterday, we will also get yesterday's logs:

```
>>> browser.getControl(name='form.from_date').value = yesterday
>>> browser.getControl('Submit').click()
>>> 'Write document' in browser.contents
True
>>> 'Close this task' in browser.contents
True
>>> 'Meeting with Joe' in browser.contents
True
>>> 'Finish document' in browser.contents
False
```

If we rescrict the report to *Project One*:

```
>>> browser.getControl(name='form.project_or_client').value = \
...     ('/app/projects/project-one',)
>>> browser.getControl('Submit').click()
>>> 'Write document' in browser.contents
True
>>> 'Close this task' in browser.contents
True
>>> 'Meeting with Joe' in browser.contents
False
>>> 'Finish document' in browser.contents
False
```

The report can be downloaded in CSV format. Let's select all projects again, resubmit the report and download the CSV file:

```
>>> browser.getControl(name='form.project_or_client').value = ('all',)
>>> browser.getControl('Submit').click()
>>> browser.getLink('Download CSV').click()
>>> csv = ('User,Project,Task,Description,Date,Hours\r\n'
...        'user,Project One,A simple task,Write document,%s,6\r\n'
...        'user,Project One,A simple task,Close this task,%s,3\r\n'
...        'user,Project Two,Define requirements,Meeting with Joe,%s,4\r\n'
...        ) % (yesterday, today, today)
>>> browser.contents == csv
True
```

Another report is the *Tasks report*, which queries the tasks worked by a user (or all of them) in a range of dates. The results are presented ordered by project and by task, showing the total amount of hours used for each task and listing the users that logged some time in that task. A sum of hours worked in the project during the period being queried is also displayed.

Let's go the the *Tasks report* page:

```
>>> browser.open('http://localhost/app')
>>> browser.getLink('Reports').click()
>>> browser.getLink('Tasks report').click()
>>> 'Run tasks report' in browser.contents
True
```

The *from* and *to* dates are set to today:

```
>>> browser.getControl(name='form.from_date').value == today
True
>>> browser.getControl(name='form.to_date').value == today
True
```

All users and all projects are selected by default:

```
>>> browser.getControl(name='form.projects').value
['all']
>>> browser.getControl(name='form.user').value
['all']
```

So, if we run the report with those options, we should get only today's tasks, which in this case are both tasks we created:

```
>>> browser.getControl('Submit').click()
>>> 'A simple task' in browser.contents
True
>>> 'Define requirements' in browser.contents
True
```

And if we restrict the report to *Project Two*, only *Define requirements* will be in the results:

```
>>> browser.getControl(name='form.projects').value = \
...     ('/app/projects/project-two',)
>>> browser.getControl('Submit').click()
>>> 'A simple task' in browser.contents
False
>>> 'Define requirements' in browser.contents
True
```

1.4 Source code reference

Merlot's source code is organized following the organization of a standard Grok project. If you are not familiar with it, please refer to the Grok documentation.

The main modules are:

- **interfaces**: where all the interfaces for the application are defined.
- **app**: where the Grok application is defined and the layout is defined based a viewlets structure.
- **project**: where the main models and views for projects, tasks and logs are defined.

1.4.1 The *interfaces* module

1.4.2 The *app* module

1.4.3 The *project* module

1.4.4 The *client* module

1.4.5 The *vocabularies* module

1.4.6 The *auth* module

1.4.7 The *reports* module

1.4.8 The *lib* module

1.5 ChangeLog

Developers:

- Emanuel Sartor (emanuel)
- Silvestre Huens (quimera)
- Juan A. Diaz (nueces)

0.2 - 2012-03-18

- Upgraded Grok version to 1.4.1. [emanuel]
- Added Japanese translation contributed by OCHIAI Gouji. [emanuel]
- Styling tweaks for the JS date range widget. [emanuel]
- Improved i18n support for the data of the *hours usage* and *due in* columns in the project view. [emanuel]
- Make sure that i18n is picked in the templates for the Clients section. [emanuel]
- Tweaked quicklog style and update some wordings. [emanuel]
- Fixed default values for some fields in the task and project models. [emanuel]
- Added buildout deployment configuration. [emanuel]

0.1 - 2011-03-24

- Initial implementation: models for projects, tasks, logs, users and clients and related views; quick log feature; starred tasks feature; reports section and logs report; application style and user interaction; i18n infrastructure and translation to Spanish; documentation and automated tests. [emanuel, quimera]
- Closable flash messages. [nueces]
- Tasks report. [nueces]

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*