groundwork-users Documentation

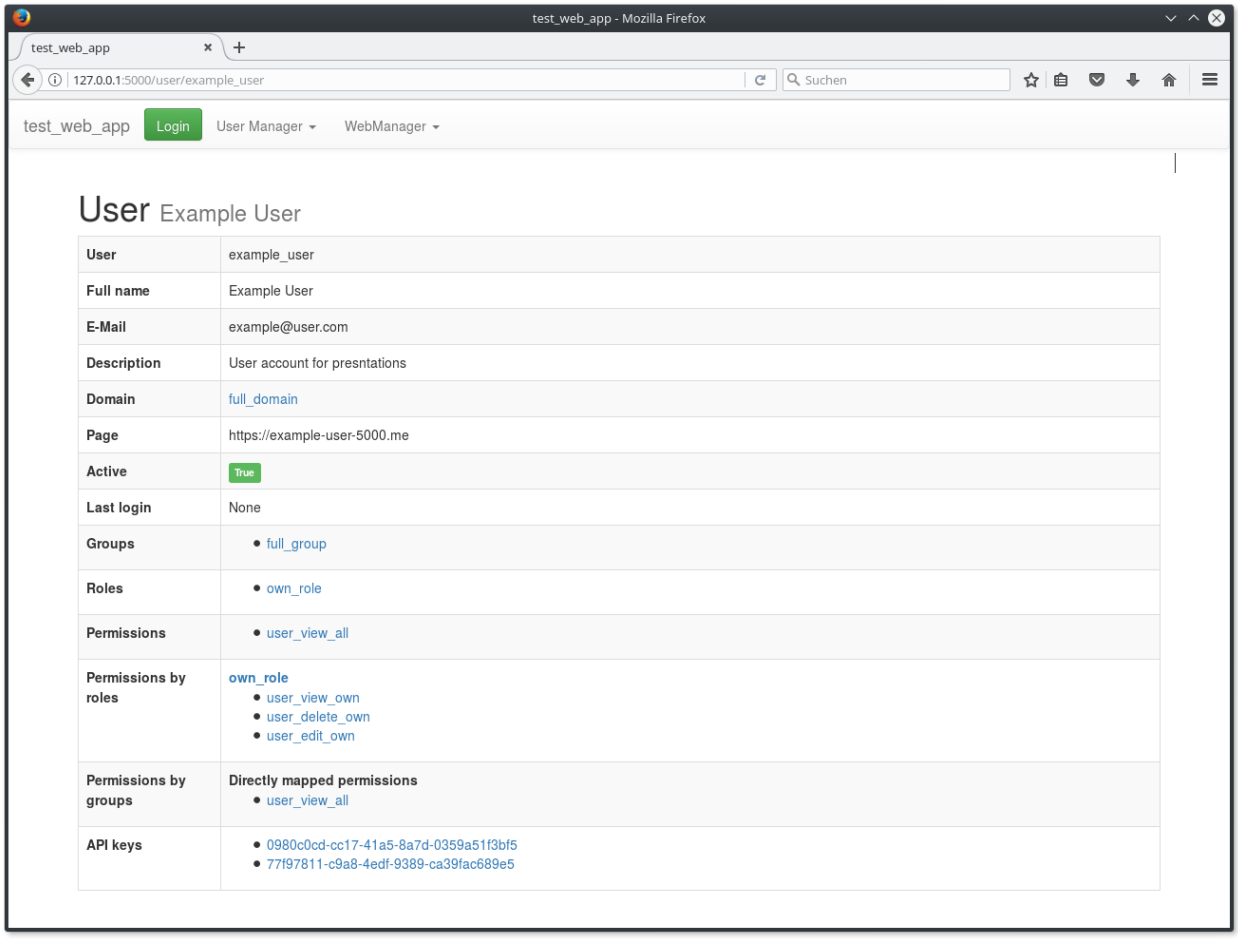
Release 0.1.2

team useblocks

Sep 07, 2017

Contents

This is a groundwork extension package for managing users and related functions.



More images are shown on the *Screenshots* page.

groundwork framework

groundwork is a plugin based Python application framework, which can be used to create various types of applications: console scripts, desktop apps, dynamic websites and more.

Visit groundwork.useblocks.com or read the technical documentation for more information.

CHAPTER 1

Functions

All of the following functions are available as API via *GwUsersPattern* or as ready-to-use web views via the plugin *GwUsersWebManager*.

Users and Groups

- Create, edit and delete users and their data
- Create, edit and delete groups
- Bundle users inside groups
- Activate/Deactivate user accounts

Permissions and Roles

- Create, edit and delete permissions
- Create, edit and delete roles
- Bundle permissions inside roles
- Assign roles to users/groups
- Assign single permission to users/groups
- Secure functions/views with needed permissions

Authentication

- Authenticate users by basicAuth, token, session or api_key
- Secure functions with specific auth methods like “api_key only for API calls”.

API keys

- Create, edit and delete API keys
- Assign multiple API keys to users
- Activate/Deactivate api keys

Domains

- Create, edit and delete domains
- Bundle users to domains for multi-client-support (Multitenancy)

CHAPTER 2

Package content

- Plugins
- GwUsersCliManager
- GwUsersWebManager
- Patterns
- GwUsersPattern

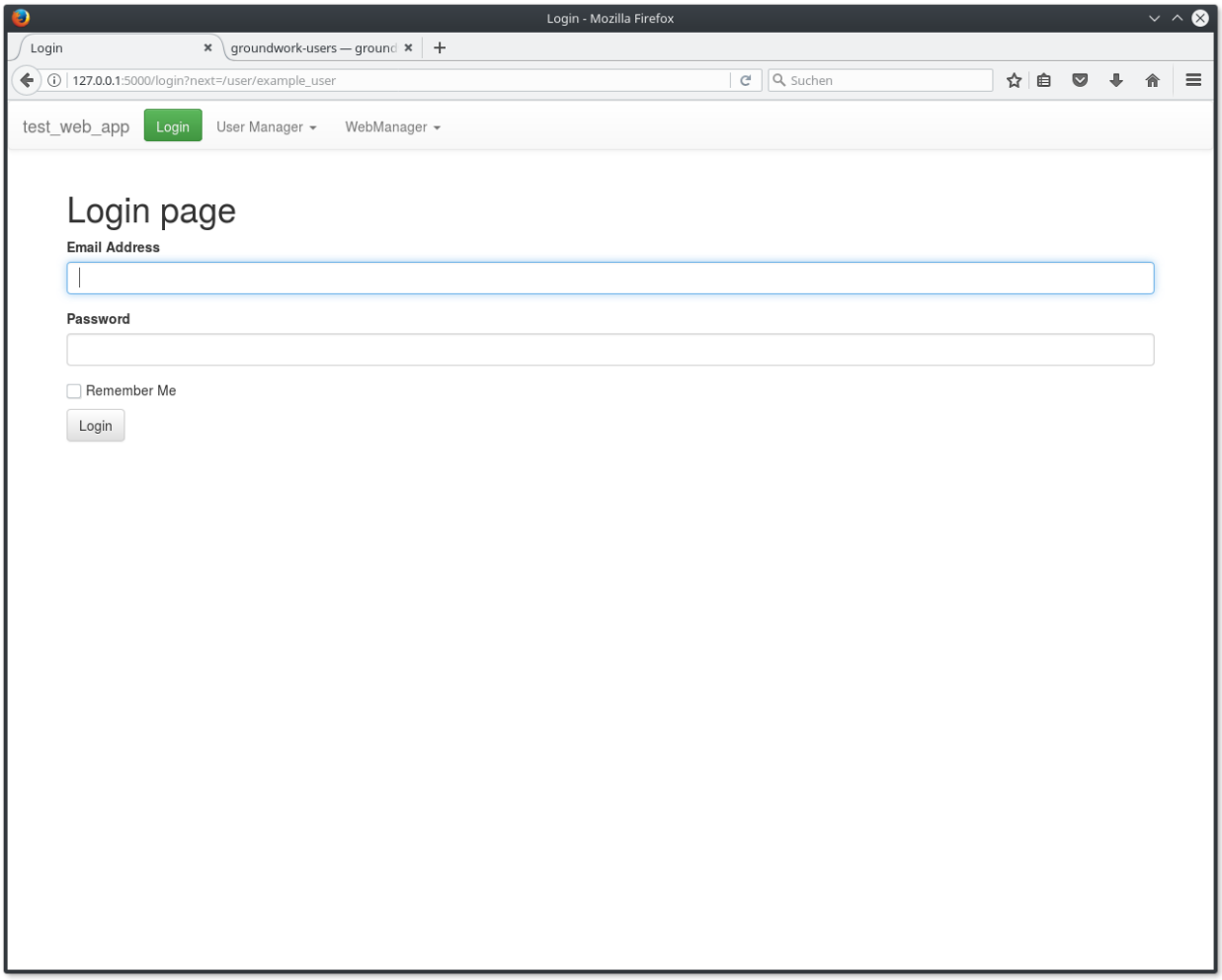
CHAPTER 3

Content

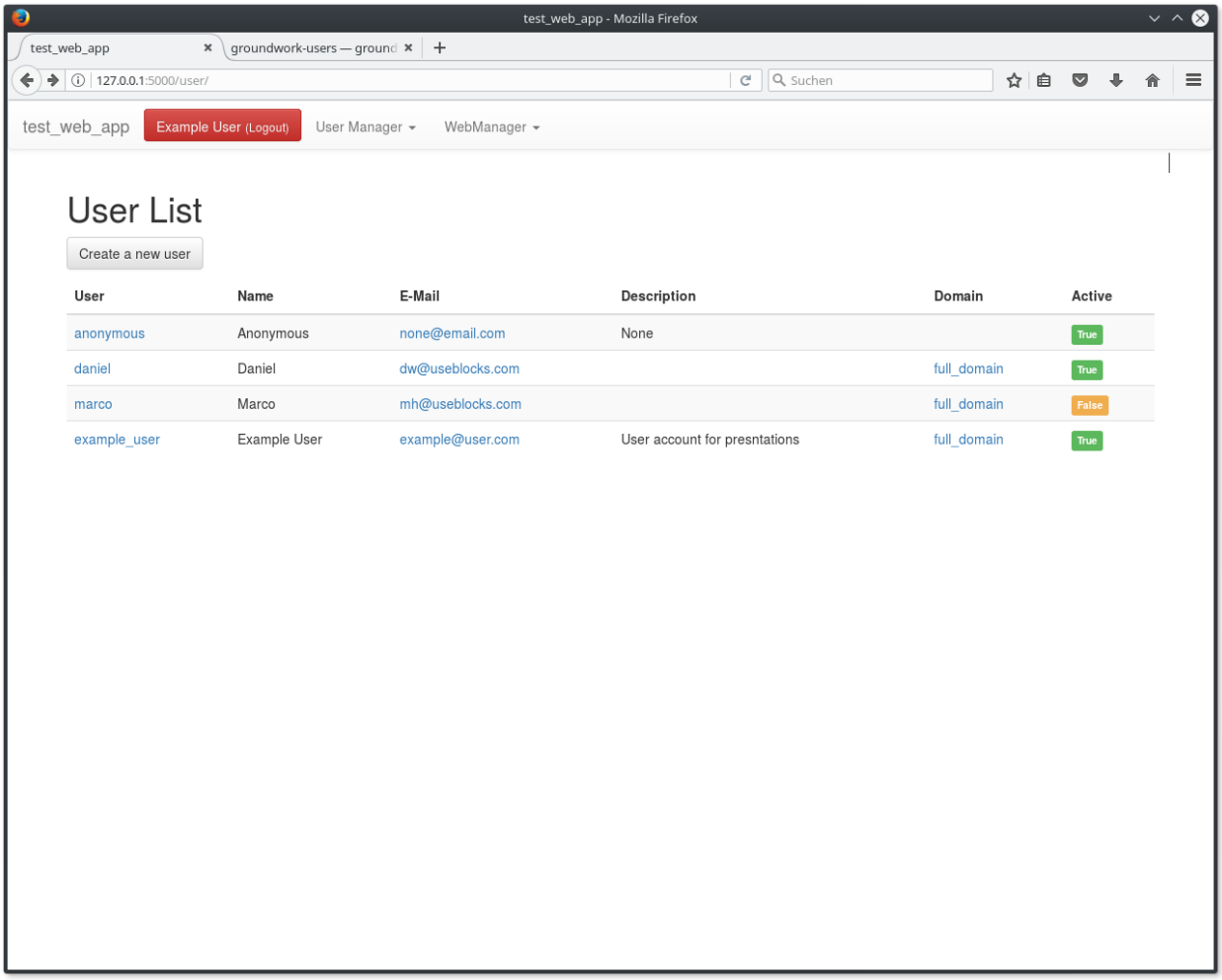
Screenshots

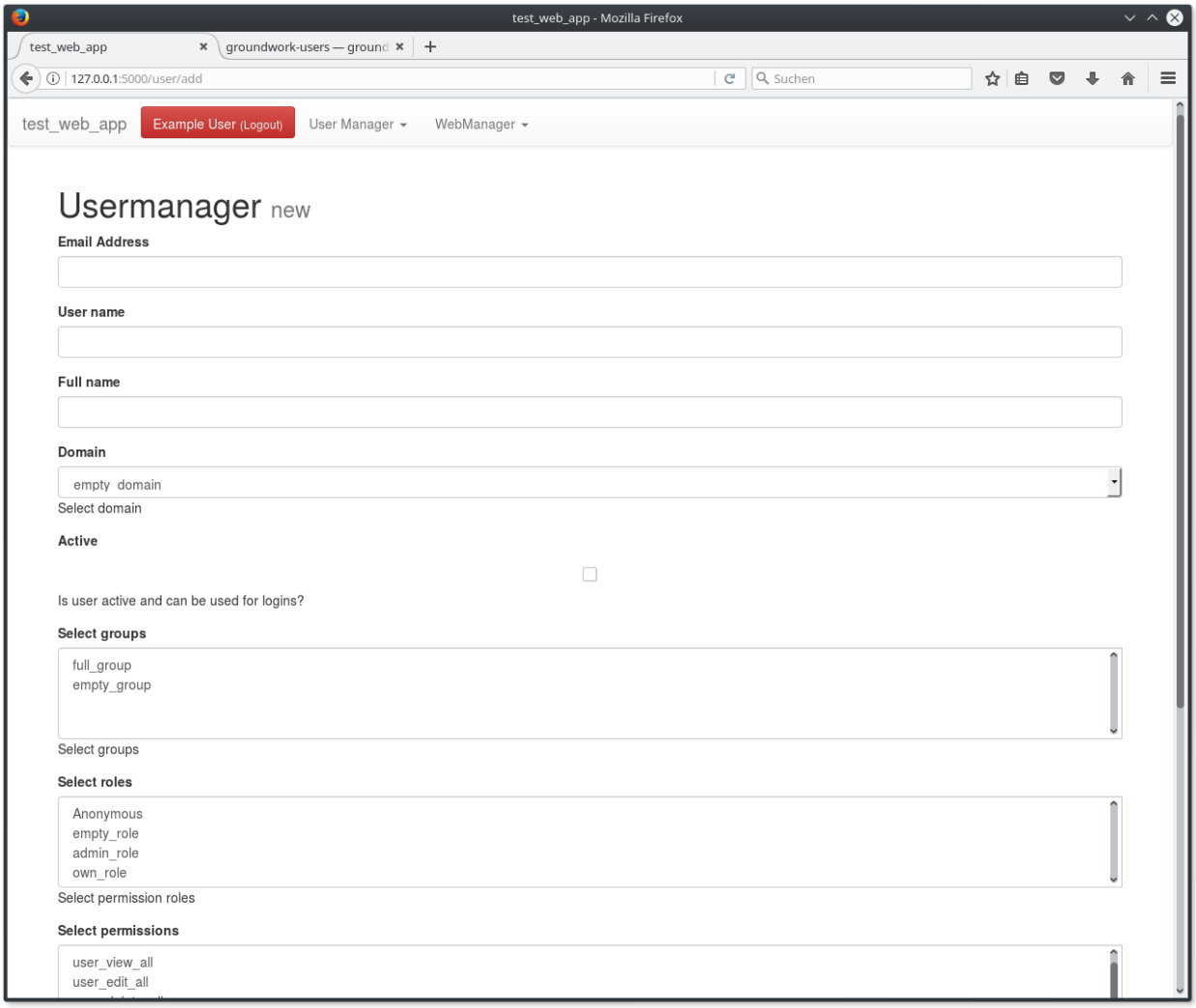
The following screenshots are all taken from the plugin *GwUsersWebManager*.

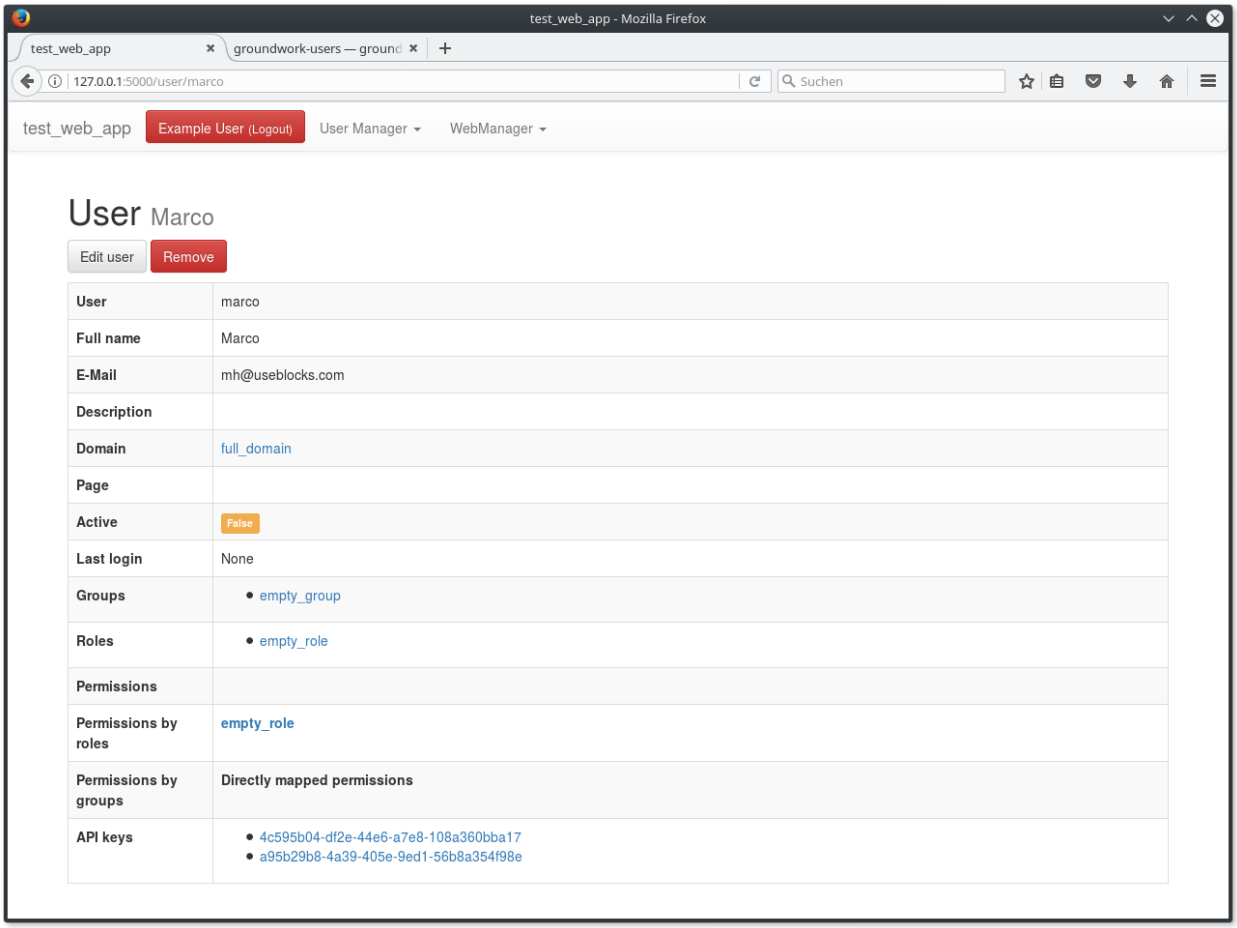
Login

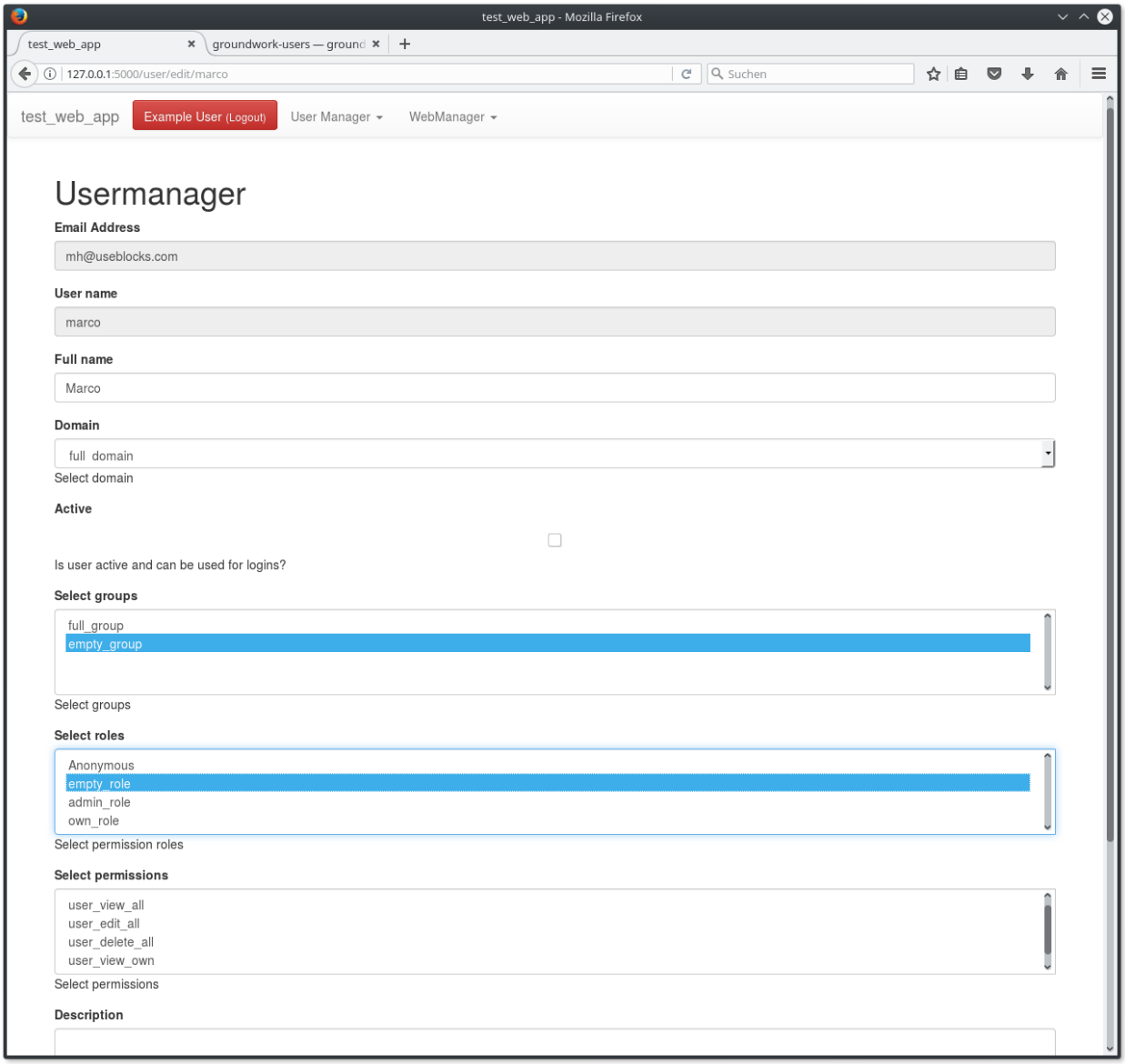

Login - Mozilla Firefox
Login
groundwork-users — ground
127.0.0.1:5000/login?next=/user/example_user
Suchen
test_web_app
Login
User Manager
WebManager
Login page
Email Address
Password
Remember Me
Login

Users

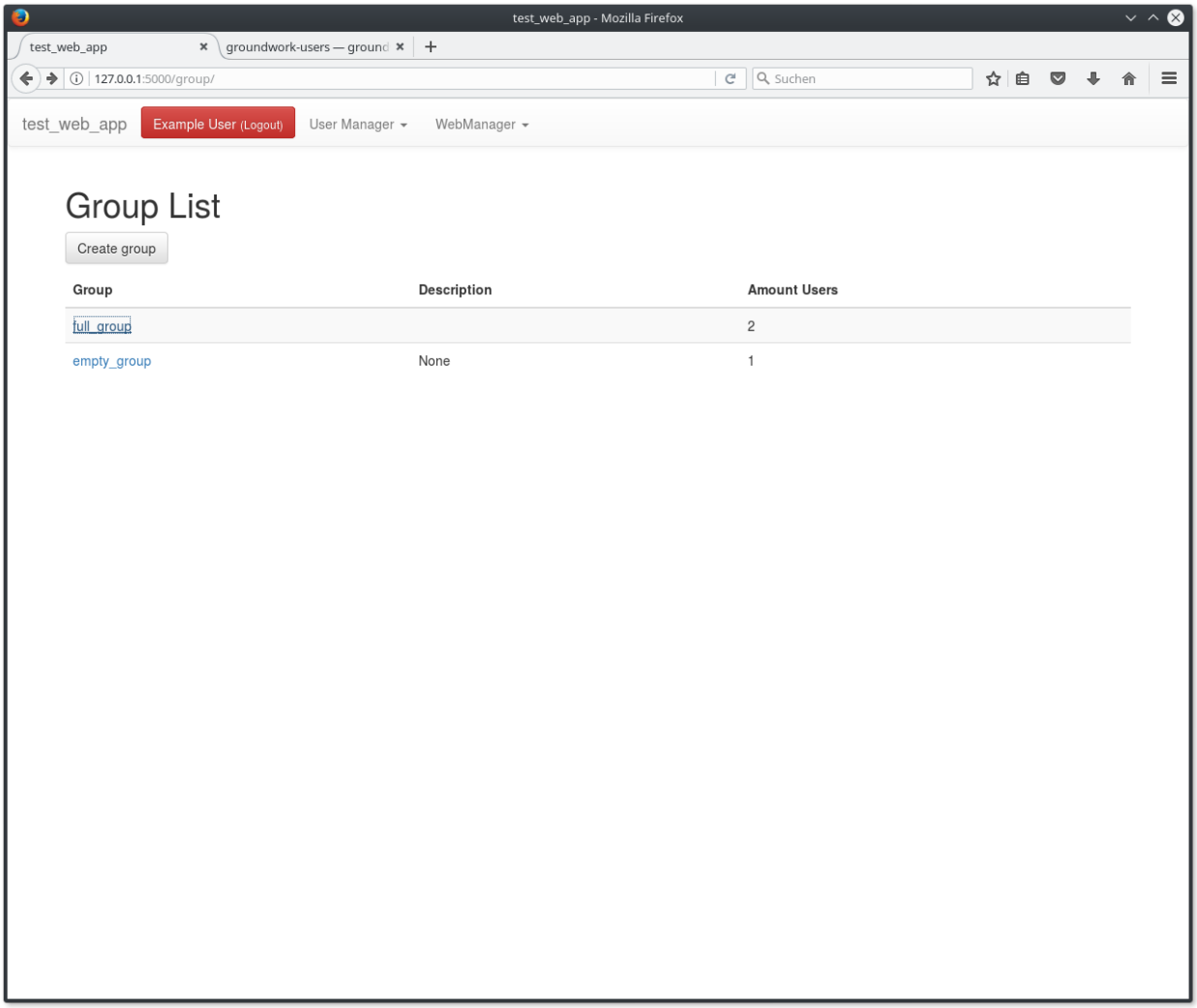

test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/user/
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
User List
Create a new user
User
Name
E-Mail
Description
Domain
Active
anonymous
Anonymous
none@email.com
None
True
daniel
Daniel
dw@useblocks.com
full_domain
True
marco
Marco
mh@useblocks.com
full_domain
False
example_user
Example User
example@user.com
User account for presntations
full_domain
True

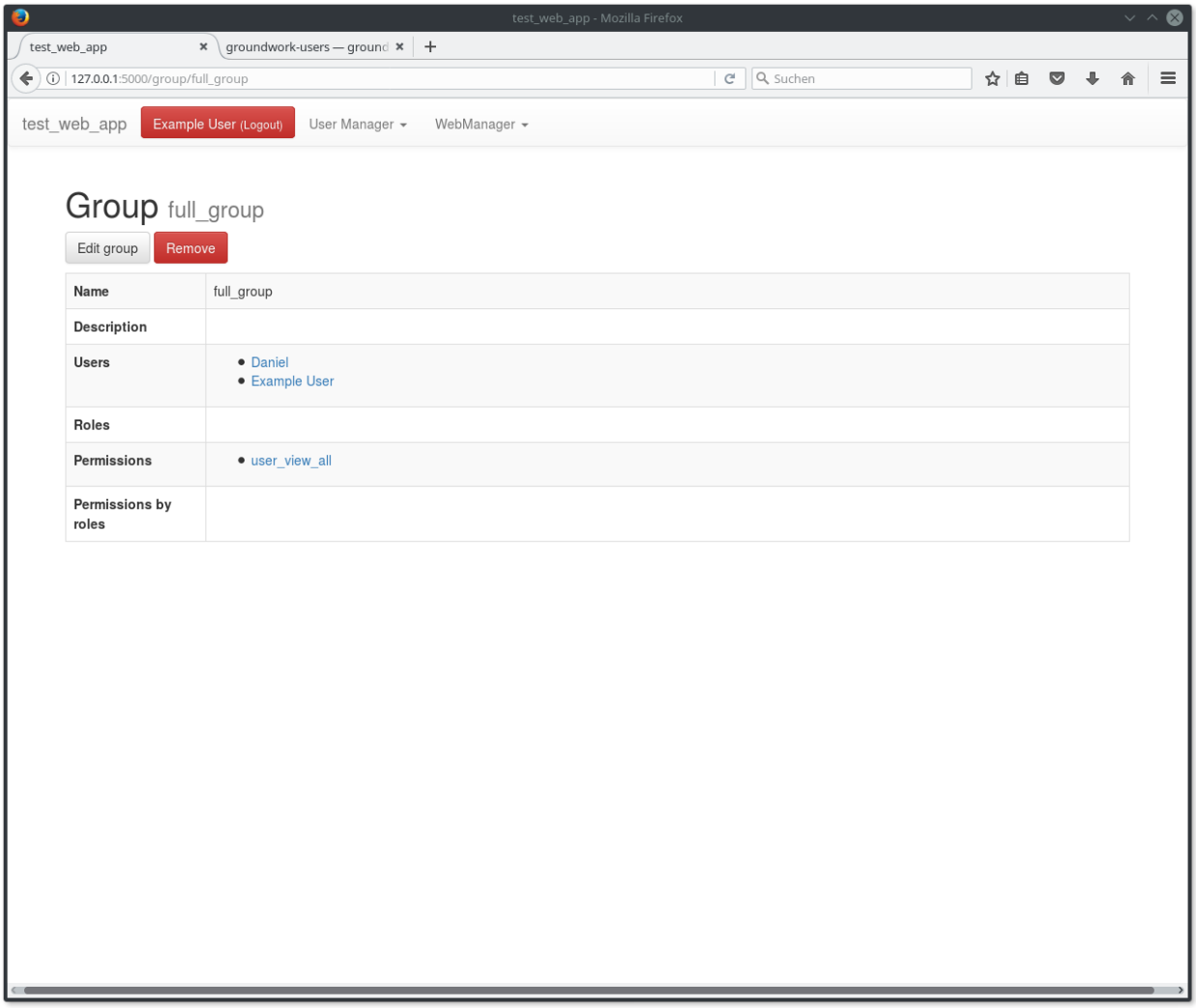

test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/user/add
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Usermanager new
Email Address
User name
Full name
Domain
empty_domain
Select domain
Active
Is user active and can be used for logins?
Select groups
full_group
empty_group
Select groups
Select roles
Anonymous
empty_role
admin_role
own_role
Select permission roles
Select permissions
user_view_all
user_edit_all

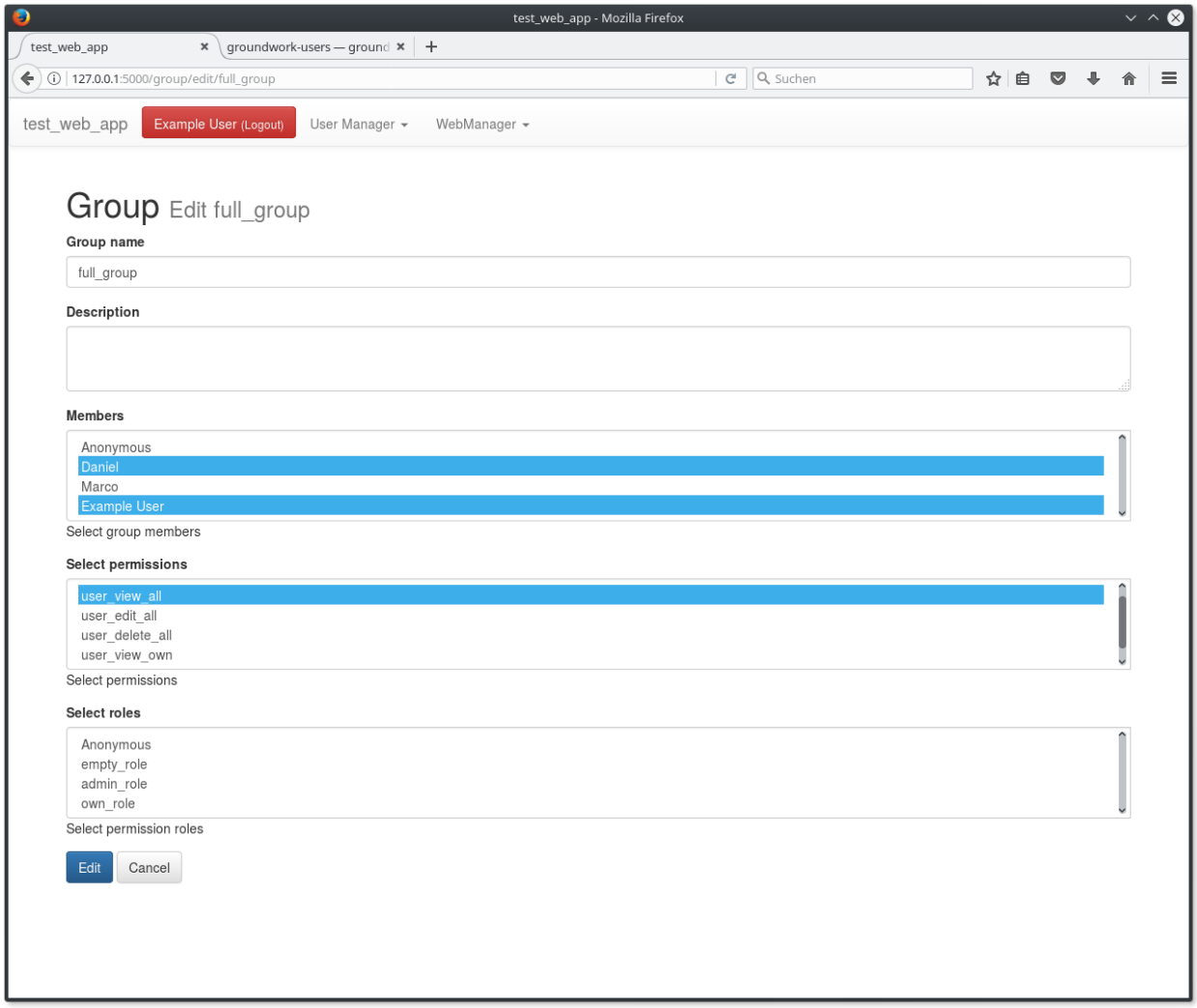

test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/user/marco
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
User Marco
Edit user
Remove
User
marco
Full name
Marco
E-Mail
mh@useblocks.com
Description
Domain
full_domain
Page
Active
False
Last login
None
Groups
empty_group
Roles
empty_role
Permissions
Permissions by roles
empty_role
Permissions by groups
Directly mapped permissions
API keys
4c595b04-df2e-44e6-a7e8-108a360bba17
a95b29b8-4a39-405e-9ed1-56b8a354f98e


test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/user/edit/marco
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Usermanager
Email Address
mh@useblocks.com
User name
marco
Full name
Marco
Domain
full_domain
Select domain
Active
Is user active and can be used for logins?
Select groups
full_group
empty_group
Select groups
Select roles
Anonymous
empty_role
admin_role
own_role
Select permission roles
Select permissions
user_view_all
user_edit_all
user_delete_all
user_view_own
Select permissions
Description

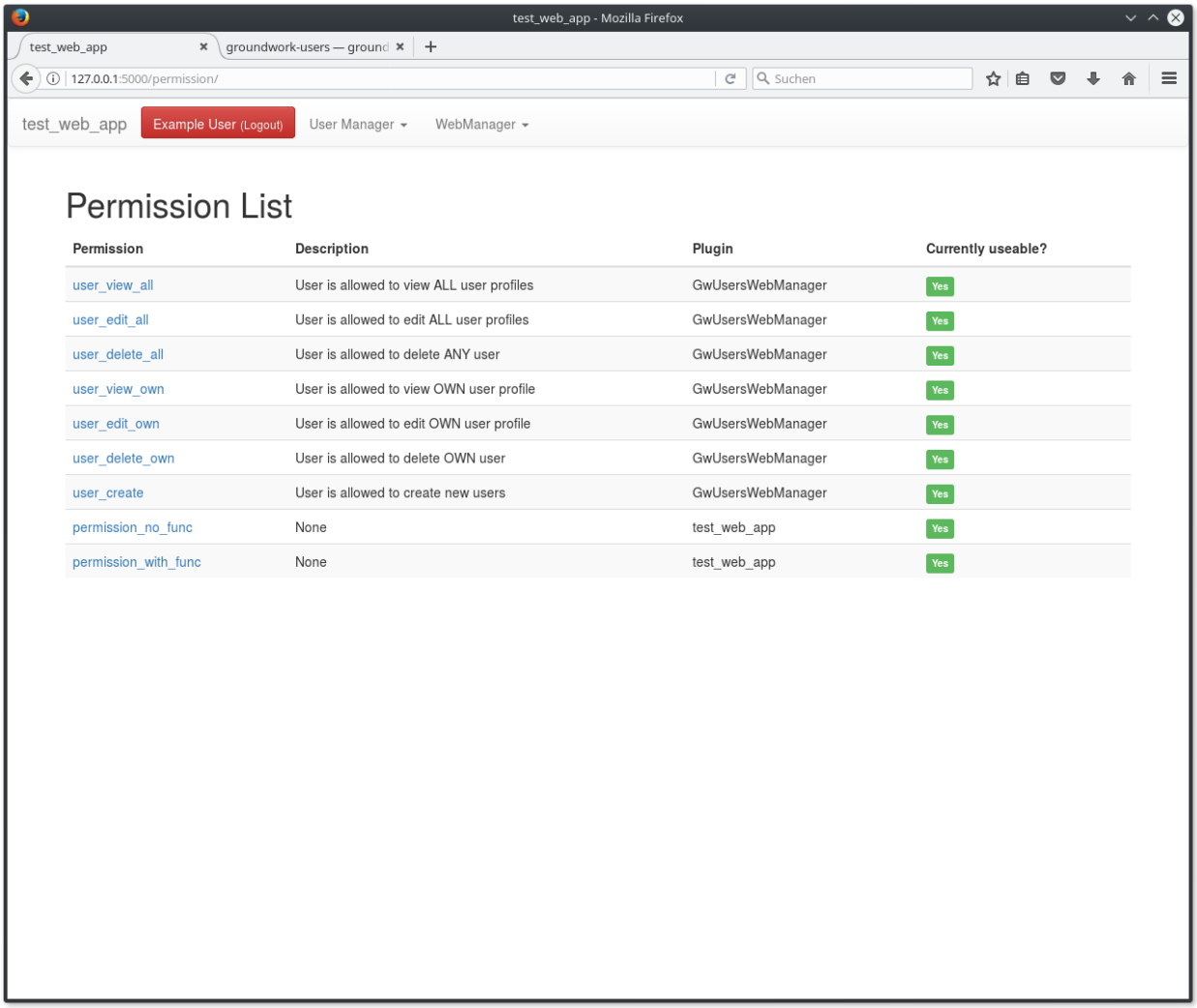
Groups

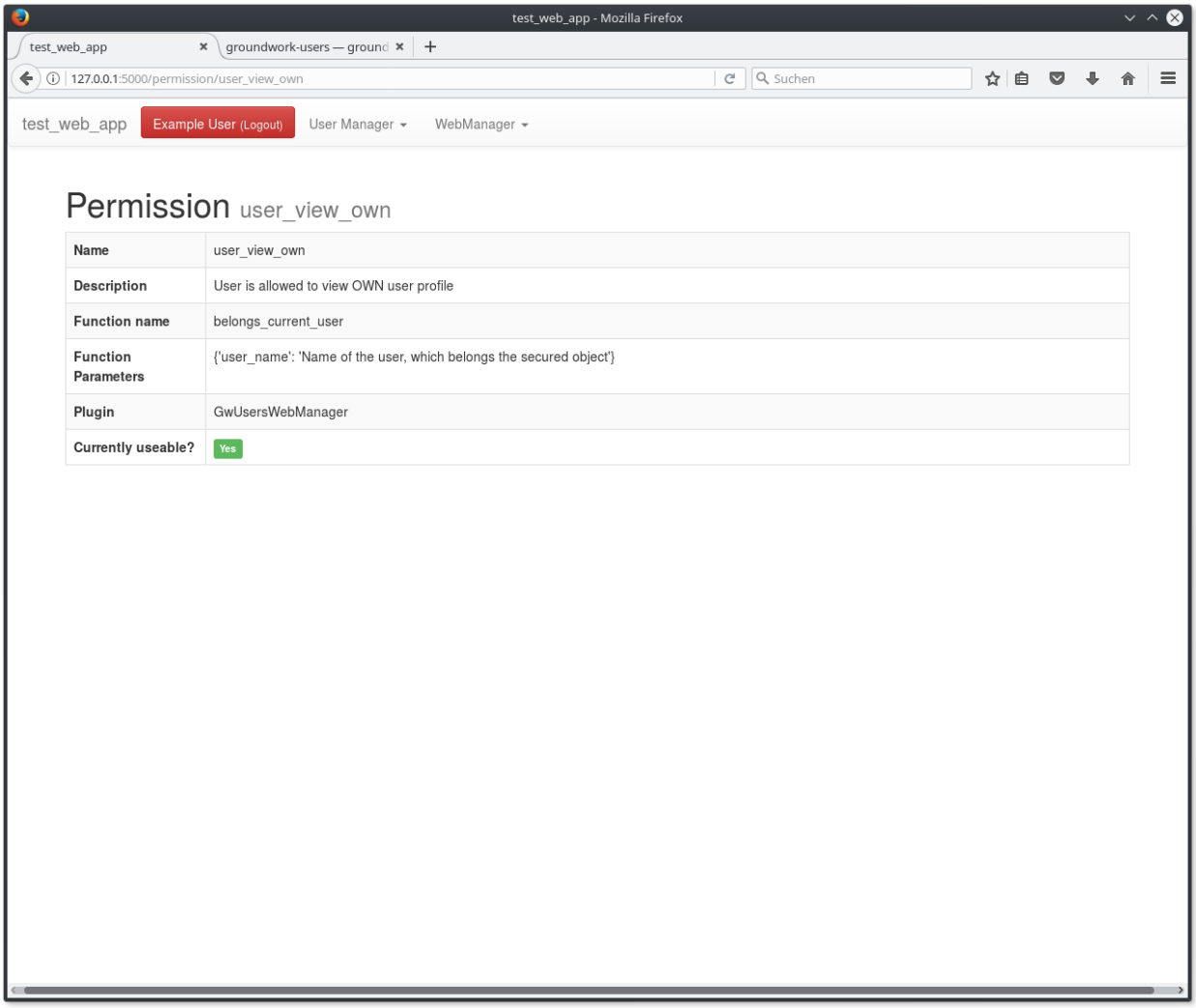

test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/group/
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Group List
Create group
Group
Description
Amount Users
full_group
2
empty_group
None
1


test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/group/full_group
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Group full_group
Edit group
Remove
Name
full_group
Description
Users
Daniel
Example User
Roles
Permissions
user_view_all
Permissions by roles

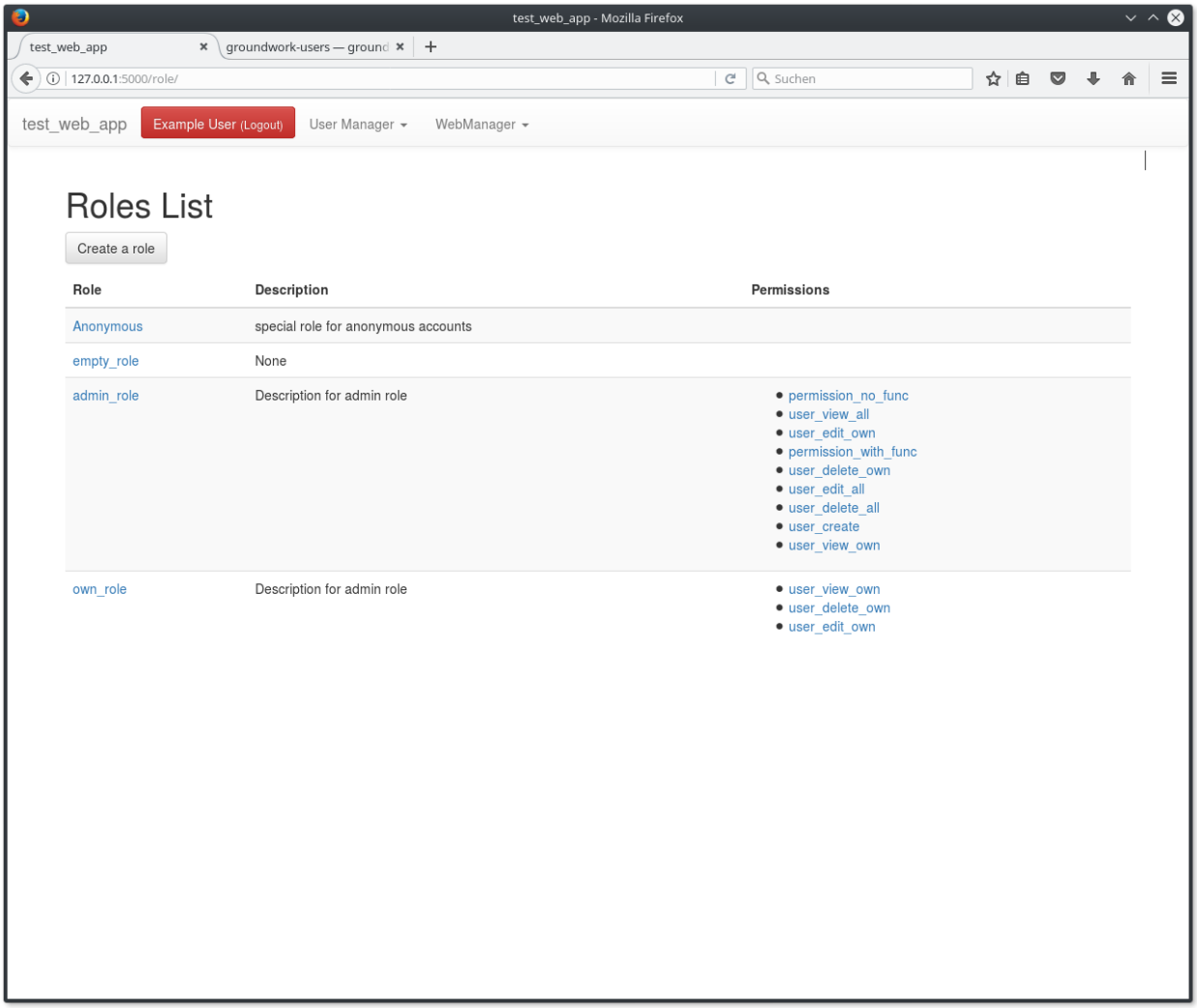

test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/group/edit/full_group
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Group Edit full_group
Group name
full_group
Description
Members
Anonymous
Daniel
Marco
Example User
Select group members
Select permissions
user_view_all
user_edit_all
user_delete_all
user_view_own
Select permissions
Select roles
Anonymous
empty_role
admin_role
own_role
Select permission roles
Edit
Cancel

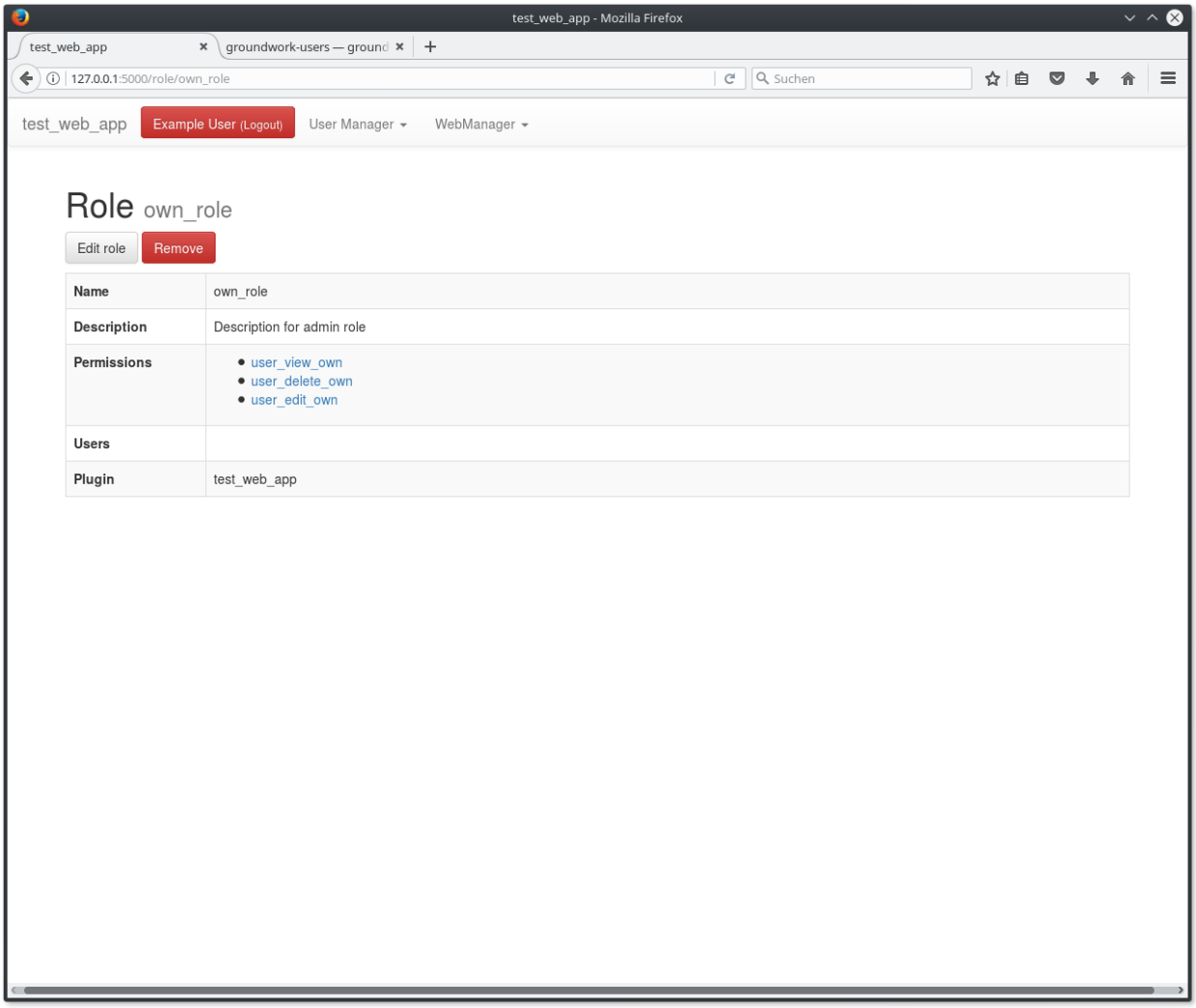
Permissions

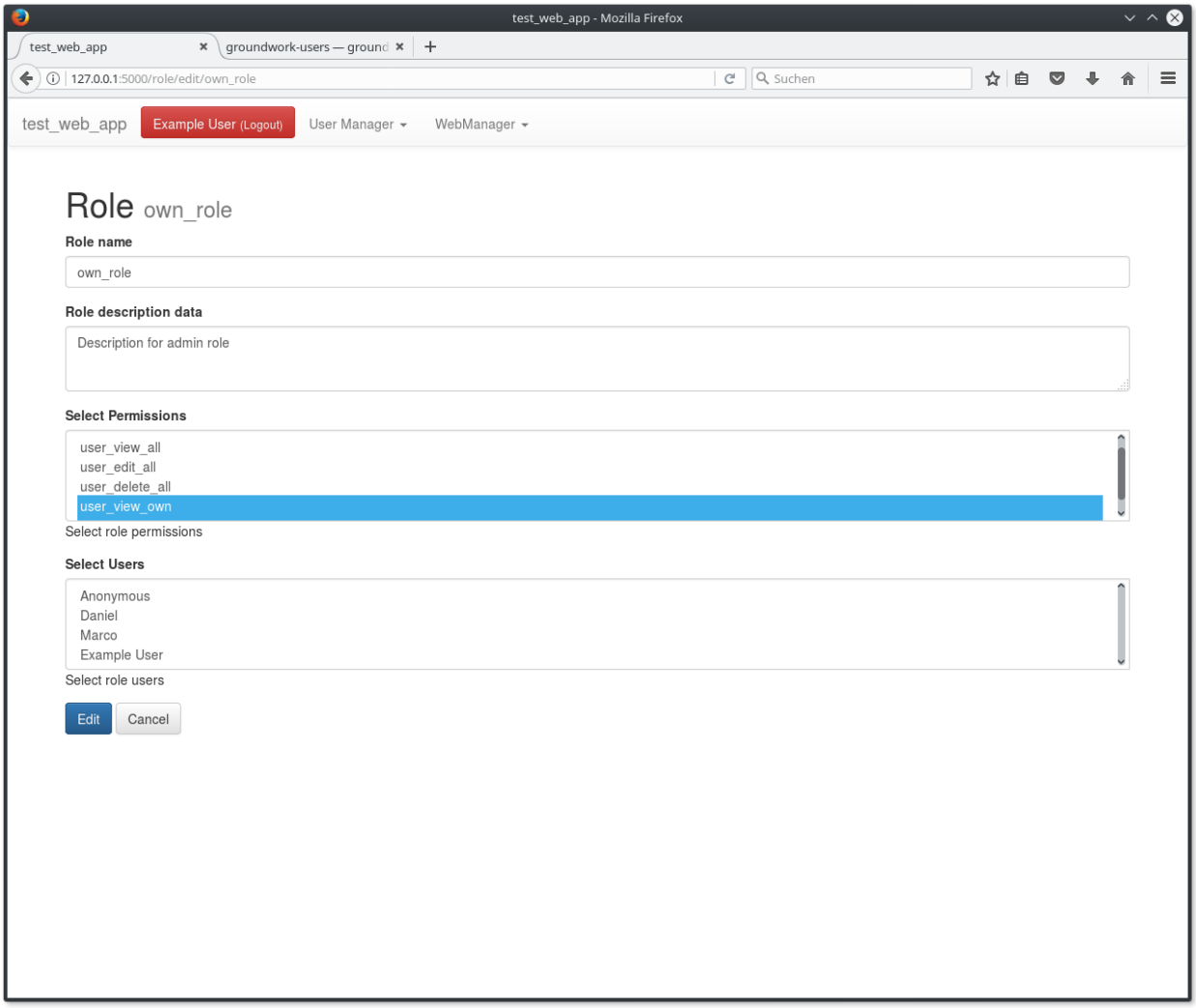



test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/permission/user_view_own
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Permission user_view_own
Name
user_view_own
Description
User is allowed to view OWN user profile
Function name
belongs_current_user
Function Parameters
{'user_name': 'Name of the user, which belongs the secured object'}
Plugin
GwUsersWebManager
Currently useable?
Yes

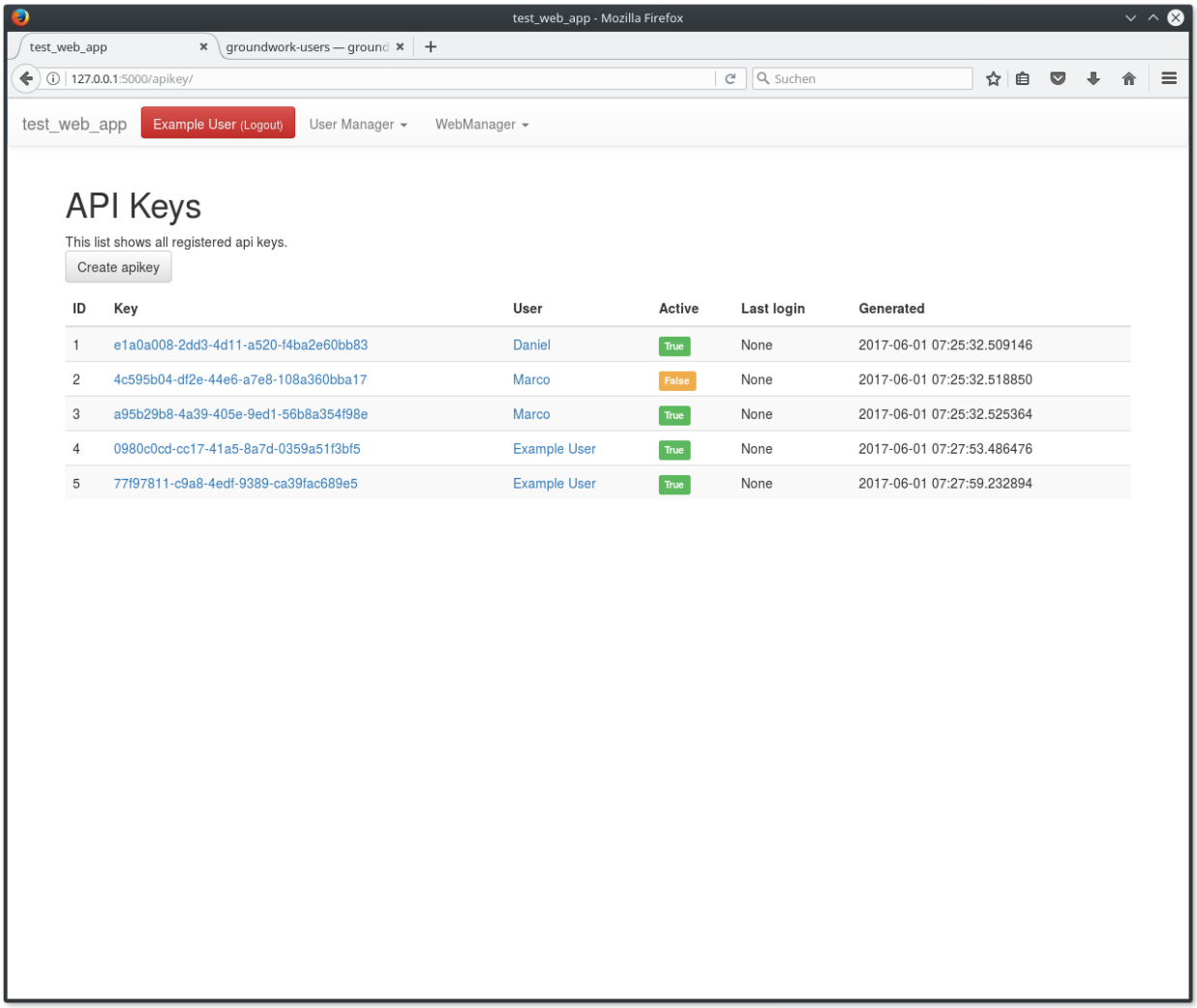
Roles

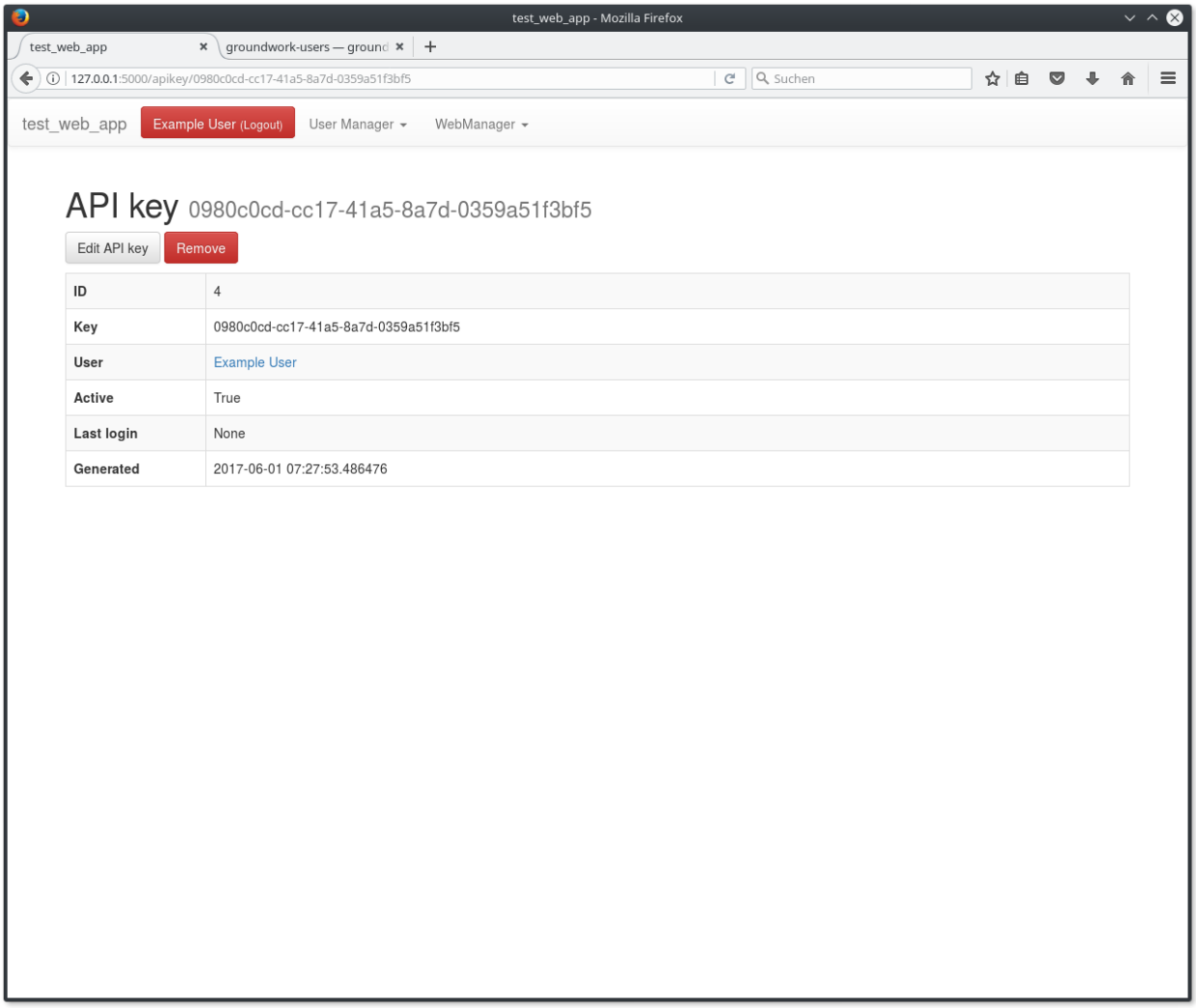

test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/role/
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Roles List
Create a role
Role
Description
Permissions
Anonymous
special role for anonymous accounts
empty_role
None
admin_role
Description for admin role
permission_no_func
user_view_all
user_edit_own
permission_with_func
user_delete_own
user_edit_all
user_delete_all
user_create
user_view_own
own_role
Description for admin role
user_view_own
user_delete_own
user_edit_own

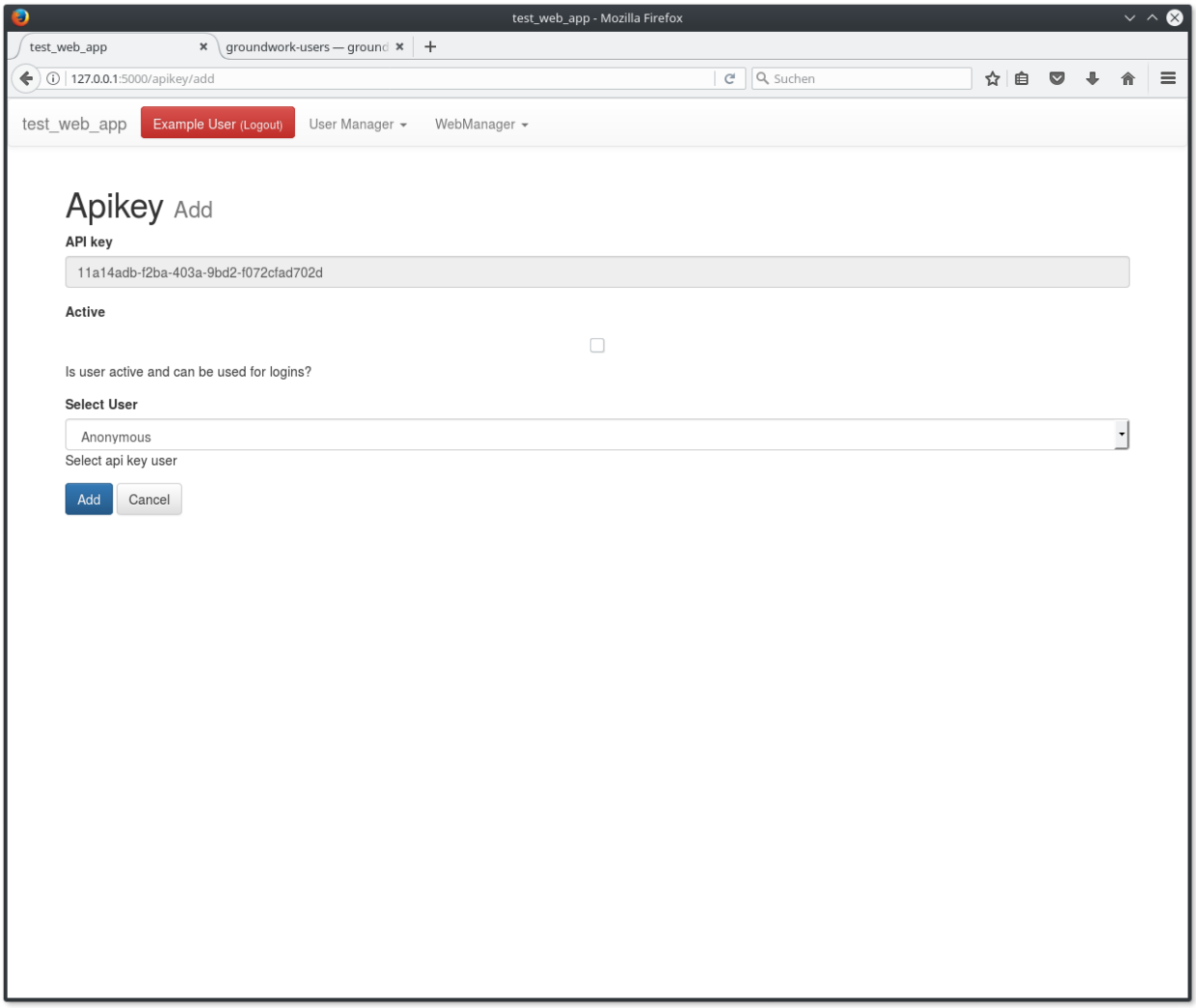

test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/role/own_role
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Role own_role
Edit role
Remove
Name
own_role
Description
Description for admin role
Permissions
user_view_own
user_delete_own
user_edit_own
Users
Plugin
test_web_app


test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/role/edit/own_role
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
Role own_role
Role name
own_role
Role description data
Description for admin role
Select Permissions
user_view_all
user_edit_all
user_delete_all
user_view_own
Select role permissions
Select Users
Anonymous
Daniel
Marco
Example User
Select role users
Edit
Cancel

API keys


test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/apikey/
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
API Keys
This list shows all registered api keys.
Create apikey
ID
Key
User
Active
Last login
Generated
1
e1a0a008-2dd3-4d11-a520-f4ba2e60bb83
Daniel
True
None
2017-06-01 07:25:32.509146
2
4c595b04-df2e-44e6-a7e8-108a360bba17
Marco
False
None
2017-06-01 07:25:32.518850
3
a95b29b8-4a39-405e-9ed1-56b8a354f98e
Marco
True
None
2017-06-01 07:25:32.525364
4
0980c0cd-cc17-41a5-8a7d-0359a51f3bf5
Example User
True
None
2017-06-01 07:27:53.486476
5
77f97811-c9a8-4edf-9389-ca39fac689e5
Example User
True
None
2017-06-01 07:27:59.232894


test_web_app - Mozilla Firefox
test_web_app
groundwork-users — ground
127.0.0.1:5000/apikey/0980c0cd-cc17-41a5-8a7d-0359a51f3bf5
Suchen
test_web_app
Example User (Logout)
User Manager
WebManager
API key 0980c0cd-cc17-41a5-8a7d-0359a51f3bf5
Edit API key
Remove
ID 4
Key 0980c0cd-cc17-41a5-8a7d-0359a51f3bf5
User Example User
Active True
Last login None
Generated 2017-06-01 07:27:53.486476



Web views

Using web views

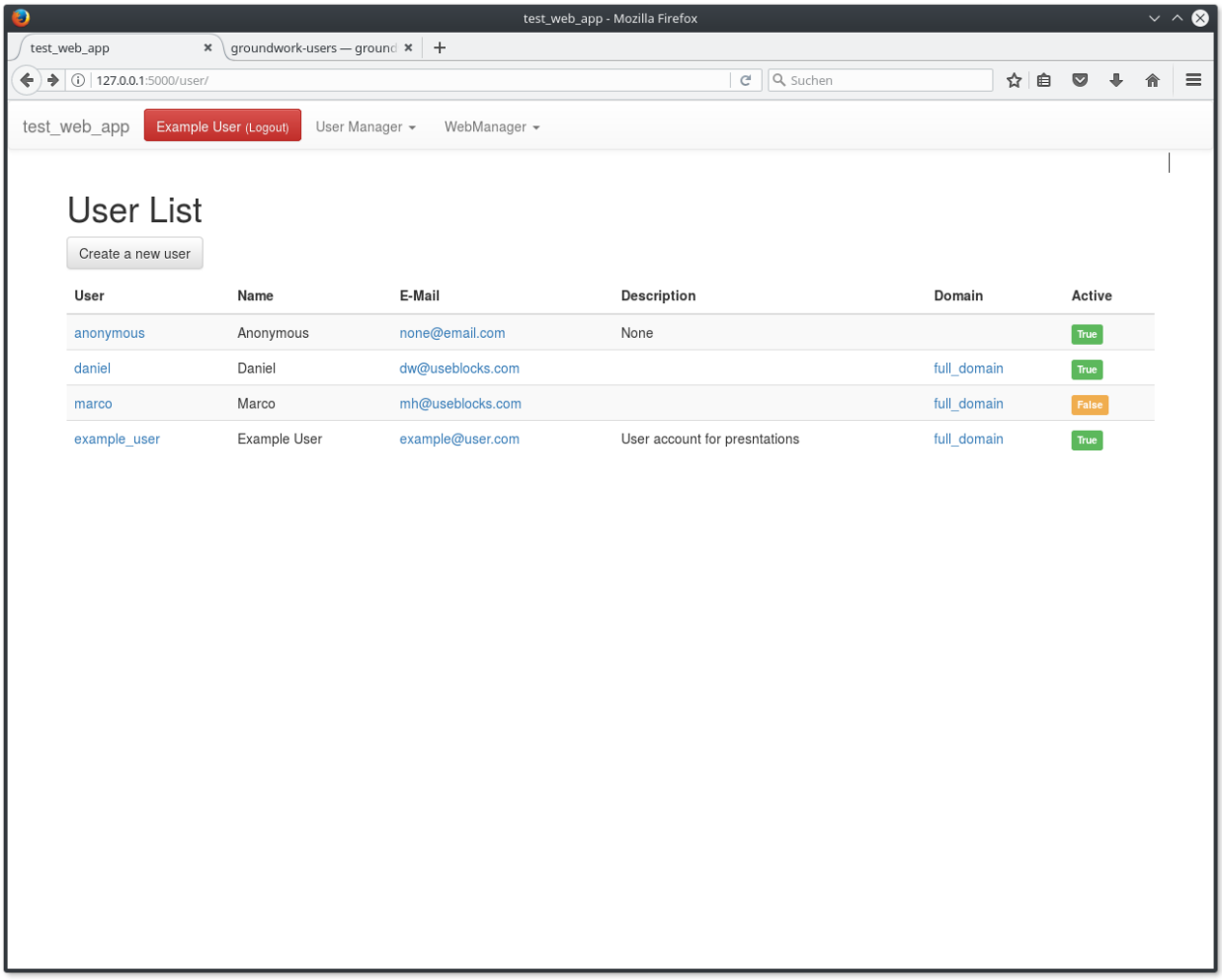
By using the plugin *GwUsersWebManager* you get automatically web views and menu entries to view, create, edit and delete users and their related objects.

All you have to do is to load and activate *GwUsersWebManager*:

```
from groundwork import App

def start_app():
    app = App(["my_config_file"])
    app.plugins.activate(["GwUsersWebManager", "Another_Plugin", "One_More_Plugin"])

    # Register a new user on application level
    user = app.users.register("test_user", "user@test.com", "my_password")

if "main" in __name__:
```

```
    start-app()
```

After starting your application, you should be able to see a list of users under the url **http://your_server/user**



Set permisisons

Before you can start creating and editing users, you have to be sure the current_user has the right permission to perform these actions.

GwUsersWebManager creates some permission during activations and secures the related views and buttons:

```
* user_view_all
* user_edit_all
* user_delete_all
* user_view_own
* user_edit_own
* user_delete_own
```

xx_all allows to perform the action on **all** user objects.

xx_own allows the currently logged in user to perform the related action on its own user profile only.

So to finally create a user via web-interface, you already need an existing user with the right permissions.

But that's quite easy to achieve, by creating some sort of an "administrator" user during application start up:

```
from groundwork import App

def start_app():
    app = App(["my_config_file"])
    app.plugins.activate(["GwUsersWebManager", "Another_Plugin", "One_More_Plugin"])

    # Check if an "admin" user already exists
    admin_user = app.users.get("admin")
    if admin_user is None or len(admin_user) == 0:
        # Get all existing permissions because our new user is admin. Muhaaaa!
        admin_permissions = app.permission.get()

        # Register admin user
        admin_user = app.users.register("admin", "admin@my_company.com", "admin_
↪password",
                                        permissions=admin_permissions)

if "main" in __name__:
    start-app()
```

After you started your application you can login with the admin user and have all available permissions to perform really every action.

Users

To register, edit or delete users and their related data, your plugin needs to inherit from *GwUsersPattern*

Register a new user

```
from groundwork_users.patterns import GwUsersPattern

class MyUserPlugin(GwUsersPattern):
    # Plugin initialisation, no user related stuff here
    def __init__(self, app, *args, **kwargs):
        self.name = "MyUserPlugin
        super(MyUserPlugin, self).__init__(app, *args, **kwargs)

    def activate(self):
        # Register a new user
        user = self.users.register("test_user", "user@test.com", "my_password")
```

If you already have registered some permissions, roles, *Groups* or domains you can use them during the registration process:

```
class MyUserPlugin(GwUsersPattern):
    ...

    def activate(self, ):
        user_roles = self.roles.get("my_role")
        user_permissions = self.permission.get("my_permission")
        user_groups = self.roles.get("my_group")
        user_domain = self.domains.get("my_domain")
```

```
        user = self.users.register(user_name="new_user",
                                   full_name="New User",
                                   email="new@user.com",
                                   password="secret_password",
                                   page="htttp://user_page.com",
                                   description="My new user for tests",
                                   domain=users_domain,
                                   groups=user_groups,
                                   roles=user_roles,
                                   permissions=user_permissions
                                   )
```

Update an existing user

To update an existing user you can use the Users database model and commit the change to the database after changes has been made:

```
class MyUserPlugin(GwUsersPattern):
    ...

    def change_user(self, user_name):
        user = self.users.get(user_name)
        if user is None:
            self.log.error("User {0} does not exist".format(user_name))
            return

        # Make a change on the user model
        user.full_name = "My new user name"

        # Commit the change
        self.users_db.commit()
```

Deleting a user

```
class MyUserPlugin(GwUsersPattern):
    ...

    def deactivate(self):
        # Let's delete our test user from database, if the plugin gets deactivated
        self.users.delete("new_user")
```

Getting users

For searching and filter users you can use the *get()* function:

```
class MyUserPlugin(GwUsersPattern):
    ...

    def activate(self):
        # Get a user by user_name
        users = self.users.get("new_user")
        if users is not None:
            user = users[0]
```

Note: `get()` always returns a list, if users were found or None, if no user was found. Even if only a single user is found a list is returned!

You can use additional key-word arguments to filter for users. Each given keyword is passed to the sqlalchemy filter function and therefore must be part of the user database model:

```
# Filter by full name
users = self.users.get(full_name="user")

# Filter by active status
users = self.users.get(active = True)

# Filter my multiple values
roles = self.roles.get("my_role")
users = self.users.get(active = True, roles=roles)
```

User database model

Below you can see the currently used database model for a user object:

```
    class User(Base, UserMixin):
        """
        Class/Table for storing single users.
        The following columns are needed by the flask-security extension:
        * confirmed_at
        * last_login_at
        * current_login_at
        * last_login_ip
        * current_login_ip
        * login_count
        """
        __tablename__ = "user"
        id = Column(Integer, primary_key=True)
        email = Column(Text(255), unique=True)
        full_name = Column(Text(255))
        password = Column(Text(255))
        user_name = Column(Text(255), unique=True)
        description = Column(Text(2048))
        page = Column(Text(255))
        plugin_name = Column(Text(255))
        active = Column(Boolean())
        confirmed_at = Column(DateTime())
        last_login_at = Column(DateTime())
        current_login_at = Column(DateTime())
        last_login_ip = Column(Text(255))
        current_login_ip = Column(Text(255))
        login_count = Column(Integer)
        roles = relationship('Role', secondary=roles_users,
                             backref=backref('users', lazy='dynamic'))
        permissions = relationship('Permission', secondary=permissions_users,
                                   backref=backref('users', lazy='dynamic'))
        domain_id = Column(Integer, ForeignKey('domain.id'))
        domain = relationship("Domain", backref="users")
```

Groups

groups are used to bundle users, so that you can assign permissions and roles easily to a group instead of assigning them separately to each user.

To register, edit or delete groups and their related data, your plugin needs to inherit from GwUsersPattern

Register a new group

```
from groundwork_users.patterns import GwUsersPattern

class MyGroupPlugin(GwUsersPattern):
    # Plugin initialisation, no group related stuff here
    def __init__(self, app, *args, **kwargs):
        self.name = "MyGroupPlugin
        super(MyUserPlugin, self).__init__(app, *args, **kwargs)

    def activate(self):
        # Register a new group
        self.groups.register("my_group", "my own group description")
```

You can also assign users, permission and roles during group registration:

```
class MyGroupPlugin(GwUsersPattern):
    ...
     def activate(self):
        # Register a user
        user = self.users.register("test_user", "user@test.com", "my_password")

        # Get a permissions
        permission = self.permissions.get("my_permission")[0]

        # Register a role
        role = self.roles.register("my_role", permissions=[permission])

        # Register a new group
        self.groups.register("my_group", "my own group description",
                             users=[user],
                             permissions=[permission],
                             roles=[role])
```

Updating an existing group

```
class MyGroupPlugin(GwUsersPattern):
    ...

    def change_group(self, group_name):
        group = self.groups.get(group_name)
        if group is None:
            self.log.error("Group {0} does not exist".format(group_name))
        return

        # Make changes on the group model
        group.name = "new_name"
```

```
        new_permissions = self.permissions.get("some_new_permissions")
        group.permissions = group.permissions + new_permissions  # Add the new␣
↪permissions to the existing ones

        # Commit the change
        self.users_db.commit()
```

Deleting a group

```
class MyGroupPlugin(GwUsersPattern):
    ...
    def deactivate(self):
        self.groups.delete("my_group")
```

Getting groups

```
class MyGroupPlugin(GwUsersPattern):
    ...

    def activate(self):
        # Get a group by name
        groups = self.groups.get("my_group")
        if groups is not None:
            group = groups[0]
```

Note: *get()* always returns a list, if groups were found or None, if no group was found. Even if only a single group is found a list is returned!

You can use additional key-word arguments to filter for groups. Each given keyword is passed to the sqlalchemy filter function and therefore must be part of the user database model:

```
# Filter by plugin_name, which has registered the group
groups = self.groups.get(plugin_name="MyGroupPlugin")

# Filter by an user
users = self.users.get("my_user")
groups = self.groups.get(users=[users])
```

Groups database model

Below you can see the currently used database model for a group object:

```
    class Group(Base):
        """
        Class/Table for storing groups.
        """
        __tablename__ = "group"
        id = Column(Integer(), primary_key=True)
        name = Column(Text(255), unique=True)
```

```
        description = Column(Text(2048))
        plugin_name = Column(Text())
        users = relationship('User', secondary=groups_users,
                             backref=backref('groups', lazy='dynamic'))
        roles = relationship('Role', secondary=roles_groups,
                             backref=backref('groups', lazy='dynamic'))
        permissions = relationship('Permission', secondary=permissions_groups,
                                   backref=backref('groups', lazy='dynamic'))
```

JINJA Templates

Check permission

You can easily check the permission status of the current user:

```
{% if current_user.has_permission("user_edit_all") %}
<a class="btn btn-default" href="{{url_for('.add')}}">
    {{_("Create a new user")}}</a>
{% endif %}
```

The above code will show a button “Create a new user” only, if the current logged in user has the permission “user_create”.

Configuration

Application configuration parameters

USERS_DB_URL

Database connection string / url, which defines the database to use for storing user related data.:

```
USERS_DB_URL = "sqlite://{0}/my_users.db".format(APP_PATH)
```

API

Plugins

GwUsersWebManager

class groundwork_users.plugins.**GwUsersWebManager**(*app*, **args*, ***kwargs*)

Patterns

GwUsersPattern

class groundwork_users.patterns.**GwUsersPattern**(*app*, **args*, ***kwargs*)

users = None
Instance of *UsersPlugin*. Provides functions to register and manage users

Users

The following functions are available inside each plugin, which inherits from `GwUsersPattern` via `self.users`.

class groundwork_users.patterns.gw_users_pattern.users.**UsersPlugin**(*plugin*)

delete(*user_name*)

get(*user_name=None*, ***kwargs*)

register(*user_name*, *email*, *password*, *full_name=''*, *page=None*, *description=None*, *domain=None*, *groups=None*, *roles=None*, *permissions=None*, *confirmed_at=None*, *active=True*)

class groundwork_users.patterns.gw_users_pattern.users.**UsersApplication**(*app*)

delete(*user_name*, *plugin=None*)

get(*user_name=None*, *plugin=None*, ***kwargs*)

register(*user_name*, *email*, *password*, *full_name=None*, *page=None*, *description=None*, *plugin=None*, *domain=None*, *groups=None*, *roles=None*, *permissions=None*, *confirmed_at=None*, *active=True*)

Errors

class groundwork_users.patterns.gw_users_pattern.users.**NoUserDatabaseException**

class groundwork_users.patterns.gw_users_pattern.users.**NoUserTableException**

class groundwork_users.patterns.gw_users_pattern.users.**UserDoesNotExistException**

Groups

The following functions are available inside each plugin, which inherits from `GwUsersPattern` via `self.groups`.

class groundwork_users.patterns.gw_users_pattern.groups.**GroupsPlugin**(*plugin*)

delete(*group_name*)

get(*group_name=None*)

register(*name*, *description=None*, *users=None*, *permissions=None*, *roles=None*)

class groundwork_users.patterns.gw_users_pattern.groups.**GroupsApplication**(*app*, *users_db*)

delete(*group_name*, *plugin=None*)

get(*group_name=None*, *plugin=None*, ***kwargs*)

register(*name*, *description=None*, *users=None*, *permissions=None*, *roles=None*, *plugin=None*)

Index