CONSUL DEMOCRACY - Documentation

Technical Documentation

ntroduction

CONSUL DEMOCRACY logo

CONSUL DEMOCRACY

Citizen Participation and Open Government Application

CONSUL DEMOCRACY Project main website

You can access the main website of the project at http://consuldemocracy.org where you can find documentation about the use of the platform, videos, and links to the community space.

Configuration for development and test environments

NOTE: For more detailed instructions, check the local installation docs.

Prerequisites: install git, Ruby 3.2.8, CMake, pkg-config, Node.js 18.20.3, ImageMagick and PostgreSQL (>=9.5).

Note: The `bin/setup` command below might fail if you've configured a username and password for PostgreSQL. If that's the case, edit the lines containing `username:` and `password:` (adding your credentials) in the `config/database.yml` file and run `bin/setup` again.

Copy

```
git clone https://github.com/consuldemocracy/consuldemocracy.git

cd consuldemocracy
```

```
bin/setup

bin/rake db:dev_seed
```

Run the app locally:
Copy
```
bin/rails s
```

You can run the tests with:
Copy
```
bin/rspec
```

Note: running the whole test suite on your machine might take more than an hour, so it's strongly recommended that you setup a Continuous Integration system in order to run them using parallel jobs every time you open or modify a pull request (if you use GitHub Actions or GitLab CI, this is already configured in `.github/workflows/tests.yml` and `.gitlab-ci.yml`) and only run tests related to your current task while developing on your machine. When you configure the application for the first time, it's recommended that you run at least one test in `spec/models/` and one test in `spec/system/` to check your machine is properly configured to run the tests.

You can use the default admin user from the seeds file:

user: admin@consul.dev**pass:** 12345678

But for some actions like voting, you will need a verified user, the seeds file also includes one:

user: verified@consul.dev**pass:** 12345678

Configuration for production environments

See installer

License

Code published under AFFERO GPL v3 (see LICENSE-AGPLv3.txt)

Contributions

See CONTRIBUTING.md
NextGetting started

Create your fork

Consul Democracy git repo is hosted at Github.com, we recommend using it for your fork's repo to make things easier. But you can use any other service like Bitbucket or Gitlab if you want to, just don't forget to put a reference link back to Consul Democracy on the footer to comply with project's license (AGPLv3).

Register a user account on Github if you don't have one.
Create an Organization on Github with the name of your city or the organization that's going to use Consul Democracy. **This is not mandatory**, but it will help understand the fork's purpose and future contributions by other users.
Fork Consul Democracy using the **fork** button on the top right corner at https://github.com/consuldemocracy/consuldemocracy
Clone your fork repository on to your computer.

Why make code public?

We strongly recommend making code public for multiple reasons:

Transparency: It should be part of the culture of public entities that adopt Consul Democracy, as well as any organization or group.
Support: If you need technical help, both community and Consul Democracy core team will be able to understand and advice by easily seeing involved code.
Collaboration: By other professionals, citizens, etc...
Last but not least, Consul Democracy is distributed under the **AGPLv3 license** that commands to publish source code.

Getting started

Configure your fork

Continuous Integration with GitHub Actions

GitHub Actions is a tool integrated into GitHub that allows you to automate tasks such as running tests every time you make a change to your code. Since Consul Democracy already includes predefined configurations for GitHub Actions, enabling continuous integration in your fork is very straightforward.

Steps to enable GitHub Actions

Enable GitHub Actions in your fork:

Once you have created the fork, go to the "**Actions**" tab in your GitHub repository.

You will see a message that says: "Workflows aren't being run on this forked repository." This is normal because GitHub disables workflows by default in newly forked repositories for security reasons.

Click the "**I understand my workflows, go ahead and enable them**" button to enable workflows in your fork.

Verify the configuration:

Make a change to any project file (for example, edit a `.md` file) in a branch other than master and push it to your fork.

Open a pull request from the new branch to master in your fork.

Go to the "**Actions**" tab and verify that the tests are running correctly based on the workflows defined in the project's `.github/workflows/` directory.

Getting started

Keep your fork updated

Preliminary steps

Running your test suite

Before upgrading to a newer version of Consul Democracy, make sure you've configured your fork to run the tests, and that all the tests in your test suite pass. If you omit this step, it'll be much harder to upgrade since it'll be very tricky to know whether some features are broken after the upgrade.

Please take as much time as necessary for this; every hour you spend making sure your test suite is in good shape will save countless hours of fixing bugs that made it to production.

Configuring your git remotes

If you created your fork following the instructions in the Create your fork section and cloned it locally, run:

Copy

```
git remote -v
```

You should get something like:

```
origin git@github.com:your_user_name/consuldemocracy.git (fetch)
origin git@github.com:your_user_name/consuldemocracy.git (push)
```

Now, add Consul Democracy's GitHub as upstream remote with:

Copy

```
git remote add upstream
git@github.com:consuldemocracy/consuldemocracy.git
```

Check it's been correctly configured by once again running:
Copy
```
git remote -v
```

This time you should get something like:
origin git@github.com:your_user_name/consuldemocracy.git (fetch)
origin git@github.com:your_user_name/consuldemocracy.git (push)
upstream git@github.com:consuldemocracy/consuldemocracy.git (fetch)
upstream git@github.com:consuldemocracy/consuldemocracy.git (push)

Pulling changes from Consul Democracy

When upgrading Consul Democracy, **it is extremely important that you upgrade one version at a time**. Some versions require running certain tasks that modify existing content in the database in order to make it compatible with future versions. Upgrading two versions at the same time could result in irreversible data loss.

When we say "one version at a time", we're excluding patch versions. For example, if you're currently using version 1.2.0 and you'd like to upgrade to version 2.2.2, first upgrade to version 1.3.1, then to 1.4.1, then to 1.5.0, then to 2.0.1, then to 2.1.1 and then to version 2.2.2. That is, always upgrade to the latest patch version of a certain major/minor version.

When upgrading to a newer version, make sure you read the release notes of that version **before** upgrading.

To upgrade, start by creating a branch named `release` from your `master` branch to apply the changes in the new version of Consul Democracy:
Copy
```
git checkout master

git pull

git checkout -b release

git fetch upstream tag <version_tag_you_are_upgrading_to>
```

Now you're ready to merge the changes in the new version.

Merging changes

Run:
Copy
```
git merge <version_tag_you_are_upgrading_to>
```

After running this command, there are two possible outcomes:

A. You get a screen on your git configured editor showing the commit message `Merge tag '<version_tag_you_are_upgrading_to>' into release`. That means git was able to apply the changes in the new version and can merge them without conflicts. Finish the commit.

B. You get some git errors along with a `Automatic merge failed; fix conflicts and then commit the result` message. That means there are conflicts between your custom code changes and the changes in the new version of Consul Democracy. That's the main reason we strongly recommend using custom folders and files for your custom changes; the more you use custom folders and files, the less conflicts you will get. Resolve the merge conflicts carefully and commit them, making sure you document how you solved the conflicts (for example, in the commit message).

Making sure everything works

We now recommend pushing your `release` branch to GitHub (or GitLab, if that's what you usually use) and create a pull request so you can check whether your test suite reports any issues.

It's possible that the test suite does indeed report some issues because your custom code might not correctly work with the newer version of Consul Democracy. It is **essential** that you fix all these issues before continuing.

Finally, you might want to manually check your custom code. For example, if you've customized components or views, check whether the original ERB files have changed and whether you should update your custom files in order to include those changes.

Finishing the process

After making sure everything works as expected, you can either merge the pull request on GitHub/GitLab or finish the process manually:

Copy

```
git checkout master

git merge release

git branch -d release

git push
```

Finally, read the release notes once again to make sure everything is under control, deploy to production, execute `bin/rake consul:execute_release_tasks RAILS_ENV=production` on the production server, and check everything is working properly.

Congratulations! You've upgraded to a more recent version of Consul Democracy. Your version of Consul Democracy is now less likely to have security vulnerabilities or become an impossible-to-maintain abandonware. Not only that, but this experience will make it easier for you to upgrade to a new version in the future.

We'd love to hear about your experience! You can use the discussions about releases for that (note that there are no discussions for versions prior to 2.2.0; if you've upgraded to an earlier version, please open a new discussion). This way, we'll manage to make it easier for you and everyone else to upgrade Consul Democracy next time.

Getting started

Communication

The preferred way to report any missing piece of information is opening an issue in the project's Github repo.

Before doing it, **please take some time to check the existing issues and make sure what you are about to report isn't already reported** by another person. In case someone else reported the same problem before or a similar one, and you have more details about it, you can write a comment in the issue page. A little more help can make a huge difference!

In order to write a new issue, take into account these few tips to make it easy to read and comprehend:

> Try to use a descriptive and to-the-point title.
>
> It's a good idea to include some sections -in case they're needed- such as: steps to reproduce the bug, expected behaviour/response, actual response or screenshots.
>
> Also it could be helpful to provide your operating system, browser version and installed plugins.

On the other hand, if you need a space for more informal and open interaction, we have the Discussions section. It's similar to a discussion forum within the project, where you can ask questions, make suggestions, share ideas, and have conversations that don't necessarily fit into an issue or pull request.

Installation

Local installation

Before installing Consul Democracy and having it up and running make sure you have all prerequisites installed.

> First, clone the Consul Democracy Github repository and enter the project folder:

Copy

```
git clone https://github.com/consuldemocracy/consuldemocracy.git
```

```
cd consuldemocracy
```

Install the Ruby version we need with your Ruby version manager. Here are some examples:

Copy
```
rbenv install `cat .ruby-version` # If you're using rbenv

rvm install `cat .ruby-version` # If you're using RVM

asdf install ruby `cat .ruby-version` # If you're using asdf
```

Check we're using the Ruby version we've just installed:

Copy
```
ruby -v

=> # (it should be the same as the version in the .ruby-version file)
```

Install the Node.js version we need with your Node.js version manager. If you're using NVM:

Copy
```
nvm install `cat .node-version`

nvm use `cat .node-version`
```

Copy the example database configuration file:

Copy
```
cp config/database.yml.example config/database.yml
```

Setup database credentials with your `consul` user in your new `database.yml` file

Note: this step is not necessary if you're using a database user without a password and the same username as your system username, which is the default in macOS.

Copy
```
nano config/database.yml
```

And edit the lines containing `username:` and `password:`, adding your credentials.

Install the project dependencies and create the database:

Copy
```
bin/setup
```

Run the following Rake task to fill your local database with development data:

Copy

```
bin/rake db:dev_seed
```

Check everything is fine by running the test suite

Note: running the whole test suite on your machine might take more than an hour, so it's strongly recommended that you setup a Continuous Integration system in order to run them using parallel jobs every time you open or modify a pull request (if you use GitHub Actions or GitLab CI, this is already configured in `.github/workflows/tests.yml` and `.gitlab-ci.yml`) and only run tests related to your current task while developing on your machine. When you configure the application for the first time, it's recommended that you run at least one test in `spec/models/` and one test in `spec/system/` to check your machine is properly configured to run the tests.

Copy
```
bin/rspec
```

Now you have all set, run the application:

Copy
```
bin/rails s
```

Congratulations! Your local Consul Democracy application will be running now at `http://localhost:3000`.

In case you want to access the local application as admin, a default user verified and with admin permissions was created by the seed files with **username** `admin@consul.dev` and **password** `12345678`.

If you need an specific user to perform actions such as voting without admin permissions, a default verified user is also available with **username** `verified@consul.dev` and **password** `12345678`.

Ubuntu Linux

System update

Run a general system update:

Copy
```
sudo apt update
```

Git

Git is officially maintained in Ubuntu:

Copy
```
sudo apt install git
```

Ruby version manager

Ruby versions packaged in official repositories are not suitable to work with Consul Democracy, so we'll have to install it manually.

First, we need to install Ruby's development dependencies:
Copy
```
sudo apt install libssl-dev autoconf bison build-essential libyaml-dev libreadline-dev zlib1g-dev libncurses-dev libffi-dev libgdbm-dev
```

The next step is installing a Ruby version manager, like rbenv:
Copy
```
wget -qO- https://github.com/rbenv/rbenv-installer/raw/main/bin/rbenv-installer | bash

source ~/.bashrc
```

CMake and pkg-config

In order to compile some of the project dependencies, we need CMake and pkg-config:
Copy
```
sudo apt install cmake pkg-config
```

Node.js version manager

To compile the assets, you'll need a JavaScript runtime. Node.js is the preferred option. To install Node.js, we will install a Node.js version manager, like NVM:
Copy
```
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh | bash

source ~/.bashrc
```

PostgreSQL

Install postgresql and its development dependencies with:
Copy
```
sudo apt install postgresql libpq-dev
```

You also need to configure a user for your database. As an example, we'll choose the username "consul":

Copy

```
sudo -u postgres createuser consul --createdb --superuser --pwprompt
```

Imagemagick

Install Imagemagick:

Copy

```
sudo apt install imagemagick
```

Chrome or Chromium

In order to run the system tests, we need to install Chrome or Chromium.

Copy

```
sudo apt install chromium-browser
```

Now you're ready to go get Consul Democracy installed!

Docker

Prerequisites

Docker and Docker Compose should be installed on your machine. The installation process depends on your operating system.

macOS

You can follow the official Docker installation documentation.

Or, if you have homebrew installed, you can just:

Copy

```
brew install docker

brew install docker-compose

brew install --cask docker

open -a docker
```

You'll be asked to give permissions to the Docker app and type your password; then, you're set.

Linux

Install Docker and Docker Compose. For example, in Ubuntu 22.04:

Copy
```
sudo apt-get install docker.io docker-compose-v2
```

Windows

The official Docker documentation includes a page with instructions to install Docker Desktop on Windows. From there, download Docker Desktop for Windows and execute it.

Installation

Clone the repo on your computer and enter the folder:
Copy
```
git clone https://github.com/consuldemocracy/consuldemocracy.git

cd consuldemocracy
```

Then create the secrets and database config files based on the example files:
Copy
```
cp config/secrets.yml.example config/secrets.yml

cp config/database.yml.example config/database.yml
```

Then build the image with:
Copy
```
POSTGRES_PASSWORD=password docker-compose build
```

And create the containers:
Copy
```
POSTGRES_PASSWORD=password docker-compose create
```

You can now initialize your development DB and populate it with:
Copy
```
POSTGRES_PASSWORD=password docker-compose run app rake db:create
db:migrate

POSTGRES_PASSWORD=password docker-compose run app rake db:dev_seed
```

Running Consul Democracy in development with Docker

Now we can finally run the application with:
Copy
```
POSTGRES_PASSWORD=password docker-compose up
```

And you'll be able to access it by opening your browser and visiting http://localhost:3000.

Additionally, if you'd like to run the rails console:
Copy
```
POSTGRES_PASSWORD=password docker-compose run app rails console
```

To verify the containers are up execute:
Copy
```
docker ps
```

You should see an output similar to this:
Copy
```
CONTAINER ID   IMAGE                  COMMAND
CREATED          STATUS          PORTS      NAMES

603ec83b78a6   consuldemocracy-app   "./docker-entrypoint…"   23
seconds ago   Up 22 seconds
consuldemocracy-app-run-afb6d68e2d99

d57fdd9637d6   postgres:13.16        "docker-entrypoint.s…"   50
minutes ago   Up 22 seconds   5432/tcp
consuldemocracy-database-1
```

Running tests with RSpec

Consul Democracy includes more than 6000 tests checking the way the application behaves. While we recommend you configure your fork to use a continuous integration system to run the whole test suite and verify that the latest changes don't break anything, while developing you probably want to run the tests related to the code you're working on.

First, setup the database for the test environment:
Copy
```
POSTGRES_PASSWORD=password docker-compose run app bundle exec rake
db:test:prepare
```

Then you can run tests using RSpec. For example, to run the tests for the proposal model:
Copy
```
POSTGRES_PASSWORD=password docker-compose run app bundle exec
rspec spec/models/proposal_spec.rb
```

System tests also work out of the box, although they might fail the first time while the tool running the tests downloads the right version of Chromedriver (which is needed to run them), and only "headless" mode (with a browser running in the background) is supported, which is

the mode you'd probably use more than 95% of the time anyway. For example, to run the tests for the homepage:

Copy
```
POSTGRES_PASSWORD=password docker-compose run app bundle exec rspec spec/system/welcome_spec.rb
```

Troubleshooting

Run these commands **inside Consul Democracy's directory**, to erase all your previous Consul Democracy's Docker images and containers. Then start the Docker installation process once again.

Remove all Consul Democracy images:

Copy
```
docker-compose down --rmi all -v --remove-orphans
```

Remove all Consul Democracy containers:

Copy
```
docker-compose rm -f -s -v
```

Check whether there are still containers left:

Copy
```
docker ps -a
```

In case there are, remove each one manually:

Copy
```
docker container rm <container_id>
```

Installation

Production and staging servers

Recommended minimum system requirements

1. Production server

Supported distributions: Ubuntu 22.04, Ubuntu 24.04, Debian Bullseye or Debian Bookworm

RAM: 32GB

Processor: Quad core

Hard Drive: 20 GB

Database: Postgres

2. Staging server

Supported distributions: Ubuntu 22.04, Ubuntu 24.04, Debian Bullseye or Debian Bookworm

RAM: 16GB

Processor: Dual core

Hard Drive: 20 GB

Database: Postgres

If your city has a population of over 1,000,000, consider balancing your load using 2-3 production servers and a separate server for the database.

CONSUL DEMOCRACY Installer ![Build status on Ubuntu](https://github.com/consuldemocracy/installer/workflows/ubuntu/badge.svg)

[CONSUL DEMOCRACY](https://github.com/consuldemocracy/consuldemocracy) installer for production environments

Using [Ansible](http://docs.ansible.com/), it will install and configure the following:

- Ruby
- Rails
- Postgres
- Nginx
- Puma
- SMTP
- Memcached
- DelayedJobs
- HTTPS
- Capistrano

It will also create a `deploy` user to install these libraries

Screencast

[How to setup CONSUL DEMOCRACY for a production environment](https://youtu.be/1IvnjDuRFzw)

Prerequisities

A remote server with one of the supported distributions:

- Ubuntu 22.04 x64
- Ubuntu 24.04 x64
- Debian Bullseye x64

- Debian Bookworm x64

Access to a remote server via public ssh key without password.
The default user is `deploy` but you can [use any user](#using-a-different-user-than-deploy) with sudo privileges.

```
ssh root@remote-server-ip-address
```

Updated system package versions

```
sudo apt-get update
```

Python 3 installed in the remote server

Running the installer

The following commands must be executed in your local machine

[Install Ansible >= 2.7](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html)

Get the Ansible Playbook

```
git clone https://github.com/consuldemocracy/installer
cd installer
```

Create your local `hosts` file

```
cp hosts.example hosts
```

Update your local `hosts` file with the remote server's ip address

```
remote-server-ip-address (maintain other default options)
```

Run the ansible playbook

```
ansible-playbook -v consul.yml -i hosts
```

```

Note about old versions: if you've already used the installer before version 1.1 was released, you might need to remove your `~/.ansible` folder.

Visit remote-server-ip-address in your browser and you should see CONSUL DEMOCRACY running!

## Admin user

You can sign in to the application with the default admin user:

```
admin@consul.dev
12345678
```

## Deploys with Capistrano

To restart the server and deploy new code to the server we have to configure Capistrano.

### Screencast

[How to setup Capistrano](https://youtu.be/ZCfPz_c_H6g)

Create your [fork](https://help.github.com/articles/fork-a-repo/)

Setup locally for your [development environment](https://docs.consuldemocracy.org/tech_docs/introduction-1/local_installation)

Checkout the latest stable version:

```
git checkout origin/2.3.1 -b stable
```

Create your `deploy-secrets.yml`

```
cp config/deploy-secrets.yml.example config/deploy-secrets.yml
```

Update `deploy-secrets.yml` with your server's info

```
production:
  deploy_to: "/home/deploy/consul"
  ssh_port: "22"
```
```

```
  server1: "your_remote_ip_address"
  user: "deploy"
```

Update your `repo_url` in `deploy.rb`

```
set :repo_url, 'https://github.com/your_github_username/consuldemocracy.git'
```

Make a change in a view and push it your fork in Github

```
git add .
git commit -a -m "Add sample text to homepage"
git push origin stable
```

Deploy to production

```
branch=stable cap production deploy
```

You should now see that change at your remote server's ip address

Email configuration

Screencast

[How to setup email deliveries](https://youtu.be/9W6txGpe4v4)

Screencast update: The Installer now configures a queue to send emails asynchronously. Thus you will not see a 500 error when there is a misconfiguration, as the email is sent asyncronously and the error will be raised in the queue. To see email error logs open the rails console (`cd /home/deploy/consul/current/ && bin/rails c -e production`) and search for the last error in the queue `Delayed::Job.last.last_error`)

Update the following file in your production server:
`/home/deploy/consul/shared/config/secrets.yml`

You want to change this block of code for your production environment and use your own SMTP credentials:

```
  mailer_delivery_method: "smtp"
  smtp_settings:
    :address:              "smtp.example.com"
```

```
  :port:                "25"
  :domain:              "your_domain.com"
  :user_name:           "username"
  :password:            "password"
  :authentication:      "plain"
  :enable_starttls_auto: true
```

And restart the server running this command from your local CONSUL DEMOCRACY installation (see [Deploys with Capistrano](https://github.com/consuldemocracy/installer#deploys-with-capistrano) for details).

```
cap production deploy:restart
```

Once you setup your domain, depending on your SMTP provider, you will have to do two things:

- Update the `server_name` with your domain in `/home/deploy/consul/shared/config/secrets.yml`.
- Update the `sender_email_address` from the admin section (`remote-server-ip-address/admin/settings`)

If your SMTP provider uses an authentication other than `plain`, check out the [Rails docs on email configuration](https://guides.rubyonrails.org/action_mailer_basics.html#action-mailer-configuration) for the different authentation options.

Staging server

To setup a staging server to try things out before deploying to a production server:

Update your local `hosts` file with the staging server's ip address

```
remote-server-ip-address (maintain other default options)
```

And run the playbook with an extra var "env":

```
ansible-playbook -v consul.yml --extra-vars "env=staging" -i hosts
```

Visit remote-server-ip-address in your browser and you should now see CONSUL DEMOCRACY running in your staging server.

SSL with LetsEncrypt

Using https instead of http is an important security configuration. Before you begin, you will need to either buy a domain or get access to the configuration of an existing domain. Next, you need to make sure you have an A Record in the DNS configuration of your domain, pointing to the correponding IP address of your server. You can check if your domain is correctly configured at this url https://dnschecker.org/, where you should see your IP address when searching for your domain name.

Once you have that setup we need to configure the Installer to use your domain in the application.

First, uncomment the `domain` variable in the [configuration file](https://github.com/consuldemocracy/installer/blob/2.3.1/group_vars/all) and update it with your domain name:

```
#domain: "your_domain.com"
```

Next, uncomment the `letsencrypt_email` variable in the [configuration file](https://github.com/consuldemocracy/installer/blob/2.3.1/group_vars/all) and update it with a valid email address:

```
#letsencrypt_email: "your_email@example.com"
```

Re-run the installer:

```
ansible-playbook -v consul.yml -i hosts
```

You should now be able to see the application running at https://your_domain.com in your browser.

Configuration Variables

These are the main configuration variables:

```
# Server Timezone
timezone: Europe/Madrid

# Authorized Hosts
ssh_public_key_path: "~/.ssh/id_rsa.pub"
```

```
ansible_ssh_private_key_file: "~/.ssh/id_rsa"

#Postgresql
database_name: "consul_production"
database_user: "deploy"
database_password: "change_me"
database_hostname: "localhost"

#SMTP
smtp_address:        "smtp.example.com"
smtp_port:           25
smtp_domain:         "your_domain.com"
smtp_user_name:      "username"
smtp_password:       "password"
smtp_authentication: "plain"
```

If you are on Ubuntu and would like to use its default `sudo` group instead of `wheel`, change the `deploy_group` variable to:

```
deploy_group: sudo
```

There are many more variables available check them out [here]((https://github.com/consuldemocracy/installer/blob/2.3.1/group_vars/all))

Other deployment options

Split database from application code

The [`consul` playbook](consul.yml) creates the database on the same server as the application code. If you are using a cloud host that offers managed databases (such as [AWS RDS](https://aws.amazon.com/rds/), [Azure Databases](https://azure.microsoft.com/en-us/product-categories/databases/), or [Google Cloud SQL](https://cloud.google.com/sql/)), we recommend using that instead.

To set up the application by itself:

1. Fork this repository.
1. Specify your database credentials (see the `database_*` [group variables](group_vars/all)) in a [vault](https://docs.ansible.com/ansible/latest/user_guide/vault.html).
1. Run the [`app` playbook](app.yml) instead of the [`consul`](consul.yml) one against a clean server.

```sh
ansible-playbook -v app.yml -i hosts
```

Platform-as-a-Service (PaaS)

Aside from just using managed databases, you might also look into platform-as-a-service options (like [Azure App Service](https://azure.microsoft.com/en-us/services/app-service/) or [Google App Engine](https://cloud.google.com/appengine/)) to not have to manage a server at all.

No root access

By default the installer assumes you can log in as `root`. The `root` user will only be used once to login and create a `deploy` user. The `deploy` user is the one that will actually install all libraries and is the user that must be used to login to the server to do maintenance tasks.

If you do not have `root` access, you will need your system administrator to grant you sudo privileges for a `deploy` user in the `wheel` group without password. You will also need to change the variable `ansible_user` to `deploy` in your `hosts` file.

Using a different user than deploy

Change the variable [deploy_user](https://github.com/consuldemocracy/installer/blob/2.3.1/group_vars/all#L12) to the username you would like to use.

Ansible Documentation

http://docs.ansible.com/

Roadmap

Cross platform compatibility (Ubuntu, CentOS)

Greater diversity of interchangeable roles (nginx/apache, unicorn/puma/passenger, rvm/rbenv)

How to contribute

- [Open an issue](https://help.github.com/en/articles/creating-an-issue)
- [Send us a Pull Request](https://help.github.com/en/articles/creating-a-pull-request)

Support

[![Join the chat at https://gitter.im/consul/consul](https://badges.gitter.im/consul/consul.svg)](https://gitter.im/consul/consul?utm_source=badge&utm_medium=badge&utm_campaign=pr-badge&utm_content=badge)

License

Create a deploy user

The installer by default connects as the `root` user only to create a `deploy` user. This `deploy` user is the one who installs all libraries. If you do not have `root` access, please ask your system administrator to follow these instructions to create a user manually.

You could create a user called `deploy` or any other name. **In this example, we are going to create a user named `jupiter`.**

Copy
```
adduser jupiter
```

Remember to replace `jupiter` with whatever username makes sense for you. Input a password when prompted, and leave the rest of the options empty.

Let's create a `wheel` group and add the user `jupiter` to this group.

Copy
```
sudo groupadd wheel

sudo usermod -a -G wheel jupiter
```

Remember to replace "jupiter" with the username you chose earlier.

Now let's give sudo privileges to the `wheel` group and allow it to not use a password. This is important to ensure the installer doesn't stall waiting for a password.

First we open the sudoers file:

Copy
```
sudo visudo -f /etc/sudoers
```

And we add this line at the end:

Copy
```
%wheel ALL=(ALL) NOPASSWD: ALL
```

Now we need to give the keys of the server to the new user. Don't close the server terminal window, because you can lock yourself out of your server if there is a mistake.

Let's create the necessary directory in the server to upload the public key:

Copy
```
su jupiter
```

```
cd ~
```

```
mkdir .ssh
```

```
cd .ssh
```

```
nano authorized_keys
```

Make sure you have generated a public key in your local terminal.

Open another local terminal window (not in the server) and type:
Copy
```
cat ~/.ssh/id_rsa.pub
```

Copy the content of your public key to the file `authorized_keys` that should still be open in the server.

Test that your user can log in by typing:
Copy
```
ssh jupiter@your-copied-ip-address
```

You should see the server welcome page and a prompt like this:
Copy
```
jupiter@consuldemocracyserver:~$
```

Note the username at the prompt is not "root", but your username. So everything is fine and we can now block the root account from outside access and also stop allowing password access so only people with SSH keys can log in.

Type the following command to edit the SSH config file of the server:
Copy
```
sudo nano /etc/ssh/sshd_config
```

Look for the "PasswordAuthentication yes" line and change it to "PasswordAuthentication no". Type `Control+X` to close the nano editor and type:
Copy
```
sudo service ssh restart
```

Installation
Production and staging servers

Generating SSH Key

These instructions will help you generate a public key with which you can connect to the server without using a password.

In the terminal window, type:
Copy

```
ssh-keygen
```

When prompted for the file in which to save the key just press ENTER to leave the default. When prompted for a passphrase, just press ENTER again to leave this empty. At the end you should see a message like this:

Copy

```
Your identification has been saved in /your_home/.ssh/id_rsa.

Your public key has been saved in /your_home/.ssh/id_rsa.pub.
```

Take note of the **id_rsa.pub** file location, because you'll need the content of this file later.

Mail server configuration

This is an example of how to integrate a mailing service with Consul Democracy.

Get an account from any email provider

To configure email in Consul Democracy, you will need:

The *smtp_address*, which is the address of your email provider's SMTP server (e.g., smtp.yourdomain.com).
The *domain*, which is the domain name of your application.
The *user_name* and *password*, which are the credentials provided by your email provider to authenticate with the SMTP server.

Email configuration in Consul Democracy

Go to the `config/secrets.yml` file.
On this file, change these lines under the section `staging`, `preproduction` or `production`, depending on your setup:

Copy

```
 mailer_delivery_method: :smtp

 smtp_settings:

    :address: "<smtp address>"

    :port: 587

    :domain: "<domain>"
```

```
    :user_name: "<user_name>"

    :password: "<password>"

    :authentication: "plain"

    :enable_starttls_auto: true
```

Fill `<smtp address>`, `<domain>`, `<user_name>` and `<password>` with your information.

Save the file and restart your Consul Democracy application.

Installation

Basic configuration

Once you have Consul Democracy running on the server, there are some basic configuration options that you probably want to define in order to start using it.

To do this, you will need to go to your Consul Democracy installation URL with any browser and log in with the administration user (initially it is the `admin@consul.dev` user with the password `12345678`).

Once you have logged in you will see on the top right of the screen a "Menu" button. Click on it and then click on "Administration" to go to the administration area. From this interface you can configure the following basic options:

Change the administrator credentials

Since the `admin@consul.dev` email address doesn't exist, and since anyone familiar with Consul Democracy will know the default password, we strongly recommended that you change the login credentials immediately after installing the application.

First, sign in using the default administrator credentials.

Then, click on "My account". On the "My account" page, click on "Change my login details". Fill in the form with the new email and password.

The password will be updated immediately, and you can sign out and sign in using the new password. In order to update the email address, you will receive an email on that address asking to confirm your account. Click on the link included in that email, and you'll be able to sign in using the new email address.

You can also change the default "admin" username. Once again, go to "My account". On that page, you will see a form containing a "username" field. Just change the username and click on the form button to update it.

It's possible that, before following this process, you've already signed up with the email you'd like to use as an administrator. If that's the case, instead of following the previous process, sign in as an administrator and go the admin area. Once there, on the side navigation menu, click on "Profiles" to open a submenu and then click on "Administrators". To add an administrator, use the search form and enter the email of the user you'd like to make an administrator (not very intuitive; sorry for that!). Once the search results are returned, click on the "Add" icon. Now, sign out, sign in as the new administrator, go to the admin area, click once again on "Profiles" and then on "Administrators" and delete the default `admin@consul.dev` administrator. Note that this won't delete the user, but that user will no longer be an administrator.

Global configuration parameters

In the admin area, on the side navigation menu, click on "Settings" to open a submenu and then click on "Global settings". Here you will find many interesting parameters, but at the moment we recommend you to define some of the most basic ones (later, when you are more familiar with the tool, you will be able to configure other parameters):

Site name. This name will appear in the subject of emails, help pages, ...
Sender email name. This name will appear as the name of the sender in the emails sent from the application. For example, the email that users receive to confirm that they have created their account.
Sender email address. This email address will appear in the emails sent from the application.
Minimum age needed to participate. If you use a user verification system, this will be the minimum age that users will be required to be.
Number of supports necessary for approval of a Proposal. If you use the citizen proposals section, you can define a minimum number of supports that the proposals need in order to be considered. Any user will be able to create proposals but only those that reach that value will be taken into account.
Level x public official. Consul Democracy allows some user accounts to be marked as "official accounts", and their interventions on the platform will be highlighted. This is used, for example, in a city, if you want to define accounts for the Mayor, Councillors, etc. This public official option will allow you to define the official label that appears next to the usernames of these accounts from most important (level 1) to least important (level 5).

Categories of proposals

When users create proposals on the platform, a few general categories are suggested to help organize the proposals. To define these categories, on the side navigation menu you can click on "Settings" and then click on "Proposals topics". At the top you can find a form where you can write the name of a topic and create it with the button below it.

Definition of geozones

Geozones are smaller territorial areas than the area in which you use Consul Democracy (e.g. districts in a city in which Consul Democracy is used). If the geozones are activated, it will allow for example that citizen proposals are assigned to a specific area, or that polls are restricted to people living in some area.

On the side navigation menu, click on "Settings" and then click on "Geozones". The "Create geozone" button on the right will allow you to create new geozones. Only the name is necessary to define them, but you can add other data which might be useful in certain sections. Initially we recommend that you start by defining only the names of the zones.

Once defined, if you create a citizen proposal you will see how one of the options in the proposal creation form allows you to choose if your proposal refers to a specific geozone.

If you activate the geozones, you can also display an image that represents the area with the zones. You can change this image by clicking on "Site content" (on the side navigation menu) and then clicking on "Custom images". The default image you can change is the one titled "map".

Map to geolocate proposals

You can allow users to place proposals on a map when creating proposals. To do this, you have to define which map you'd like to show.

First, on the side navigation menu click on "Settings" and then click on "Global settings". At the top of this page you'll find a few tabs. Click on the "Features" tab.

On this page, look for a feature titled "Proposals and budget investments geolocation ". If it isn't already enabled, enable it.

Then, at the top of this page, go to the "Map configuration" tab. There you will find three parameters that you will have to fill in:

Latitude. Latitude to show the map position.

Longitude. Longitude to show the map position.

Zoom. Zoom to show the position of the map. You can try an initial value and then change it later.

Then, click on the "Update" button.

If everything has been configured correctly, you will see the map centered on the latitude and longitude you entered before. You can also center the map and change the zoom level directly on the map and then click on the "Update" button.

Emails to users

Consul Democracy sends a series of emails to users. For example, when creating a user account, trying to recover a password, receiving a message from another user, etc.

You can check the content of the templates of the emails that Consul Democracy sends by clicking on "Messages to users" (on the side navigation menu) and then clicking on "System Emails". There you'll be able to preview each email and see a filename indicating where the content of the email is in case you'd like to customize it.

Basic information pages

By default, Consul Democracy includes some basic information pages that will be shown to users, e.g. "Privacy Policy", "Frequently Asked Questions", "Congratulations you have just created your user account", etc.

You can modify these pages and add new ones by clicking on "Site content" (on the side navigation menu) and then clicking on "Custom pages".

Homepage

When users access your Consul Democracy installation, they will see the homepage. This page is configurable, so that you can show the content that seems most relevant to you. You can modify it by clicking on "Site content" (on the side navigation menu) and then clicking on "Homepage".

Try creating "Headers" and "Cards" and activating/deactivating the different functionalities you will find below them to check the effect they have on your homepage.

Platform texts

If, on the side navigation menu, you click on "Site content" and then on "Custom information texts", you will see different tabs with a series of texts. These are all the texts displayed on the platform. By default, you can use the existing ones, but at any time you can access this section to modify any of the texts.

For more information on how to add new translations to your version of Consul Democracy, check the "Texts and translations" section of this documentation.

Types of participation processes

By default, in Consul Democracy users can participate in different ways. In order to get familiar with the tool, we recommend that you activate them all, but you can deactivate the ones that you don't need. To do this, on the side navigation menu click on "Settings" and then click on "Global settings". At the top of this page you will find a few tabs. Click on the "Participation processes" tab.

You will find different functionalities with the names of the different types of participation processes: "Debates", "Proposals", "Voting", "Collaborative legislation" and "Participatory budgets". You can deactivate any of these functionalities and it will no longer be shown in your Consul Democracy installation.

More information and detailed documentation

The options above will allow you to configure your initial version of Consul Democracy. We recommend that you check the User documentation and guides section, where you can find links to more detailed documentation.

Installation

User documentation and guides

There are several guides where you can read detailed information about Consul Democracy and its possibilities. You can find them at: https://docs.consuldemocracy.org.

Consul Democracy Use Guide. In this guide, you can see different ways to use Consul Democracy and examples of participation processes.

Consul Democracy Administration Guide. This guide contains detailed information about the administration and management of Consul Democracy.

Consul Democracy Communication Guide. This guide can give you an initial idea of how to plan communication campaigns to invite people to use your Consul Democracy platform. Communication is a key issue in getting relevant participation and engagement.

In addition to these guides, you can access the Consul Democracy Discussions, a discussion space to share more documentation, questions, learning, etc.

Customization

Introduction

You can modify your own Consul Democracy to adapt it to the visual style of your institution. To do so, first you'll have to create your own fork.

The main principle to follow when customizing Consul Democracy is doing so in a way that custom changes are isolated so, when updating to a new version of Consul Democracy, it'll be easier to check which code belongs to the application and how the code changed in the new version. Otherwise, it'll be very hard to adapt your custom changes to the new version.

For that purpose, Consul Democracy includes custom folders so you rarely have to touch the original application files.

Special Folders and Files

In order to customize your Consul Democracy fork, you'll make use of some `custom` folders on the following paths:

```
app/assets/images/custom/
app/assets/javascripts/custom/
app/assets/stylesheets/custom/
app/components/custom/
app/controllers/custom/
app/form_builders/custom/
app/graphql/custom/
app/lib/custom/
app/mailers/custom/
app/models/custom/
app/models/custom/concerns/
app/views/custom/
config/locales/custom/
spec/components/custom/
spec/controllers/custom/
spec/models/custom/
spec/routing/custom/
spec/system/custom/
```

There are also files where you can apply some customizations:

```
app/assets/javascripts/custom.js
app/assets/stylesheets/custom.css
```

`app/assets/stylesheets/_custom_settings.css`

`app/assets/stylesheets/_consul_custom_overrides.css`

`config/application_custom.rb`

`config/environments/custom/development.rb`

`config/environments/custom/preproduction.rb`

`config/environments/custom/production.rb`

`config/environments/custom/staging.rb`

`config/environments/custom/test.rb`

`config/routes/custom.rb`

`Gemfile_custom`

Using these files, you will be able to customize translations, images, CSS, JavaScript, HTML views, any Ruby code like models, controllers, components or other Ruby classes, gems, application configuration, routes and tests.

Running the tests

When customizing the code, it is **very important** that all the tests in your test suite are still passing. If not, there will be issues when upgrading to a new version of Consul Democracy (or even when updating the custom changes) that won't be detected until the code is running on production. Consul Democracy includes more than 6000 tests checking the way the application behaves; without them, it'd be impossible to make sure that new code doesn't break any existing behavior.

So, first, make sure you've configured your fork so it uses a continuous integration system that runs all the tests whenever you make changes to the code. When changing the code, we recommend opening pull requests (a.k.a. merge requests) using a development branch so the tests run before the custom changes are added to the `master` branch.

Then, if some tests fail, check one of the tests and see whether the test fails because the custom code contains a bug or because the test checks a behavior that no longer applies due to your custom changes (for example, you might modify the code so only verified users can add comments, but there might be a test checking that any user can add comments, which is the default behavior). If the test fails due to a bug in the custom code, fix it ;). If it fails due to a behavior that no longer applies, check the tests customization section.

We also **strongly recommend adding tests for your custom changes**, so you'll have a way to know whether these changes keep working when upgrading to a new version of Consul Democracy.

Coding conventions

Consul Democracy includes linters that define code conventions for Ruby, ERB, JavaScript, SCSS and Markdown. We recommend you follow these conventions in your code as well in order to make it easier to maintain. See also the coding conventions section for more information.

Customization

Translations and texts

Translations

Currently, Consul Democracy is totally or partially translated to multiple languages. You can find the translations at the Crowdin project.

Please join the translators to help us complete existing languages, or contact us through Consul Democracy's discussions to become a proofreader and validate translators' contributions.

If your language isn't already present in the Crowdin project, please open an issue and we'll set it up in a breeze.

The existing translations of the user-facing texts are organized in YAML files under the `config/locales/` folder. Take a look at the official Ruby on Rails internationalization guide to better understand the translations system.

Custom texts

Since Consul Democracy is always evolving with new features, and in order to make it easier to update your fork, we strongly recommend that translation files aren't modified, but instead "overwritten" with custom translation files in case you need to customize a text.

So, if you'd like to change some of the existing texts, you can add your changes to the `config/locales/custom/` folder. You should only include the texts you'd like to change instead of copying the original file. For example, if you'd like to customize the text "CONSUL DEMOCRACY, 2024" (or the current year) that appears on the footer of every page, first locate where it's used (in this case, `app/components/layouts/footer_component.html.erb`) and look at the locale identifier inside the code:

Copy

```
<%= t("layouts.footer.copyright", year: Time.current.year) %>
```

Then find the file where this identifier will be located (in this case, `config/locales/en/general.yml`), and create a file under `config/locales/custom/` (in this case, create a file named `config/locales/custom/en/general.yml`) with the following content:

Copy

```
en:

 layouts:

   footer:

     copyright: Your Organization, %{year}
```

Note that custom locale files should only include your custom texts and not the original texts. This way it'll be easier to upgrade to a new version of Consul Democracy.

Maintaining your custom texts and languages

Consul Democracy uses the i18n-tasks gem, which is an awesome tool to manage translations. Run `i18n-tasks health` for a nice report.

If you have a custom language different than English, you should add it to both `base_locale` and `locales` variables in the i18n-tasks.yml config file, so your language files will be checked as well.

Customization

Images

If you want to overwrite any image, first you need to find out its filename, which will be located under `app/assets/images/`. For example, if you'd like to change the header logo (`app/assets/images/logo_header.png`), create another file with the exact same filename under the `app/assets/images/custom/` folder. Please note that, due to restrictions in the way Ruby on Rails loads images, **you will also have to rename the original file**. In the example, rename `app/assets/images/logo_header.png` to (for example) `app/assets/images/original_logo_header.png` and then create your custom image in `app/assets/images/custom/logo_header.png`.

The images and icons that you will most likely want to change are:

- `apple-touch-icon-200.png`
- `icon_home.png`
- `logo_email.png`
- `logo_header.png`
- `map.jpg`
- `social_media_icon.png`
- `social_media_icon_twitter.png`

Note that, instead of customizing the image using the method explained above, many of these images can be customized in the admin area, under the "Site content custom images" section.

City Map

You'll find the city map at `/app/assets/images/map.jpg`. You can replace it with an image of your city districts, like this map image example.

Afterwards, we recommend you use an online tool like http://imagemap-generator.dariodomi.de/ or https://www.image-map.net/ to generate the html coordinates to be able to generate an image-map for each of the districts. Those coordinates should be introduced on the respective geozones at the admin geozones panel (`/admin/geozones`).

Customization

Styles with CSS

Consul Democracy uses stylesheets written using Sass with the SCSS syntax and placed under the `app/assets/stylesheets/` folder.

In order to make changes to styles, you can add them directly to a file under `app/assets/stylesheets/custom/`. Alternatively, you can use the `app/assets/stylesheets/custom.scss` file.

For example, to change the margin of the footer, create an `app/assets/stylesheets/custom/layout.scss` file with the following content:

Copy

```
.footer {

 margin-top: $line-height;

}
```

This will overwrite the `margin-top` property applied to the `.footer` selector.

Note that CSS precedence rules still apply, so if there's a rule defining the `margin-top` property for the `body .footer` selector, the code above will be ignored. So, to effectively overwrite properties for certain elements, use the exact same selector used in the original code.

Sass variables

Consul Democracy uses variables and functions defined by the Foundation framework and adds a few other variables. In order to overwrite these variables, use the

`app/assets/stylesheets/_consul_custom_overrides.scss` file. For example, to change the background color of the footer, add:

Copy

```
$footer: #fdfdfd;
```

The variables you can override are defined in the `_settings.scss` and `_consul_settings.scss` files.

To define new variables, you can use the `app/assets/stylesheets/_custom_settings.scss` file.

CSS variables

In multi-tenant applications, Sass variables have a limitation: their value will be the same for every tenant.

So, for the most commonly customized colors, Consul Democracy provides CSS variables, which allow you to define different colors for different tenants.

For example, you can customize the brand, buttons, links and main layout colors for just the main tenant with:

Copy

```
.tenant-public {

 --anchor-color: #372;

 --anchor-color-hover: #137;

 --brand: #153;

 --brand-secondary: #134a00;

 --button-background-hover: #fa0;

 --button-background-hover-contrast: #{$black};

 --footer: #e6e6e6;

 --main-header: #351;

 --top-links: var(--main-header);

 --subnavigation: #ffd;

}
```

Check the `app/assets/stylesheets/custom/tenants.scss` file for more information.

Accessibility

To maintain an appropriate accessibility level, if you add new colors, use a Color contrast checker (WCAG AA is mandatory, WCAG AAA is recommended).

Customization

JavaScript

In order to add custom JavaScript code, you can create a file under `app/assets/javascripts/custom/`.

Adding new functions

To create a new function, we need to define it and then call it when the browser loads a page.

For example, in order to create an alert, we can create the file `app/assets/javascripts/custom/alert.js` with the following content:

Copy

```
(function() {

 "use strict";

 App.Alert = {

   initialize: function() {

     alert("foobar");

   }

 };

}).call(this);
```

Then, edit the `initialize_modules` function in `app/assets/javascripts/custom.js`:

Copy

```
var initialize_modules = function() {

 "use strict";



 // Add calls to your custom code here; this will be called when
```

```
 // loading a page



 App.Alert.initialize();

};
```

Overwriting existing functions

In Consul Democracy, functions are defined as object properties. That means that, in order to overwrite a function, you can simply redefine the property of the object containing it. Where possible, it's recommended that your custom code calls the original code and only modifies the cases where it should behave differently. This way, when upgrading to a new version of Consul Democracy that updates the original functions, your custom functions will automatically include the modifications in the original code as well. Sometimes this won't be possible, though, which means you might need to change your custom function when upgrading.

For example, let's change the `generatePassword` function in `app/assets/javascripts/managers.js`. To do so, create a file `app/assets/javascripts/custom/managers.js` with the following content:

Copy

```
(function() {

 "use strict";

 App.Managers.consulGeneratePassword =
App.Managers.generatePassword;



 App.Managers.generatePassword = function() {

   return App.Managers.consulGeneratePassword() + "custom";

 };

}).call(this);
```

This will return what the original function returns followed by the "custom" string.

Customization

Models

In order to create new models or customize existing ones, you can use the `app/models/custom/` folder.

If you're adding a new model that doesn't exist in the original Consul Democracy, simply add it to that folder.

If, on the other hand, you are changing a model that exists in the application, create a file in `app/models/custom/` with same name as the file you're changing. For example, if you're changing the `app/models/budget/investment.rb` file, create a file named `app/models/custom/budget/investment.rb` and require the original one:

Copy

```
load Rails.root.join("app", "models", "budget", "investment.rb")



class Budget

 class Investment

 end

end
```

Since the custom file requires the original one, at this point, the custom model will behave exactly as the original.

Note that, if we do not require the original file, that file won't be loaded and the custom model will do nothing, which will likely result in errors.

Adding new methods

In order to add a new method, simply add the new code to the model. For example, let's add a scope that returns the investments created last month:

Copy

```
load Rails.root.join("app", "models", "budget", "investment.rb")



class Budget

 class Investment

   scope :last_month, -> { where(created_at: 1.month.ago..) }

 end

end
```

With this code, the custom model will have all the methods of the original model (because it loads the original file) plus the `last_month` scope.

Modifying existing methods

When modifying existing methods, it is strongly recommended that, when possible, **your custom code calls the original code** and only modifies the cases where it should behave differently. This way, when upgrading to a new version of Consul Democracy that updates the original methods, your custom methods will automatically include the modifications in the original code as well. Sometimes this won't be possible, though, which means you might need to change your custom method when upgrading.

For example, to change the common abilities model so only verified users can create comments, we'll create a file `app/models/custom/abilities/common.rb` with the following content:

Copy

```
load Rails.root.join("app", "models", "abilities", "common.rb")



module Abilities

 class Common

   alias_method :consul_initialize, :initialize # create a copy of the original method



   def initialize(user)

     consul_initialize(user) # call the original method

     cannot :create, Comment # undo the permission added in the original method



     if user.level_two_or_three_verified?

       can :create, Comment

     end

   end

 end
```

```
end
```

You can find another example in the `app/models/custom/setting.rb` file.

Controllers

Just like models, controllers are written using Ruby code, so their customization is similar, only we'll use the `app/controllers/custom/` folder instead of the `app/models/custom/` folder. Check the models customization section for more information.

Customizing allowed parameters

When customizing Consul Democracy, sometimes you might want to add a new field to a form. Other than customizing the view or the component that renders that form, you need to modify the controller so the new field is accepted. If not, the new field will silently be ignored; this is done to prevent mass assignment attacks.

For example, let's say you've modified the `SiteCustomization::Page` model so it uses a field called `author_nickname` and you've added that field to the form to create a custom page in the admin area. To add the allowed parameter to the controller, create a file `app/controllers/custom/admin/site_customization/pages_controller.rb` with the following content:

Copy

```
load Rails.root.join("app", "controllers", "admin",
"site_customization", "pages_controller.rb")


class Admin::SiteCustomization::PagesController


 private


   alias_method :consul_allowed_params, :allowed_params


   def allowed_params

     consul_allowed_params + [:author_nickname]
```

```
    end

end
```

Note we're aliasing and then calling the original `allowed_params` method, so all the parameters allowed in the original code will also be allowed in our custom method.

Customization

Views and HTML

Just like most Ruby on Rails application, Consul Democracy uses ERB templates to render HTML. These templates are traditionally placed in the `app/views/` folder.

Unlike Ruby code, CSS code or JavaScript code, it isn't possible to overwrite only certain parts of an ERB template. So, in order to customize a view, find the correct file under the `app/views/` folder and copy it to `app/views/custom/`, keeping as well any sub-folder structure, and then apply your customizations. For example, to customize `app/views/welcome/index.html.erb`, copy it to `app/views/custom/welcome/index.html.erb`.

In order to keep track of your custom changes, when using git, we recommend copying the original file to the custom folder in one commit (without any modifications) and then modifying the custom file in a different commit. When upgrading to a new version of Consul Democracy, this will make it easier to check the differences between the view in the old version of Consul Democracy, the view in the new version of Consul Democracy, and your custom changes.

As mentioned earlier, the custom file will completely overwrite the original one. This means that, when upgrading to a new version of Consul Democracy, the changes in the original file will be ignored. You'll have to check the changes in the original file and apply them to your custom file if appropriate.

Note: Consul Democracy only uses the `app/views/` folder for code written before 2021. Code written since then is placed under the `app/components/` folder. The main reason is that components allow extracting some of the logic to a Ruby file, and maintaining custom Ruby code is easier than maintaining custom ERB code.

Customization

Components

For components, customization can be used to change both the logic (included in a `.rb` file) and the view (included in a `.erb` file). If you only want to customize the logic for, let's say, the `Admin::TableActionsComponent`, create a file named `app/components/custom/admin/table_actions_component.rb` with the following content:

Copy

```
load Rails.root.join("app", "components", "admin",
"table_actions_component.rb")



class Admin::TableActionsComponent

 # Your custom logic here

end
```

Check the models customization section for more information about customizing Ruby classes.

If, on the other hand, you also want to customize the view, you need a small modification. Instead of the previous code, use:

Copy

```
class Admin::TableActionsComponent < ApplicationComponent; end



load Rails.root.join("app", "components", "admin",
"table_actions_component.rb")



class Admin::TableActionsComponent

 # Your custom logic here

end
```

This will make the component use the view in `app/components/custom/admin/table_actions_component.html.erb`. You can create this file and customize it to your needs, the same way you can customize views.

Customization

Other Ruby classes (GraphQL, lib, mailers, builders)

Other than models, controllers and components, there are a few other folders containing Ruby code:

- `app/form_builders/`
- `app/graphql/`

`app/lib/`

`app/mailers/`

The files in these folders can be customized like any other Ruby file (see customizing models for more information).

For example, in order to customize the `app/form_builders/consul_form_builder.rb` file, create a file `app/form_builders/custom/consul_form_builder.rb` with the following content:

Copy

```
load Rails.root.join("app", "form_builders",
"consul_form_builder.rb")



class ConsulFormBuilder

 # Your custom logic here

end
```

Or, in order to customize the `app/lib/remote_translations/caller.rb` file, create a file `app/lib/custom/remote_translations/caller.rb` with the following content:

Copy

```
load Rails.root.join("app", "lib", "remote_translations",
"caller.rb")



class RemoteTranslations::Caller

 # Your custom logic here

end
```

Customization

Gems

To add new gems (external tools/libraries written in Ruby) you can edit the `Gemfile_custom` file. For example, to add the rails-footnotes gem you would add:

Copy

```
gem "rails-footnotes", "~> 4.0"
```

And then run `bundle install` and follow any gem specific installation steps from its documentation.

Customization

Application configuration

Overwriting application.rb

If you need to extend or modify the `config/application.rb` file, you can do so using the `config/application_custom.rb` file. For example, if you'd like to change the application timezone to the Canary Islands, just add:

Copy

```
module Consul

 class Application < Rails::Application

   config.time_zone = "Atlantic/Canary"

 end

end
```

In this particular case, note that the application time zone can alternatively be modified by editing the `config/secrets.yml` file.

Remember that in order to apply these changes you'll need to restart the application.

Overwriting environment configuration

If you'd like to customize the development, test, staging, preproduction or production environments, you can do so by editing the files under `config/environments/custom/`.

Note that changes in `config/environments/custom/production.rb` are also applied to the staging and preproduction environments, and changes in `config/environments/custom/staging.rb` are also applied to the preproduction environment.

And remember that in order to apply these changes you'll need to restart the application.

Customization

Routes

When adding custom controller actions, you also need to define a route to configure the URL that will be used for those actions. You can do so by editing the `config/routes/custom.rb` file.

For example, if you'd like to add a new section in the admin area to manage happy thoughts and verify they've become true, you can write:

Copy

```
namespace :admin do

 resources :happy_thoughts do

   member do

     put :verify

   end

 end

end
```

Or, if, for example, you'd like to add a form to edit debates in the admin area:

Copy

```
namespace :admin do

 resources :debates, only: [:edit, :update]

end
```

Doing so, the existing debates routes in the admin area will be kept, and the routes to edit and update them will be added.

Note that the routes you define on this file will take precedence over the default routes. So, if you define a route for `/proposals`, the default action for `/proposals` will not be used and the one you define will be used instead.

Customization

Tests

Tests check whether the application behaves as expected. For this reason, it is **extremely important** that you write tests for all the features you introduce or modify. Without tests, you'll have no reliable way to confirm that the application keeps working as expected whenever you change the code or upgrade to a new version of Consul Democracy. Consul Democracy contains more than 6000 tests checking the way the application behaves; without them, it'd be impossible to make sure that new code doesn't break any existing behavior.

Since running the tests on your development machine might take more than an hour, we recommend configuring your fork so it uses a continuous integration system that runs all the tests whenever you make changes to the code. When changing the code, we recommend

running the tests in a development branch by opening pull requests (a.k.a. merge requests) so the tests run before the custom changes are added to the `master` branch.

Then, if some tests fail, check one of the tests and see whether the test fails because the custom code contains a bug or because the test checks a behavior that no longer applies due to your custom changes (for example, you might modify the code so only verified users can add comments, but there might be a test checking that any user can add comments, which is the default behavior). If the test fails due to a bug in the custom code, fix it ;). If it fails due to a behavior that no longer applies, then you'll have to change the test.

Changing a test is a two-step process:

> Make sure the test checking the default behavior in Consul Democracy isn't run anymore
>
> Write a new test for the new behavior

For the first step, add a `:consul` tag to the test or block of tests. For example, let's check this code:

Copy

```
describe Users::PublicActivityComponent, controller:
UsersController do

 describe "follows tab" do

   context "public interests is checked" do

     it "is displayed for everyone" do

       # (...)

     end



     it "is not displayed when the user isn't following any
followables", :consul do

       # (...)

     end



     it "is the active tab when the follows filters is selected",
:consul do

       # (...)
```

```
    end
  end

  context "public interests is not checked", :consul do
    it "is displayed for its owner" do
      # (...)
    end

    it "is not displayed for anonymous users" do
      # (...)
    end

    it "is not displayed for other users" do
      # (...)
    end

    it "is not displayed for administrators" do
      # (...)
    end
  end
 end
end
```

In the first context, only the first test will be executed. The other two tests will be ignored because they contain the `:consul` tag.

The second context contains the `:consul` tag itself, so none of its tests will be executed.

Remember: whenever you add a `:consul` tag to a test because the application no longer behaves as that test expects, write a new test checking the new behavior. If you don't, the chances of people getting 500 errors (or, even worse, errors that nobody notices) when visiting your page will dramatically increase.

To add a custom test, use the custom folders inside the `spec/` folder:

- `spec/components/custom/`
- `spec/controllers/custom/`
- `spec/models/custom/`
- `spec/routing/custom/`
- `spec/system/custom/`

Technical Features

OAuth

You can configure authentication services with external OAuth providers, right now Twitter, Facebook, Google and Wordpress are supported.

1. Create an App on the platform

For each platform, go to their developers section and follow their guides to create an app.

2. Set the authentication URL of your Consul Democracy installation

They'll ask you for the authentication URL of your Consul Democracy installation, and as you can see running `rails routes |grep omniauth` at your Consul Democracy repo locally:

Copy

```
 user_twitter_omniauth_authorize GET|POST
/users/auth/twitter(.:format) users/omniauth_callbacks#passthru

 user_twitter_omniauth_callback GET|POST
/users/auth/twitter/callback(.:format)
users/omniauth_callbacks#twitter

 user_facebook_omniauth_authorize GET|POST
/users/auth/facebook(.:format) users/omniauth_callbacks#passthru
```

```
 user_facebook_omniauth_callback GET|POST
/users/auth/facebook/callback(.:format)
users/omniauth_callbacks#facebook

 user_google_oauth2_omniauth_authorize GET|POST
/users/auth/google_oauth2(.:format) users
omniauth_callbacks#passthru

 user_google_oauth2_omniauth_callback GET|POST
/users/auth/google_oauth2/callback(.:format)
users/omniauth_callbacks#google_oauth2

 user_wordpress_oauth2_omniauth_authorize GET|POST
/users/auth/wordpress_oauth2(.:format)
users/omniauth_callbacks#passthru

 user_wordpress_oauth2_omniauth_callback GET|POST
/users/auth/wordpress_oauth2/callback(.:format)
users/omniauth_callbacks#wordpress_oauth2
```

So for example the URL for Facebook application would be `yourdomain.com/users/auth/facebook/callback`.

3. Set the key and secret values

When you complete the application registration you'll get a *key* and *secret* values, those need to be stored at your `config/secrets.yml` file:

Copy

```
 twitter_key: ""

 twitter_secret: ""

 facebook_key: ""

 facebook_secret: ""

 google_oauth2_key: ""

 google_oauth2_secret: ""

 wordpress_oauth2_key: ""

 wordpress_oauth2_secret: ""

 wordpress_oauth2_site: ""
```

Technical Features

GraphQL

Characteristics

- Read-only API
- Public access, no authentication needed
- Uses GraphQL technology:
  - Maximum page size (and the default) is 25 records
  - Maximum query depth is set at 8 levels
  - A maximum of two collections can be requested within the same query
  - Support for GET requests (query must be inside the *query string*) and POST requests (query must be within the *body*, encoded as `application/json` or `application/graphql`)

GraphQL

The Consul Democracy API uses GraphQL, specifically the Ruby implementation. If you're not familiar with t
of APIs, we recommended you to check the GraphQL official documentation.

One of the characteristics that differentiates a REST API from a GraphQL one is that with the latter one it's p
for the client to build its own *custom queries*, so the server will only return information in which we're interest

GraphQL queries are written following a format which resembles JSON. For example:

Copy

```
{

 proposal(id: 1) {

   id,

   title,

   public_author {

     id,

     username

   }

 }
```

```
}
```

Responses are formatted in JSON:

Copy

```
{
  "data": {
    "proposal": {
      "id": 1,
      "title": "Increase the amount of green spaces",
      "public_author": {
        "id": 2,
        "username": "abetterworld"
      }
    }
  }
}
```

Making API requests

Following the official recommendations, the Consul Democracy API supports the following kind of requests:

- GET requests, with the query inside the *query string*.
- POST requests
  - With the query inside the *body*, with `Content-Type: application/json`
  - With the query inside the *body*, with `Content-Type: application/graphql`

Supported clients

Since this is an API that works through HTTP, any tool capable of making this kind of requests is capable of the API.

This section presents a few examples about how to make requests using:

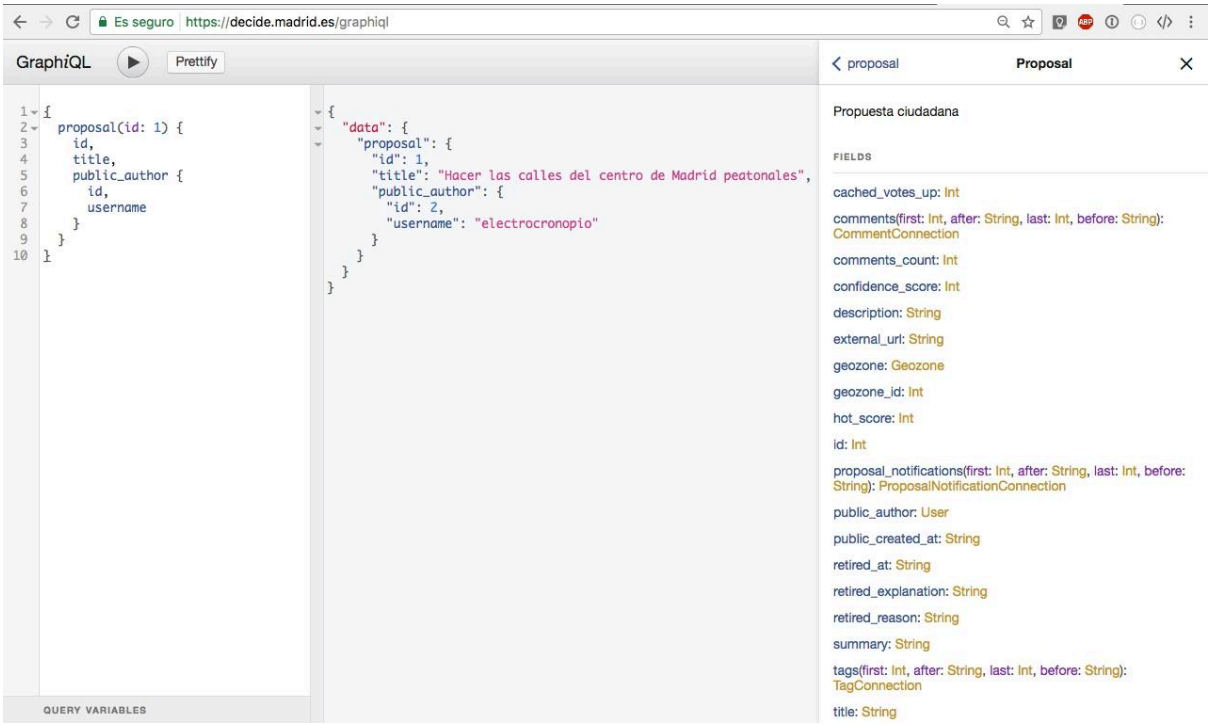
- GraphiQL

Chrome extensions like Postman

Any HTTP library

GraphiQL

GraphiQL is a browser interface for making queries against a GraphQL API. It's also an additional source of documentation. Consul Democracy uses the graphiql-rails to access this interface at `/graphiql`; it's the best get familiar with GraphQL-based APIs.



The interface of GraphiQL

It's got three main panels:

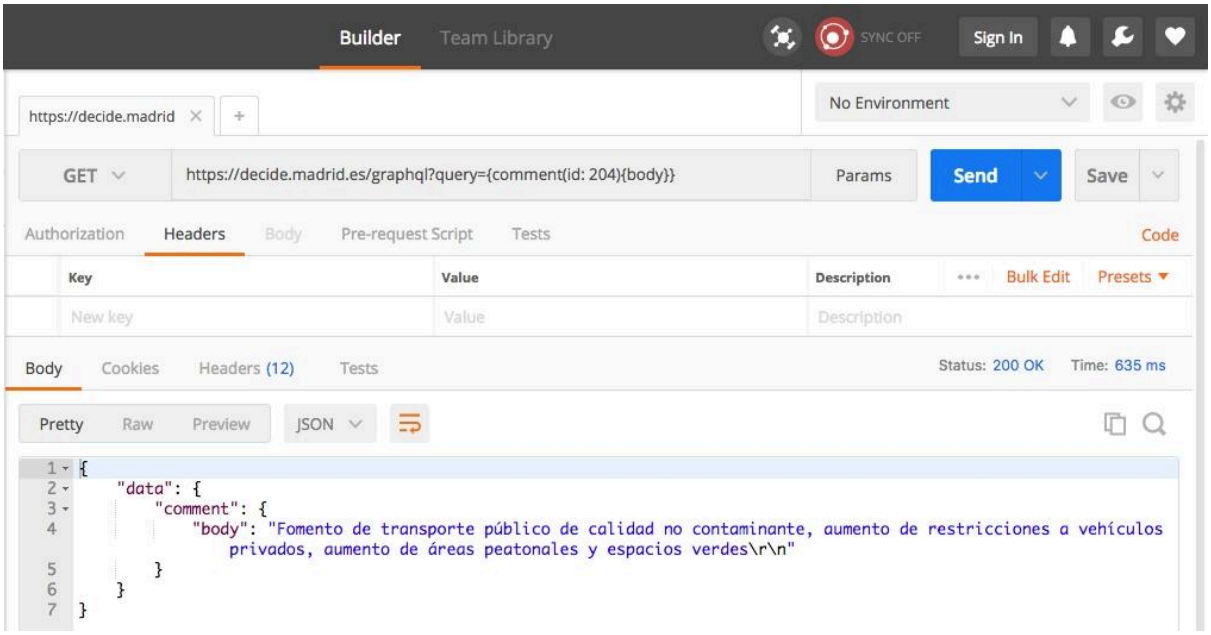
The left panel is used to write the query.

The central panel shows the result of the request.

The right panel (hideable) shows a documentation autogenerated from the models and fields exposed in the API.

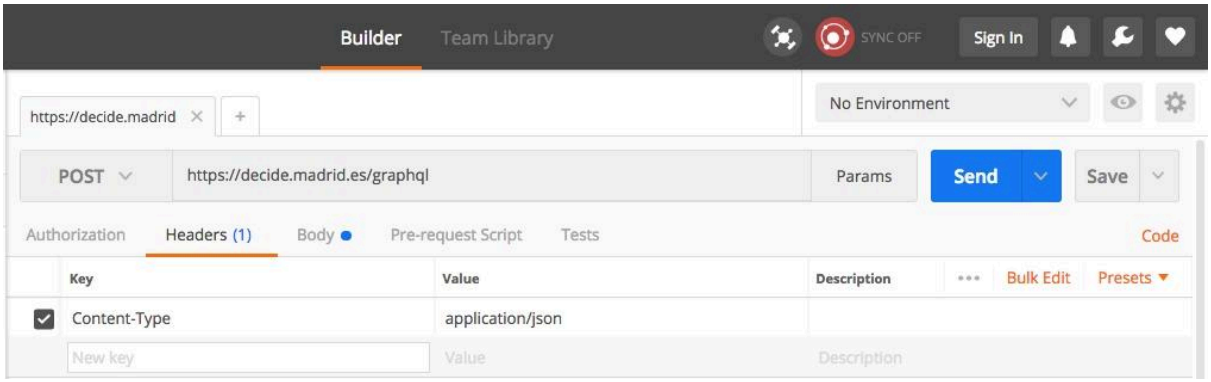
Postman

Here's an example of a `GET` request, with the query as part of the *query string*:



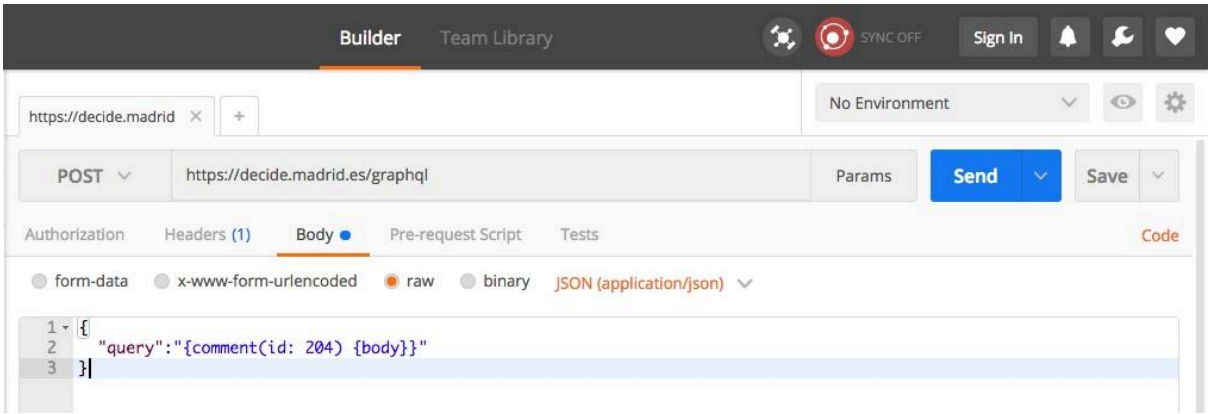
GET request with Postman, showing the query on the browser's URL

And here's an example of a `POST` request, with the query as part of the *body* and encoded as `application/`



"Headers" tabe in Postman, with "Content-Type" set to "application/json"

The query must be located inside a valid JSON document, as the value of the `"query"` key:



"Body" tab in Postman, with the query inside inside the "query" key

HTTP libraries

You can use any of the HTTP libraries available for most programming languages.

IMPORTANT: Some servers might use security protocols that will make it necessary to include a *User Agen* from a web browser so the request is not rejected. For example:

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0
```

Available information

The `app/graphql/types/` folder contains a complete list of all the models (and their attributes) which are cu being exposed in the API.

The models are the following:

Model	Description
`User`	Users
`Debate`	Debates
`Proposal`	Proposals
`Budget`	Participatory budgets
`Budget::Investment`	Budget investments
`Comment`	Comments on debates, proposals and other comments
`Milestone`	Proposals, investments and processes milestones
`Geozone`	Geozones (districts)
`ProposalNotification`	Notifications related to proposals
`Tag`	Tags on debates and proposals
`Vote`	Information related to votes

Examples of queries

Request a single record from a collection

Copy

```
{
 proposal(id: 2) {
   id,
   title,
   comments_count
 }
}
```

Response:

Copy

```
{
 "data": {
   "proposal": {
     "id": 2,
     "title": "Create a dog-friendly area near the beach",
     "comments_count": 10
   }
 }
}
```

Request a complete collection

Copy

```
{
 proposals {
   edges {
     node {
       title
     }
```

```
    }
  }
}
```

Response:
Copy

```
{
  "data": {
    "proposals": {
      "edges": [
        {
          "node": {
            "title": "Bus stop near the school"
          }
        },
        {
          "node": {
            "title": "Improve the lighting in the football stadium"
          }
        }
      ]
    }
  }
}
```

Pagination

The maximum (and default) number of records that each page contains is set to 25. In order to navigate thro
different pages, it's necessary to request `endCursor` information:

Copy

```
{
 proposals(first: 25) {
   pageInfo {
     hasNextPage
     endCursor
   }
   edges {
     node {
       title
     }
   }
 }
}
```

The response:

Copy

```
{
 "data": {
   "proposals": {
     "pageInfo": {
       "hasNextPage": true,
       "endCursor": "NQ=="
     },
     "edges": [
       # ...
```

```
    ]
  }
 }
}
```

To retrieve the next page, pass the cursor received in the previous request as a parameter:

Copy

```
{
 proposals(first: 25, after: "NQ==") {
   pageInfo {
     hasNextPage
     endCursor
   }
   edges {
     node {
       title
     }
   }
 }
}
```

Accessing several resources in a single request

This query requests information about several models in a single request: `Proposal`, `User`, `Geozone` and `Com`

Copy

```
{
 proposal(id: 15262) {
   id,
   title,
```

```
    public_author {
      username
    },
    geozone {
      name
    },
    comments(first: 2) {
      edges {
        node {
          body
        }
      }
    }
  }
}
```

Security limitations

Allowing a client to customize queries is a major risk factor. If queries that are too complex were allowed, it w
possible to perform a DoS attack against the server.

There are three main mechanisms to prevent such abuses:

- Pagination of results
- Limit the maximum depth of the queries
- Limit the amount of information that is possible to request in a query

Example of a query which is too deep

The maximum depth of queries is currently set at 8. Deeper queries (such as the following) will be rejected:

Copy

```
{
```

```
 user(id: 1) {
   public_proposals {
     edges {
       node {
         id,
         title,
         comments {
           edges {
             node {
               body,
               public_author {
                 username
               }
             }
           }
         }
       }
     }
   }
 }
}
```

The response will look something like this:

Copy

```
{
 "errors": [
```

```
    {
      "message": "Query has depth of 9, which exceeds max depth of
8"
    }
  ]
}
```

Example of a query which is too complex

The main risk factor is the option to request multiple collections of resources in the same query. The maximu number of collections that can appear in the same query is limited to 2. The following query requests informa the `users`, `debates` and `proposals` collections, so it will be rejected:

Copy

```
{
 users {
   edges {
     node {
       public_debates {
         edges {
           node {
             title
           }
         }
       },
       public_proposals {
         edges {
           node {
             title
```

```
            }
          }
        }
      }
    }
  }
}
```

The response will look something like this:

Copy

```
{
 "errors": [
    {
      "message": "Query has complexity of 3008, which exceeds max
complexity of 2500"
    }
 ]
}
```

However, it is possible to request information belonging to more than two models in a single query, as long a not try to access the entire collection. For example, the following query that accesses the `User`, `Proposal` an `Geozone` models is valid:

Copy

```
{
 user(id: 468501) {
   id
   public_proposals {
     edges {
       node {
         title
```

```
          geozone {
            name
          }
        }
      }
    }
  }
```

The response:

Copy


```
{
  "data": {
    "user": {
      "id": 468501,
      "public_proposals": {
        "edges": [
          {
            "node": {
              "title": "Make a discount to locals in sports centers",
              "geozone": {
                "name": "South area"
              }
            }
          }
```

```
        ]
      }
    }
  }
}
```

Code examples

Simple example

Here's a simple example of code accessing the Consul Democracy demo API:

Copy

```
require "net/http"

API_ENDPOINT = "https://demo.consuldemocracy.org/graphql".freeze

def make_request(query_string)
  uri = URI(API_ENDPOINT)
  uri.query = URI.encode_www_form(query: query_string)
  request = Net::HTTP::Get.new(uri)
  request[:accept] = "application/json"

  Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |https|
    https.request(request)
  end
end

query = <<-GRAPHQL
```

```
 {
   proposal(id: 1) {
       id,
       title,
       public_created_at
   }
 }
GRAPHQL

response = make_request(query)

puts "Response code: #{response.code}"
puts "Response body: #{response.body}"
```

Example with pagination

And here is a more complex example using pagination, once again accessing the Consul Democracy demo

Copy

```
require "net/http"
require "json"

API_ENDPOINT = "https://demo.consuldemocracy.org/graphql".freeze

def make_request(query_string)
 uri = URI(API_ENDPOINT)
 uri.query = URI.encode_www_form(query: query_string)
 request = Net::HTTP::Get.new(uri)
```

```
 request[:accept] = "application/json"


 Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |https|
   https.request(request)
 end
end


def build_query(options = {})
 page_size = options[:page_size] || 25
 page_size_parameter = "first: #{page_size}"


 page_number = options[:page_number] || 0
 after_parameter = page_number.positive? ? ", after:
\"#{options[:next_cursor]}\"" : ""


 <<-GRAPHQL
 {
   proposals(#{page_size_parameter}#{after_parameter}) {
     pageInfo {
       endCursor,
       hasNextPage
     },
     edges {
       node {
```

```
          id,
          title,
          public_created_at
        }
      }
    }
  }
 GRAPHQL
end

page_number = 0
next_cursor = nil
proposals   = []

loop do
 puts "> Requesting page #{page_number}"

 query = build_query(page_size: 25, page_number: page_number,
next_cursor: next_cursor)
 response = make_request(query)

 response_hash  = JSON.parse(response.body)
 page_info      = response_hash["data"]["proposals"]["pageInfo"]
 has_next_page  = page_info["hasNextPage"]
 next_cursor    = page_info["endCursor"]
```

```
proposal_edges = response_hash["data"]["proposals"]["edges"]

puts "\tHTTP code: #{response.code}"

proposal_edges.each do |edge|
  proposals << edge["node"]
end

page_number += 1

break unless has_next_page
end
```

Technical Features

Debates and proposals recommendations

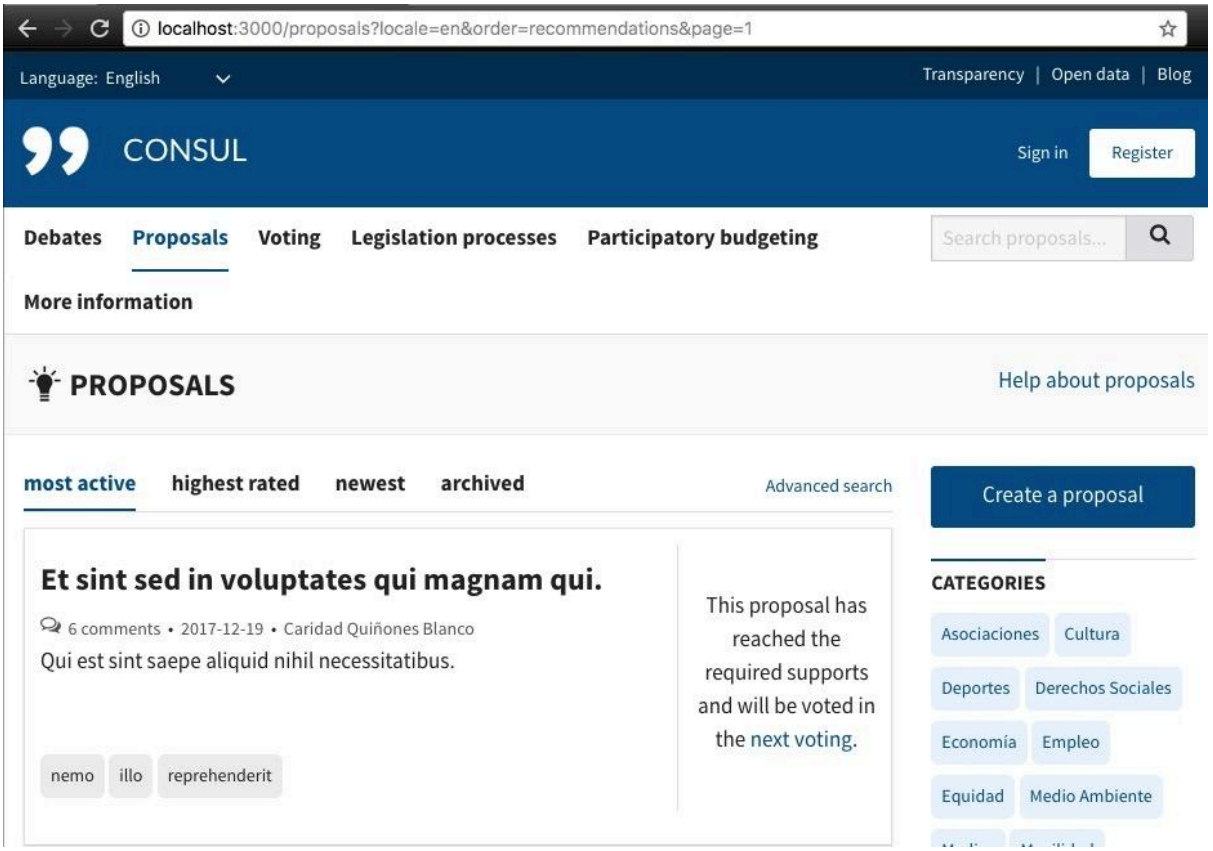
Logged in users can see recommended debates or proposals by sorting them by "recommendations".

The sorted list shows, ordered by votes in descending order, those elements that:

- Have tags that interest the user. Those tags are the ones on the proposals that the user follows.
- The user isn't the author.
- In the case of proposals: only those that haven't reached the required threshold of votes and the user isn't already following.

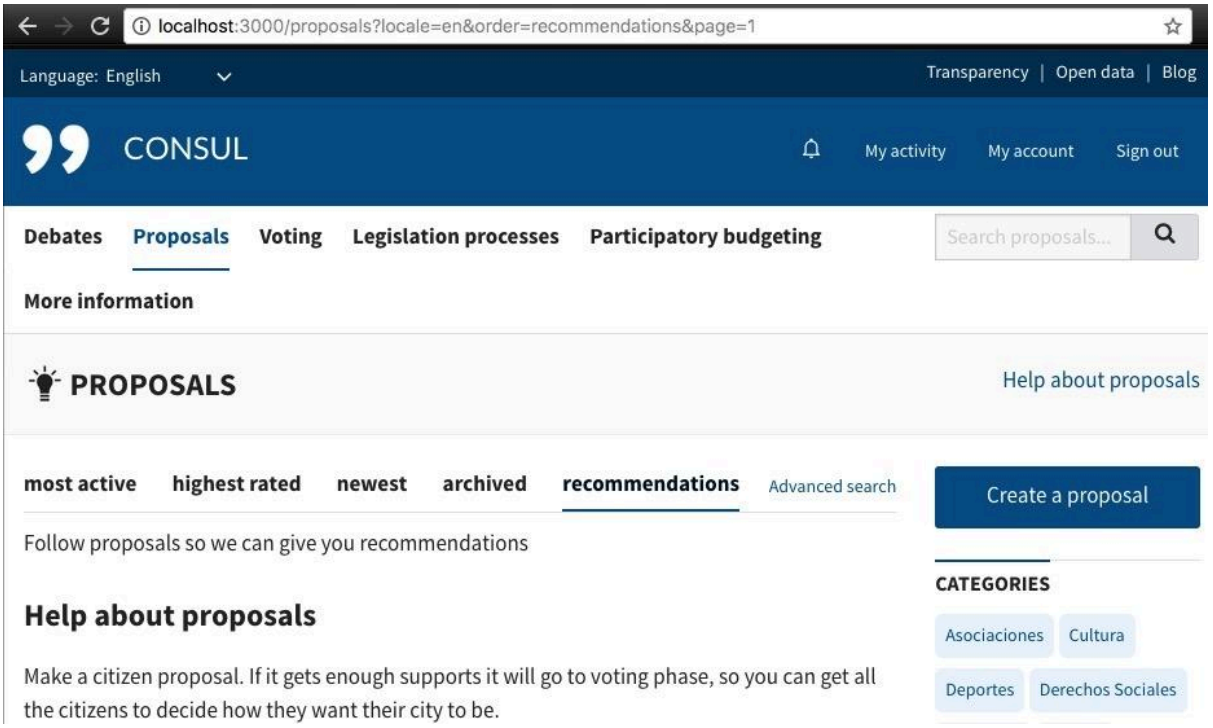
How to try this feature

In our local installation, if we haven't logged in, we can check at http://localhost:3000/proposals that the "recommendations" sorting option isn't present:



The sorting options don't include "recommendations"

Once we log in we see the menu, but since we aren't following any proposals we get the message "Follow proposals so we can give you recommendations" at http://localhost:3000/proposals?locale=en&order=recommendations&page=1

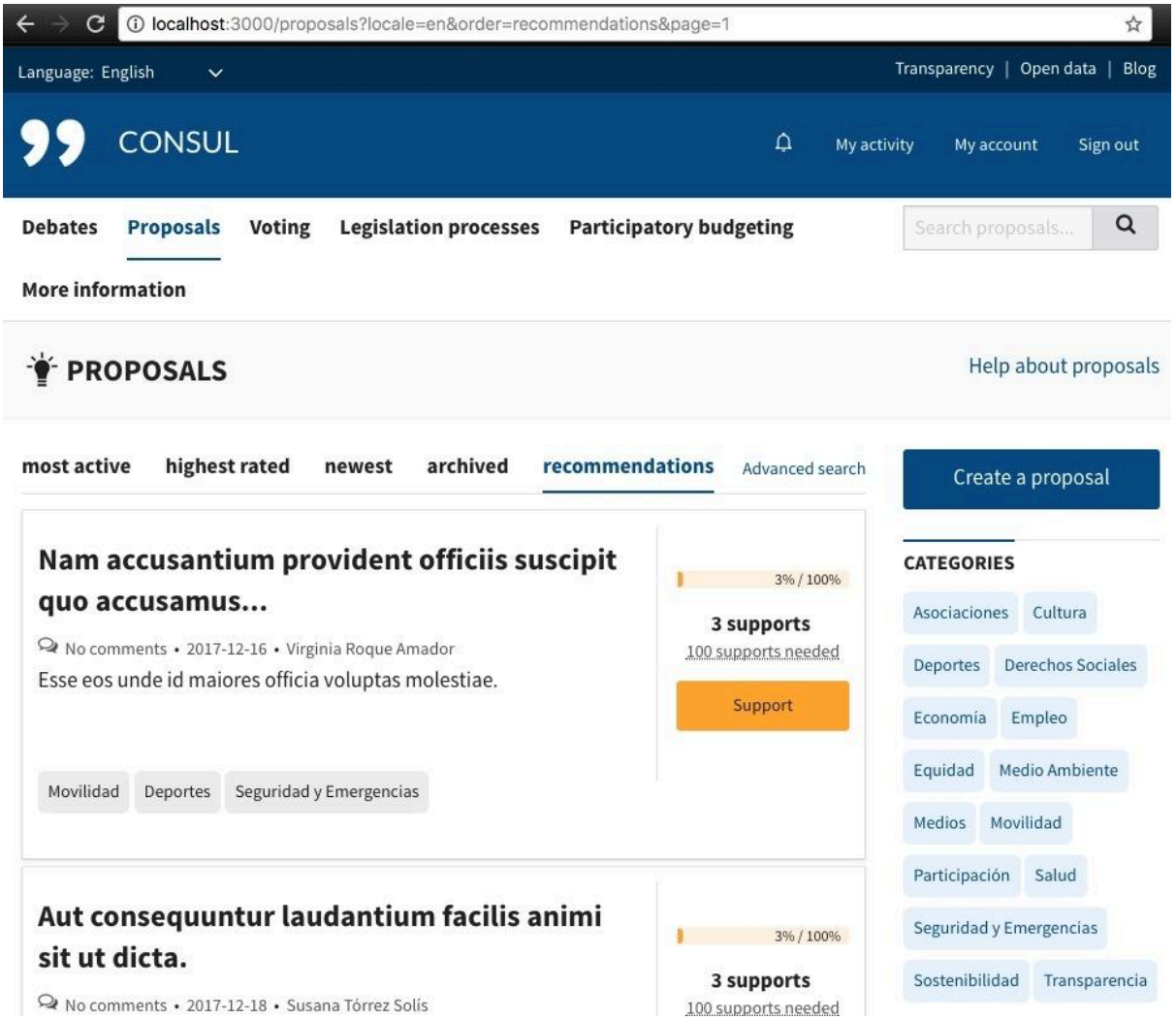


Recommendations are empty

Follow any proposal using the "Follow citizen proposal" button on the side menu:

Button to follow a citizen proposal

Now we can finally see some recommendations:



List of recommendations

The feature works the same way for debates.

Technical Features

Configure Census Connection

The objective of this service is to be able to configure the connection with a census through the administration panel without having to modify the application code.

It should be noted that to properly configure this connection, a technical profile that knows the WebService with which we want to connect will be required.

Enabling the feature

To enable the feature you have to access from the administration panel to the **Settings > Global settings > Features** section and enable the **Configure connection to the remote census (SOAP)** module.

Configuration

Once the feature is activated, we can access the section **Settings > Global settings** and click on the tab **Remote Census Configuration** to be able to configure the connection with the census.

The information to be filled in is divided into three sections:

General Information

Endpoint: Host name where the census service is available (wsdl).

Request data

In this section we will fill in all the necessary fields to be able to make a request to verify a user through a census.

To help you understand how to fill in each of the fields, we will rely on a supposed WebService that receives a method called `:get_habita_datos` with the following structure:

Copy

```
{

 request: {

   codigo_institucion: 12,         # Static Value

   codigo_portal:      5,          # Static Value

   codigo_usuario:     10,         # Static Value

   documento:          12345678Z, # Dynamic value related to
Document Number

   tipo_documento:     1,          # Dynamic value related to
Document Type
```

```
   nivel:                3           # Static Value

 }

}
```

Required fields for the request:

Request method name: Request method name accepted by the census WebService.

Request data - Request method name is filled in with get_habita_datos

Request Structure: Structure of the request received by the WebService of the census. The *static* values of this request should be reported. The *dynamic* values related to Document Type, Document Number, Date of Birth and Postal Code should be filled with *null* value.

Request data - Structure

Copy

```
{


 request: {


   codigo_institucion: 12,          # Since it is a "fixed" value in all requests, we fill in it.
```

```
   codigo_portal:      5,          # Since it is a "fixed" value in
all requests, we fill in it.


   codigo_usuario:     10,         # Since it is a "fixed" value in
all requests, we fill in it.


   documento:          null,       # Since it is a value related to
Document Type, Document Number, Date of Birth or Postal Code, we
fill it in with a null value


   tipo_documento:     null,       # Since it is a value related to
Document Type, Document Number, Date of Birth or Postal Code, we
fill it in with a null value


   codigo_idioma:      102,        # Since it is a "fixed" value in
all requests, we fill in it.


   nivel:              3           # Since it is a "fixed" value in
all requests, we fill in it.


 }
```

```
}
```

Path for document type: Path in the request structure that sends the Document Type.

NOTE: DO NOT FILL IN if the WebService does not require the Document Type to verify a user.

Request data - Path for document type is filled in with request.tipo_documento

Path for document number: Path in the request structure that sends the Document Number.

NOTE: DO NOT FILL IN if the WebService does not require the Document Number to verify a user.

Request data - Path for document number is filled in with request.documento

Path for date of birth: Path in the request structure that sends the Date of Birth.

NOTE: DO NOT FILL IN if the WebService does not require the Date of Birth to verify a user.

In *this example*, we will leave it blank, since it is not necessary to send the date of birth to verify a user.

Request data - Path for date of birth is left blank

Path for Postal Code: Path in the request structure that sends the Postal Code.

NOTE: DO NOT FILL IN if the WebService does not require the Postal Code to verify a user.

In *this example*, we will leave it blank, since it is not necessary to send the postal code to verify a user.

Request data - Path for postal code is left blank

Response data

In this section we will configure all the necessary fields to be able to receive the answer of the WebService and to verify a user in the application.

As in the previous section, we will define an example answer to help you understand how to fill in each of the fields in this section.

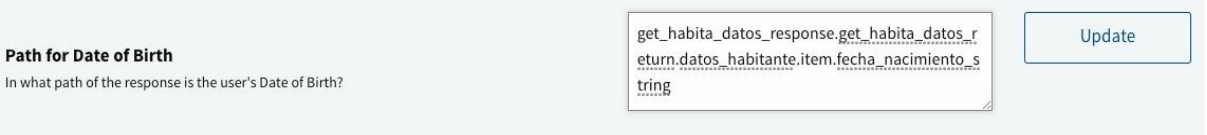
Copy

```
{

 get_habita_datos_response: {

   get_habita_datos_return: {
```

```
      datos_habitante: {
        item: {
          fecha_nacimiento_string: "31-12-1980",
          identificador_documento: "12345678Z",
          descripcion_sexo: "Varón",
          nombre: "José",
          apellido1: "García"
        }
      },
      datos_vivienda: {
        item: {
          codigo_postal: "28013",
          codigo_distrito: "01"
        }
      }
    }
  }
}
```

Required fields to parse the response:

Path for Date of Birth: Path in the response structure containing the user's Date of Birth.



Response data - Path for date of birth is filled in with get_habita_datos_response.get_habita_datos_return.datos_habitante.item.fecha_nacimiento_string

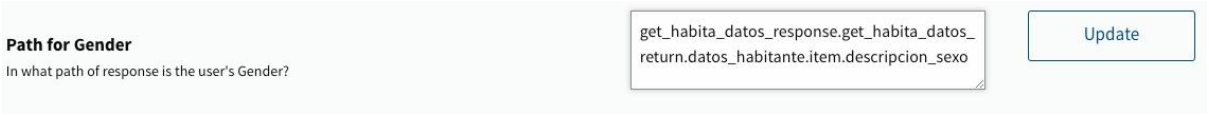
Path for Postal Code: Path in the response structure containing the user's Postal Code.

Response data - Path for postal code is filled in with get_habita_datos_response.get_habita_datos_return.datos_vivienda.item.codigo_postal

Path for District: Path in the response structure containing the user's District.

Response data - Path for district is filled in with get_habita_datos_response.get_habita_datos_return.datos_vivienda.item.codigo_distrito

Path for Gender: Path in the response structure containing the user's Gender.



Response data - Path for gender is filled in with get_habita_datos_response.get_habita_datos_return.datos_habitante.item.descripcion_sexo

Path for Name: Path in the response structure containing the user's Name.

Response data - Path for name is filled in with get_habita_datos_response.get_habita_datos_return.datos_habitante.item.nombre

Path for the Last Name: Path in the response structure containing the user's Last Name.

Response data - Path for last name is filled in with get_habita_datos_response.get_habita_datos_return.datos_habitante.item.apellido1

Path for detecting a valid response: Path in the response structure that must be filled in in valid responses.

Response data - Path for valid response is filled in with get_habita_datos_response.get_habita_datos_return.datos_habitante.item

Once the general data, the necessary fields of the request and **all** fields to validate the response have been correctly filled in, the application will be able to verify any user through the defined WebService.

Local Census

This feature is designed for installations with limited resources that cannot configure and customize their census connection remotely, so we offer them a solution to use a local census instead. For this purpose, administrators are provided with a way to manage the local census database through the administration panel under **Settings > Local census**.

Managing the local census

Administrators can manage this census in two different ways:

Manually: adding one citizen at a time through a form.

Automatically: through an importation process.

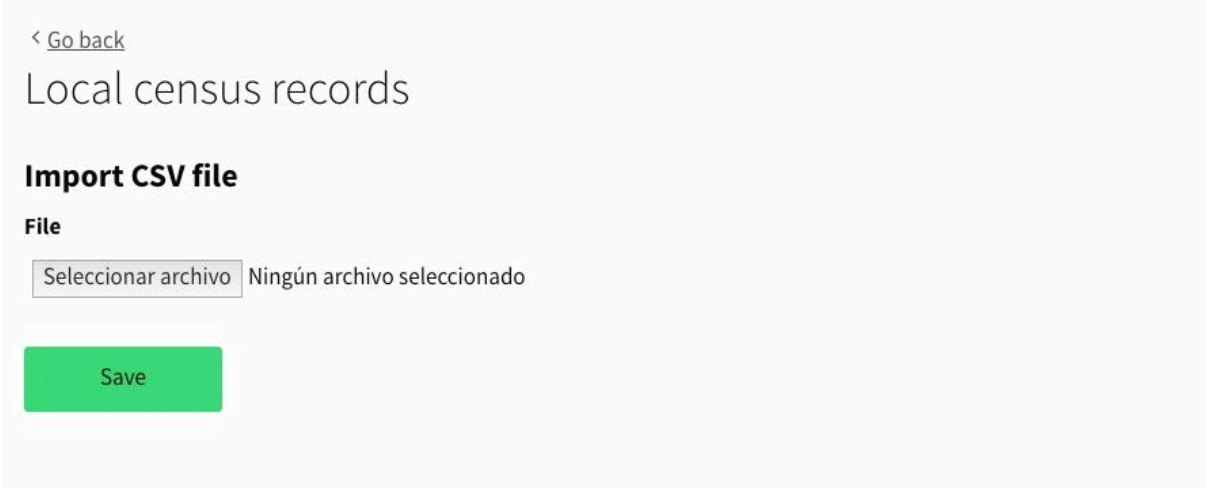
Manually

Administrators can create a record by clicking the *Create new local census record* button, which appears in the top right corner of the page. This will take us to a new page where we can fill in the following form and create the new record:

Form to create a record by filling in the document type, document number, date of birth and postal code

Automatically

Administrators can import multiple records through a CSV file by clicking the *Import CSV* button, which appears in the top right corner of the page. This will take us to a new page where we can attach a CSV file to create the new records:



Page to import new records via csv

Features

Search by document number: Since the local census could contain a lot of records, we have added a search feature to allow administrators to find existing records by document number.
Avoid the introduction of duplicated records: A model validation has been added that does not allow adding records that share the same *number* and *type* of document.

Technical Features

Multitenancy

What's multitenancy and how does it work

The multitenancy feature allows managing several independent institutions ("tenants") using the same application. For example, in our case, a user who signs up for a certain tenant will only be able to sign in on that tenant, and that user's data won't be available from any other tenant.

Which tenant we're accessing depends on the URL we're using in the browser to access the application. In Consul Democracy, the current tenant is established by the subdomain used in this URL. For example, if we used the domain `solarsystemexample.org` to manage the planets in the Solar System, using the URL `https://mercury.solarsystemexample.org` we'd access data from the planet Mercury, while using the URL `https://venus.solarsystemexample.org` we'd access data from the planet Venus. It's also possible to use different domains per tenant (for example, `earthexample.org`).

Enabling multitenancy

Preliminary steps after upgrading from Consul Democracy version 1.5.0

If you're upgrading a Consul Democracy installation to version 2.0.0 from version 1.5.0, you'll have to follow these steps before enabling multitenancy. These steps aren't necessary on Consul Democracy installations that used version 2.0.0 or later as their initial version.

First, after deploying version 2.0.0 to your production server, execute the release tasks:
Copy
```
RAILS_ENV=production bin/rails consul:execute_release_tasks
```

After running this command, you might get the following warning:
The database search path has been updated. Restart the application to apply the changes.

If that's the case, restart the application. If not, make sure the `config/database.yml` file contains the line `schema_search_path: "public,shared_extensions"` and, if that's not the

case, add it for example below the line saying `adapter: postgresql` and restart the application.

Next, open a database console with a user having permission to create and manage database extensions:

Copy

```
sudo -u postgres psql -d consul_production
```

If you didn't use the installer to install Consul Democracy, you might need to execute a couple of queries to make sure the Rails database user has permission to create schemas and the shared extensions schema has the right permissions:

Copy

```
CREATE SCHEMA shared_extensions AUTHORIZATION
<replace_with_rails_database_username>;

GRANT CREATE ON DATABASE consul_production TO
<replace_with_rails_database_username>;

GRANT usage ON SCHEMA shared_extensions TO public;
```

Whether or not you installed Consul Democracy with the installer, run:

Copy

```
ALTER EXTENSION pg_trgm SET SCHEMA shared_extensions;

ALTER EXTENSION unaccent SET SCHEMA shared_extensions;
```

Common step for all Consul Democracy installations

There are two possible ways to enable multitenancy:

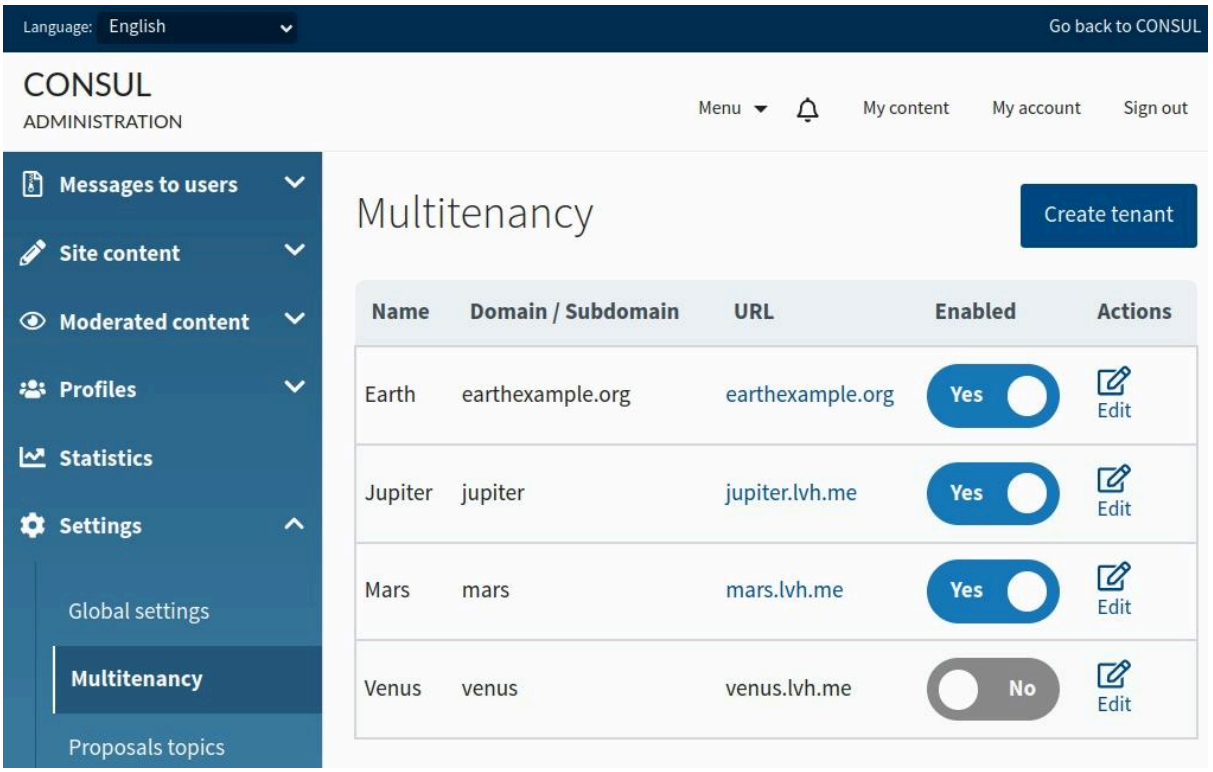
- Adding `config.multitenancy = true` inside the `class Application < Rails::Application` class in the `config/application_custom.rb` file
- Replacing the line `multitenancy: false` with `multitenancy: true` (or adding it if it isn't already there) in the `config/secrets.yml` file

The difference between these options is that the first one uses a file under version control while the second one uses a file which isn't under version control. Choose the first option if you'd like to have this change in your git repository; otherwise, use the second one.

After enabling this option, restart the application.

Managing tenants

Once multitenancy has been enabled and the application has been restarted, you'll see a new "Multitenancy" section inside the "Settings" menu in the Consul Democracy admin panel.

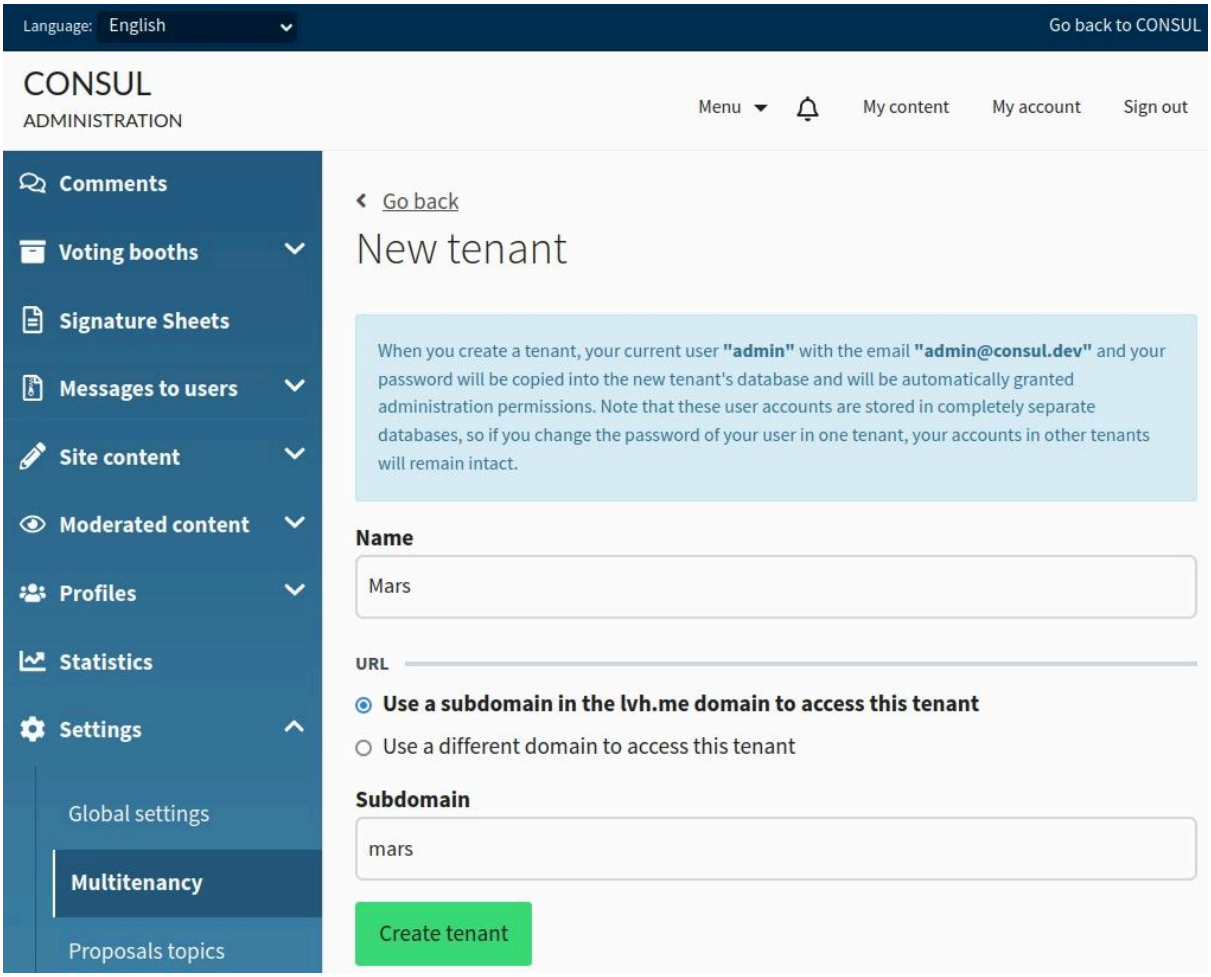


New section with the list of tenants, with their name and domain or subdomain

This section will only be available from the "main" tenant (the one which is created by default). It will not be possible to add/edit tenants by accessing the admin section of any other tenant.

Since removing a tenant would delete **all** its associated data, making it impossible to restore it, Consul Democracy doesn't allow deleting a tenant using the admin panel. However, it's possible to disable a tenant so it cannot be accessed.

The interface to manage tenants is very simple, needing just a name and a domain or subdomain.



Form to edit a tenant, with name and domain or subdomain fields; radio buttons are used to choose domain or subdomain

The name will be used to set the default site name for new tenants. Note that, once a tenant is created, changing this name will have no effect. To change the site name of an existing tenant, edit it in the "Global settings" section in the administration of that tenant.

The domain or subdomain will be used to access this tenant. If you've got a domain like `solarsystemexample.org` and would like to access tenants using subdomains (like `mars.solarsystemexample.org`), choose "Use a subdomain". If you're using a different domain for the tenant (like `marsexample.org`), choose "Use a different domain".

Note that, if you use a different domain for a tenant, you'll have to configure your SSL certificates, web server and DNS so they support that domain and point to your Consul Democracy application.

When adding a new tenant, an admin user **copying the same login data as the administrator creating the tenant** will be automatically created. Note this user is stored in the database schema of the new tenant, so changing their password in one tenant won't change their password in any other tenants.

Multitenancy management mode

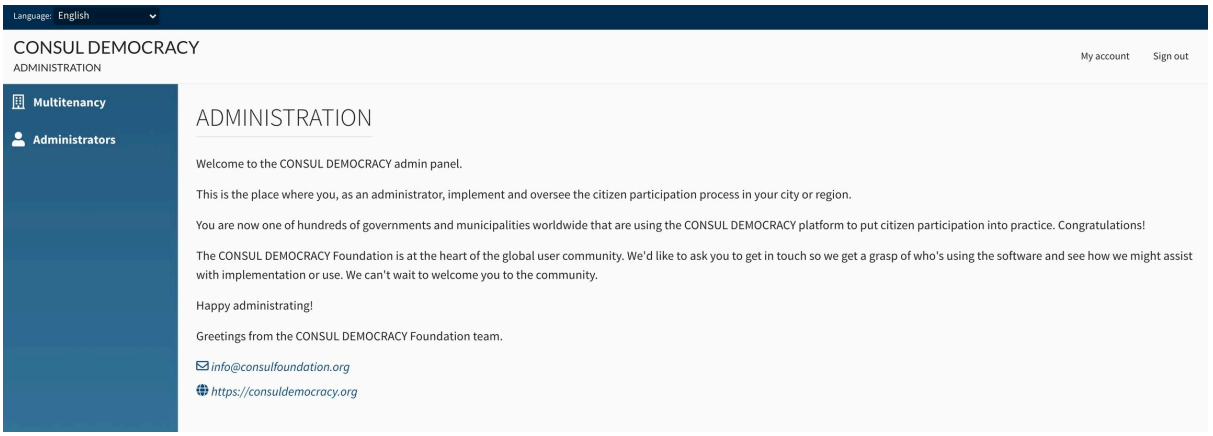
The `multitenancy_management_mode` setting allows using the main tenant solely for managing other tenants and admin users, hiding any other admin panel functionality or public content.

There are two possible ways to enable multitenancy management mode:

> Adding `config.multitenancy_management_mode = true` inside the `class Application < Rails::Application` class in the `config/application_custom.rb` file
>
> Replacing the line `multitenancy_management_mode: false` with `multitenancy_management_mode: true` (or adding it if it isn't already there) in the `config/secrets.yml` file

We recommend using the same method that has been used to enable the multitenancy functionality in the Common step for all Consul Democracy installations section.

After enabling this option, restart the application and you will see the administration panel as follows:



The administration panel only contains links to multitenancy and administrators

Steps to take after adding a tenant

SSL certificates

In order to make it possible to access the application using secure HTTPS/SSL connections, you'll need a valid SSL certificate for the tenant you've just added. Since every institution using Consul Democracy has a different system to manage these certificates, getting a valid SSL certificate for the new tenant will need a different process depending on the way your institution manages these certificates.

If you've installed Consul Democracy using the installer and are using Certbot to manage these certificates, you have two options.

One option would be adding each certificate manually every time you create a tenant. For example, in order to add a tenant using the `mars` subdomain in the `solarsystemexample.org` domain, run:

Copy

```
sudo certbot certonly --nginx --noninteractive --agree-tos
--expand -d solarsystemexample.org,mars.solarsystemexample.org
```

If you're going to add many subdomains at different times, this task can be tedious. Another option is enabling any subdomain. In order to achieve this goal, you need access to your DNS configuration in order to follow the instructions you'll get by either using one of the Certbot DNS plugins or the manual generation of the certificate with the following command:

Copy

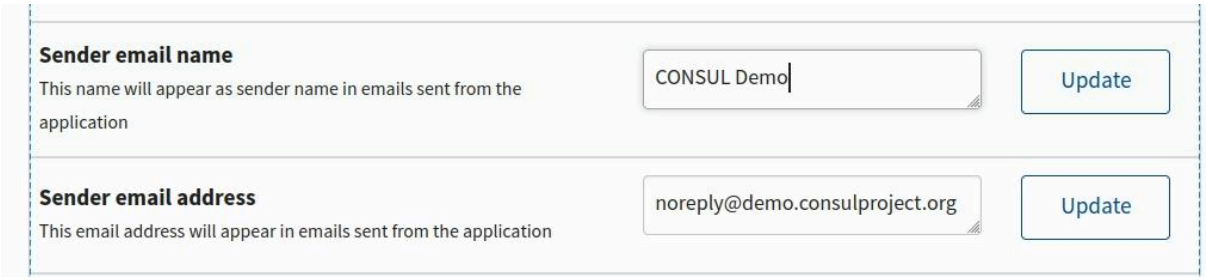
```
sudo certbot certonly --manual --agree-tos --expand -d
solarsystemexample.org,*.solarsystemexample.org
```

You'll be asked to create a DNS TXT record with the subdomain `_acme-challenge` on your domain, with a certain value. You might also be asked to create a file with a certain name containing a certain content (usually in a `.well-known/acme-challenge` folder); if that's the case, assuming you're using Consul Democracy's default folders, create it in `/home/deploy/consul/current/public/.well-known/acme-challenge/`.

After doing so, update your web server configuration file (by default `/etc/nginx/sites-enabled/default`) so it uses the generated certificate, and restart the web server with `sudo systemctl restart nginx`.

Sending e-mails

In order to reduce the chance your application sends emails which are erroneously identified as spam, you might want to edit the fields "Sender email name" and "Sender email address" in the administration panel of the new tenant. The default values for these fields are the name and subdomain introduced when creating the tenant.



Fields to edit sender email name and address

If you'd like to use a different mail configuration for the new tenant, like one for a hypothetical `jupiter` subdomain, edit the `config/secrets.yml` file this way:

Copy

```
production:

  # (...) Other secrets
```

```
 multitenancy: true

 tenants:

   jupiter:

     mailer_delivery_method: :smtp

     smtp_settings:

       :address: <mail_server>

       :port: <port>

       :domain: <your_domain>

       :user_name: <username>

       :password: <password>

       :authentication: "plain"

       :enable_starttls_auto: true

  # (...) Other secrets
```

After editing this file, restart the application.

Sign in via social networks

If you've configured applications so users can sign in via Twitter, Google, Facebook or Wordpress, you need to allow the new tenant to access these applications. You have two options.

The first option is changing your existing application using the Twitter/Google/Facebook/Wordpress dashboard and add the new tenant's URL to the list of allowed domains. When doing so, take into account your application's terms and conditions settings, which might not be compatible with this option.

The other option is creating a new Twitter/Google/Facebook/Wordpress application and configuring it so it can be used from the new tenant. In this case, you'll have to add the configuration of this application to the `config/secrets.yml` file. For example, if you've added a tenant using the `saturn` subdomain, edit the file this way, filling in the keys and secrets of the services you're using:

Copy

```
production:

 # (...) Other secrets
```

```
multitenancy: true

tenants:

  saturn:

    twitter_key: <twitter_key>

    twitter_secret: <twitter_secret>

    facebook_key: <facebook_key>

    facebook_secret: <facebook_secret>

    google_oauth2_key: <google_key>

    google_oauth2_secret: <google_secret>

    wordpress_oauth2_key: <wordpress_key>

    wordpress_oauth2_secret: <wordpress_secret>

    wordpress_oauth2_site: <wordpress_site>

  # (...) Other secrets
```

After editing this file, restart the application.

Enabling different languages for different tenants

In Consul Democracy 2.2.0 or later, it's possible to display the application in different languages for different tenants.

By default, every tenant uses all the languages defined in `config/application.rb`. You can (optionally) overwrite this value by customizing your application configuration. Note that **if you overwrite this value, tenants will only be able to enable the languages you define here**. So, for instance, with this code in the `config/application_custom.rb` file:

Copy

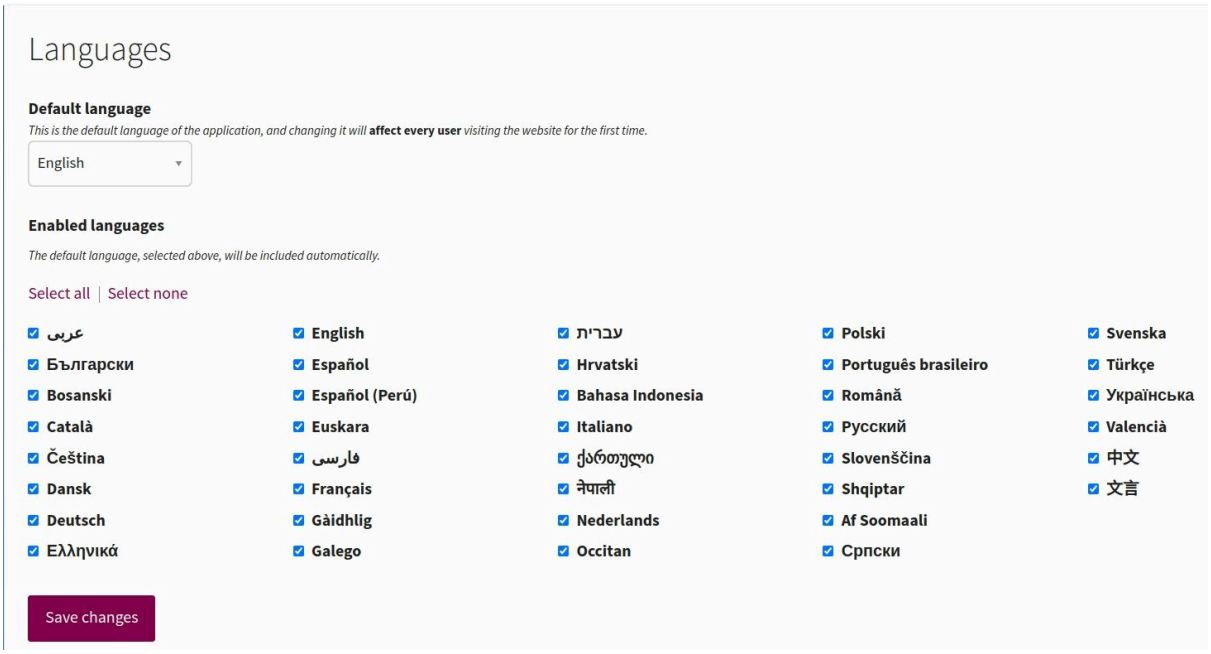
```
module Consul

  class Application < Rails::Application

    config.i18n.available_locales = ["de", "en", "es", "fr", "it",
"ru", "zh-CN"]

  end
```

```
end
```

After restarting the application, tenants will be able to choose which language is the default and which ones are enabled among German, English, Spanish, French, Italian, Russian and Simplified Chinese. However, they'll no longer be able to enable any other language.

To define the default and the enabled languages for a tenant, go the administration area of that tenant. Then, on the side navigation menu, click on "Settings" and then click on "Languages". Note that **this section is not present if you only have one language in `available_locales`**.

On this page you'll find a form to choose the default and the enabled languages for this tenant (note: this form changes slightly when only a few languages are available):



Form with a select control for the default language and a list of checkboxes to choose which ones to enable

Choose the ones you'd like, save the changes, and the language selector at the top of the web will be updated immediately.

Information to take into account during development

Maintenance of the schema.rb file

When Consul Democracy creates a tenant, it loads the content of the `db/schema.rb` file to create a new database schema for the new tenant. This means that if for some reason this file doesn't contain the same database structure you'd get by creating a new database and running the migrations with `rake db:migrate`, you could end up with different database tables or columns on different tenants. This could result in a disastrous situation.

In order to avoid it, we recommend checking the integrity of the `db/schema.rb` file in your continuous integration system. If you're doing continuous integration using GitHub Actions, you can use the workflow already included in Consul Democracy. Pull requests adding this check on GitLab CI or other continuous integration environments are welcome.

Using custom styles for a tenant with CSS

When the multitenancy feature is enabled, Consul Democracy adds a class to the `<html>` element, making it possible to apply styles (or JavaScript events) to just a specific tenant. For example, the tenant with the `uranus` subdomain would have the `tenant-uranus` class.

This way, it'll be possible to overwrite the default styles for just this tenant by creating a new stylesheet in the `app/assets/stylesheets/custom/` folder:

Copy

```
.tenant-uranus {

 // Styles that will only be applied to the Uranus tenant

}
```

To easily change the default colors on a specific tenant, you can use CSS variables; their usage is documented in the app/assets/stylesheets/custom/tenants.scss file. For example, to make the brand colors green on the tenant with the `uranus` subdomain, write:

Copy

```
.tenant-uranus {

 --brand: #350;

 --brand-secondary: #173a00;

}
```

Using different custom ERB views for a tenant

Sometimes it might be convenient to use completely different views for different tenants. For example, a certain tenant might use a footer that has nothing to do with the default one.

In these cases, instead of adding conditions like `case Tenant.current_schema` to the view, using a different file might result in code easier to maintain.

For this purpose, we can use Rails variants, which means that, for example, a tenant named `milky-way` will use a view file ending with `.html+milky-way.erb` if it's available. That is, in order to use a different `application.html.erb` layout for the `milky-way` tenant, add a new file under `app/views/custom/layouts/application.html+milky-way.erb`.

We recommend only using this feature when there are substantial differences between the default view and the specific view for a tenant. If the differences are small, use `if` or `case` conditions instead in order to avoid code duplication.

The same principle works for components too, but in this case, when using the `custom/` folder to add ERB files for a tenant, the default tenant ERB file needs to be added to the `custom/` folder as well; if there aren't changes to this file, a symbolic link will do.

For example, if you're writing a custom `admin/action_component` component view for the `milky-way` tenant but don't need to change this file for the default tenant:

> Create the `app/components/custom/admin/action_component.rb` file according to the components customization documentation
> Create the custom view for the `milky-way` tenant and save it under `app/components/custom/admin/action_component.html+milky-way.erb`
> Enter the `app/components/custom/admin/` folder and run `ln -s ../../admin/action_component.html.erb`

Current limitations of multitenancy

The multitenancy feature was first included in Consul Democracy 2.0.0 and there are a few things that are still missing.

Applications which can be accessed from multiple domains

You might have a Consul Democracy application which can be accessed from two different domains; for example, `solarsystemexample.org` and a domain in Spanish named `ejemplodesistemasolar.org`.

In this case, the source code needs to be changed a little so multitenancy works with both domains. In particular, the `allowed_domains` method in the `Tenant` class needs to be changed in order to include both domains. See the models customization documentation for examples on how to customize methods like this one.

Custom images per tenant

The administration panel in Consul Democracy contains a "Custom images" section, where you can customize some (but not all) images appearing in the application. Using this interface allows having different images per tenant.

Sometimes it's useful to have a certain image under version control, though. For instance, if we'd like to use a different logo for a tenant with the `neptune` subdomain, we'd put that file under `app/assets/images/custom/tenants/neptune/logo_header.png`.

However, this will only work for images which can already be configured through the administration interface. If you'd like to customize a different image, you'll have to change the code rendering it. For instance, to make it possible to customize the `avatar_admin.png` image, replace the code `image_tag("avatar_admin.png")` with `image_tag(image_path_for("avatar_admin.png"))`.

Databases on different servers for different tenants

In Consul Democracy 2.0.0, data from all tenants is stored in the same database and so it isn't possible to use several databases on different servers.

If this feature is requested often, it'll be possible to include it in Consul Democracy in the future, since Rails 6.1 and Rails 7.0 added better support for it.

Deleting tenants

Since removing a tenant would delete **all** its associated data, making it impossible to restore it, Consul Democracy doesn't allow deleting tenants using the admin panel and only allows disabling them so they cannot be accessed. To completely delete a tenant, use the Rails console.

Technical Features

User content translations

Remote translations on demand by the user

The aim of this service is to be able to offer all the dynamic contents of the application (proposals, debates, budget investments and comments) in different languages automatically.

When a user visits a page with a language where there is untranslated content, they will have a button to request the translation of all the content. This content will be sent to an automatic translator (in this case Microsoft TranslatorText) and as soon as the response is obtained, all these translations will be available to any user.

Getting started

In order to use this functionality, the following steps are necessary:

Create an Azure account.

Once you are logged into the Azure portal, create a resource of type *Translator Text*

(note: for this implementation it is necessary to select the GLOBAL region when

creating the resource, otherwise it will be necessary to customize the API calls by adding the selected region).

Once you have subscribed to the Translator Text service, you will have access to two API keys in the section **Resource Management > Keys and Endpoint** that will be necessary for the configuration of the translation service in your application.

Configuration

To enable the translation service in your application you must complete the following steps:

Add api key in the application

In the previous section we have mentioned that, once subscribed to the translation service, we get two API keys. To configure the service correctly in our application we must add one of the two API keys to the `apis:` section of the `secrets.yml` file, with the key `microsoft_api_key` as follows:

Copy

```
apis: &apis

 census_api_end_point: ""

 census_api_institution_code: ""

 census_api_portal_code: ""

 census_api_user_code: ""

 sms_end_point: ""

 sms_username: ""

 sms_password: ""

 microsoft_api_key: "new_api_key_1_for_translator_text"
```

Enabling the feature

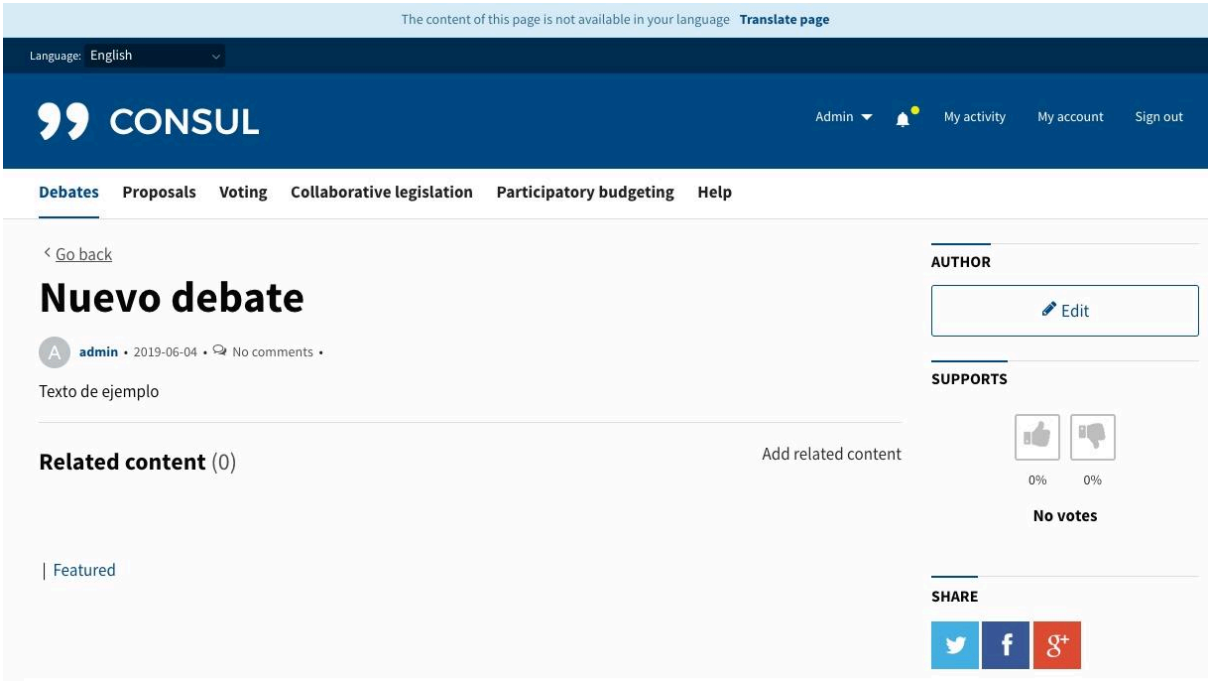
Once we have the new key in the `secrets.yml` we can now proceed to enable the feature. To enable it, in the administration area access the section **Settings > Global settings > Features** and enable the **Remote translation** feature.

Use Cases

Once we have the api key in our `secrets.yml` and the feature enabled, users will now be able to use remote translations in the application.

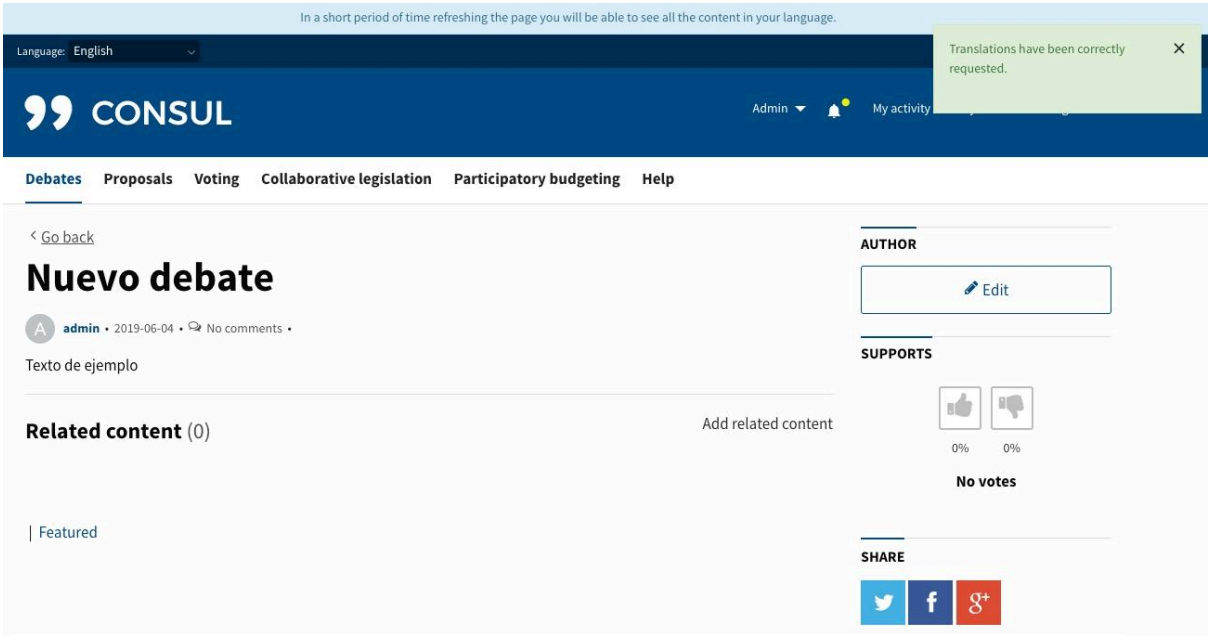
We attach some screenshots of how the application interacts with our users:

When a user visits a page in a language without translated content, an informative text will appear at the top of the page next to a button to request the translation. (**Note:** *If a user visits a page with a language not supported by the translation service, no text or translation button will be displayed. See section: Available languages for remote translation*)



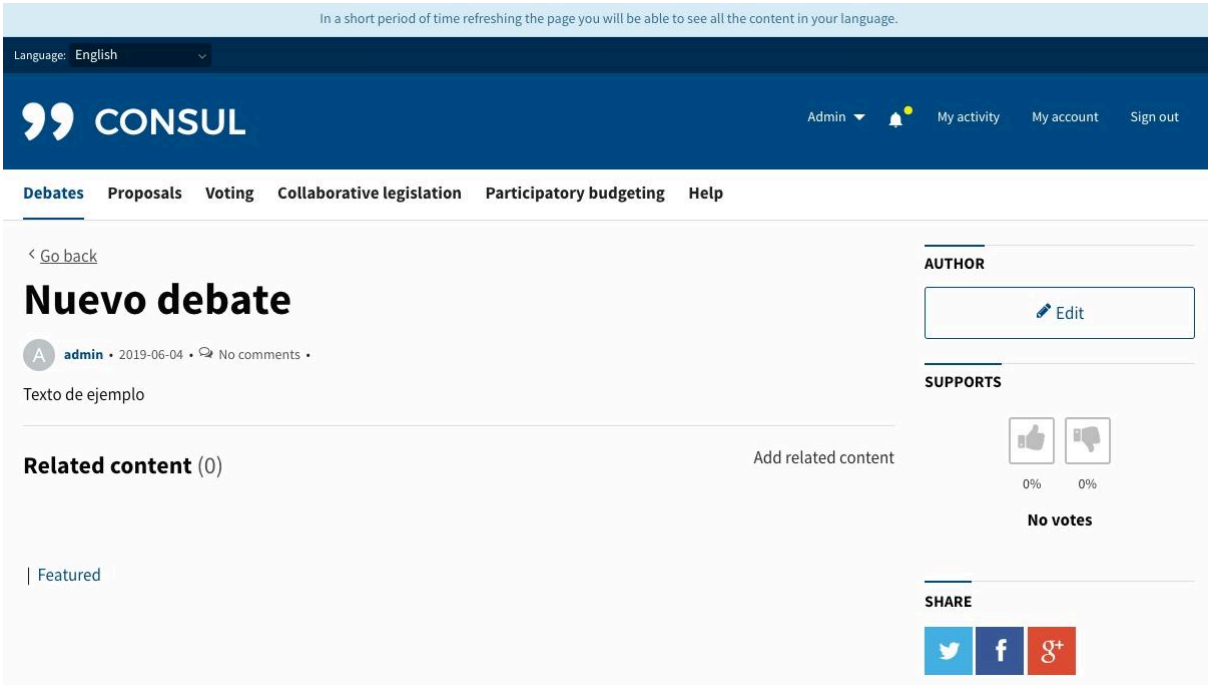
The text "The content of this page is not available in your language" is displayed next to a "Translate page" button at the top of the page

Once the user clicks the `Translate page` button, the translations are enqueued and the page is reloaded with a notice (*informing that the translations have been requested correctly*) and an informative text in the header (*explaining when you will be able to see these translations*).



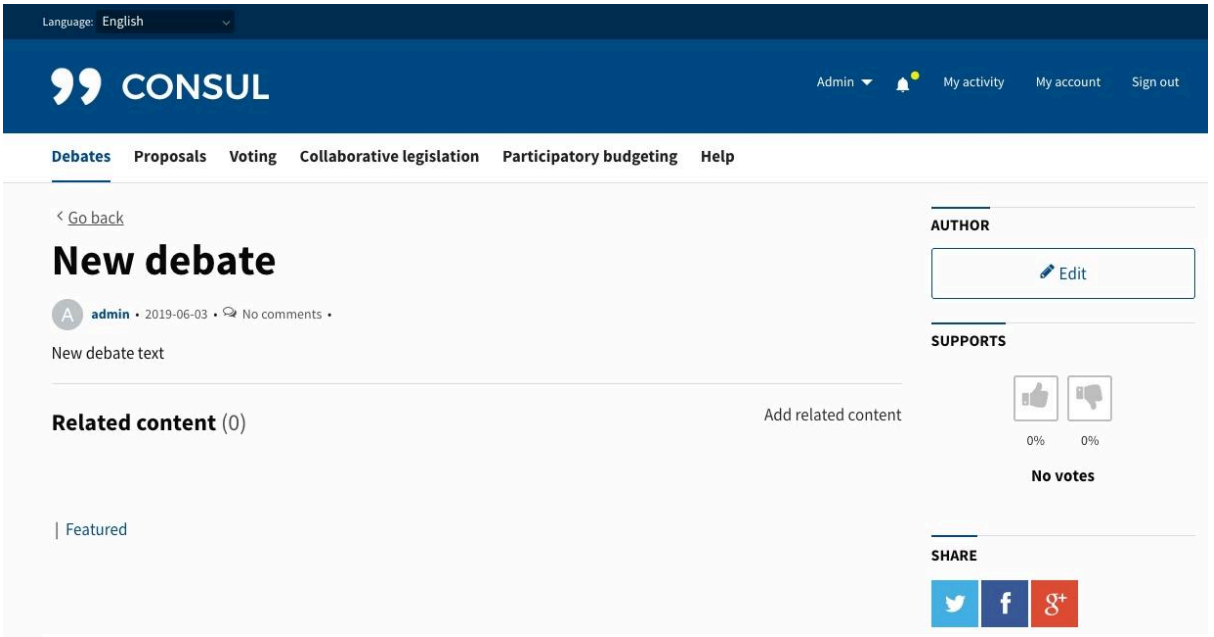
Display notice and text after enqueued translations

If a user visits a page that does not have translations but its translations have already been requested by another user, the application will not show the translate button but an informative text in the header (*explaining when you will be able to see these translations*).



Display text explaining that translations are pending

The translation request, response processing and data saving are processed by background jobs and, as soon as they've finished, the user will be able to read them after refreshing the page.



Display translated content

Available languages for remote translation

Currently these are all the available languages in the translation service:

Copy

```
["af", "am", "ar", "as", "az", "ba", "bg", "bho", "bn", "bo",
"brx", "bs", "ca", "cs", "cy", "da", "de", "doi", "dsb", "dv",
"el", "en", "es", "et", "eu", "fa", "fi", "fil", "fj", "fo", "fr",
"fr-CA", "ga", "gl", "gom", "gu", "ha", "he", "hi", "hne", "hr",
"hsb", "ht", "hu", "hy", "id", "ig", "ikt", "is", "it", "iu",
"iu-Latn", "ja", "ka", "kk", "km", "kmr", "kn", "ko", "ks", "ku",
"ky", "ln", "lo", "lt", "lug", "lv", "lzh", "mai", "mg", "mi",
"mk", "ml", "mn-Cyrl", "mn-Mong", "mni", "mr", "ms", "mt", "mww",
"my", "nb", "ne", "nl", "nso", "nya", "or", "otq", "pa", "pl",
"prs", "ps", "pt", "pt-PT", "ro", "ru", "run", "rw", "sd", "si",
"sk", "sl", "sm", "sn", "so", "sq", "sr-Cyrl", "sr-Latn", "st",
"sv", "sw", "ta", "te", "th", "ti", "tk", "tlh-Latn", "tlh-Piqd",
"tn", "to", "tr", "tt", "ty", "ug", "uk", "ur", "uz", "vi", "xh",
"yo", "yua", "yue", "zh-Hans", "zh-Hant", "zu"]
```

Of all the languages currently available in Consul Democracy (`available_locales`) in `config/application.rb`, the only one that is not listed above and therefore no translation service is offered is Valencian `["val"]`.

Pricing

The translation service used has the most competitive pricing. The price for each 1 Million characters translated is $10 and there is no fixed cost per month.

Although technical measures have been taken to prevent misuse of this service, we recommend the creation of Alerts offered by Azure so that an Administrator can be notified in the event of detecting unusual use of the service. This service has a cost of $0.10 per month.

To create an Alert in Azure we must follow the following steps:

Sign in to the **Azure Portal**.

Access the **Translator** service created earlier.

Go to **Monitoring > Alerts** in the side menu:

Go to **Create alert rule**.

In **Select a signal** select `Text Characters Translated`.

Once selected we must define the logic of the Alert to suit our needs. Ex: Fill "Operator" field with "Greater than" option, fill "Aggregation type" field with "Total" option and fill "Threshold value" field with the number of characters we consider should be translated before being notified. In this section you can also set the time period and frequency of evaluation.

In order to be notified we have to create an **Action Group** and associate it with this Alert we're creating. To do this, access the button **Create** and fill out the form. As you can see there are different types of actions, we must select **Email/SMS/Push/Voice** and configure the option that we consider convenient according to our needs.

Once this group of actions has been created, it is directly associated with the rule we are creating.

Finally, all you have to do is add a name and click on the **Review + create**

Add a new translation service

If you want to integrate more translation services for any reason (new translation service appears, you want to change to include languages that are currently not supported, etc.) the code is ready to be customized.

This is made possible by the `RemoteTranslations::Caller` class which is an intermediate layer between untranslated content management and the currently used Microsoft Translation Client.

A good solution for adding another translation service would be to replace the call to the `MicrosoftTranslateClient` in the `translations` method of `RemoteTranslations::Caller` with the new service implemented.

If you want to manage the coexistence of both, you should determine under which conditions to use one service or the other, either through specific conditions in the code or through a management in the Settings of the application.

Copy

```
class RemoteTranslationsCaller



 ...

 def translations

   @translations ||=
RemoteTranslations::Microsoft::Client.new.call(fields_values,
locale)

   # Add new RemoteTranslations Client

   # @translations =
RemoteTranslations::NewTranslateClient::Client.new.call(fields_val
ues, locale_to)

 end

 ...



end
```

Translation interface

The aim of this feature is to allow users the introduction of dynamic contents in many languages at the same time. From the administration panel you can enable or disable it. If you disable this feature (default configuration) users will be able to enter one single translation.

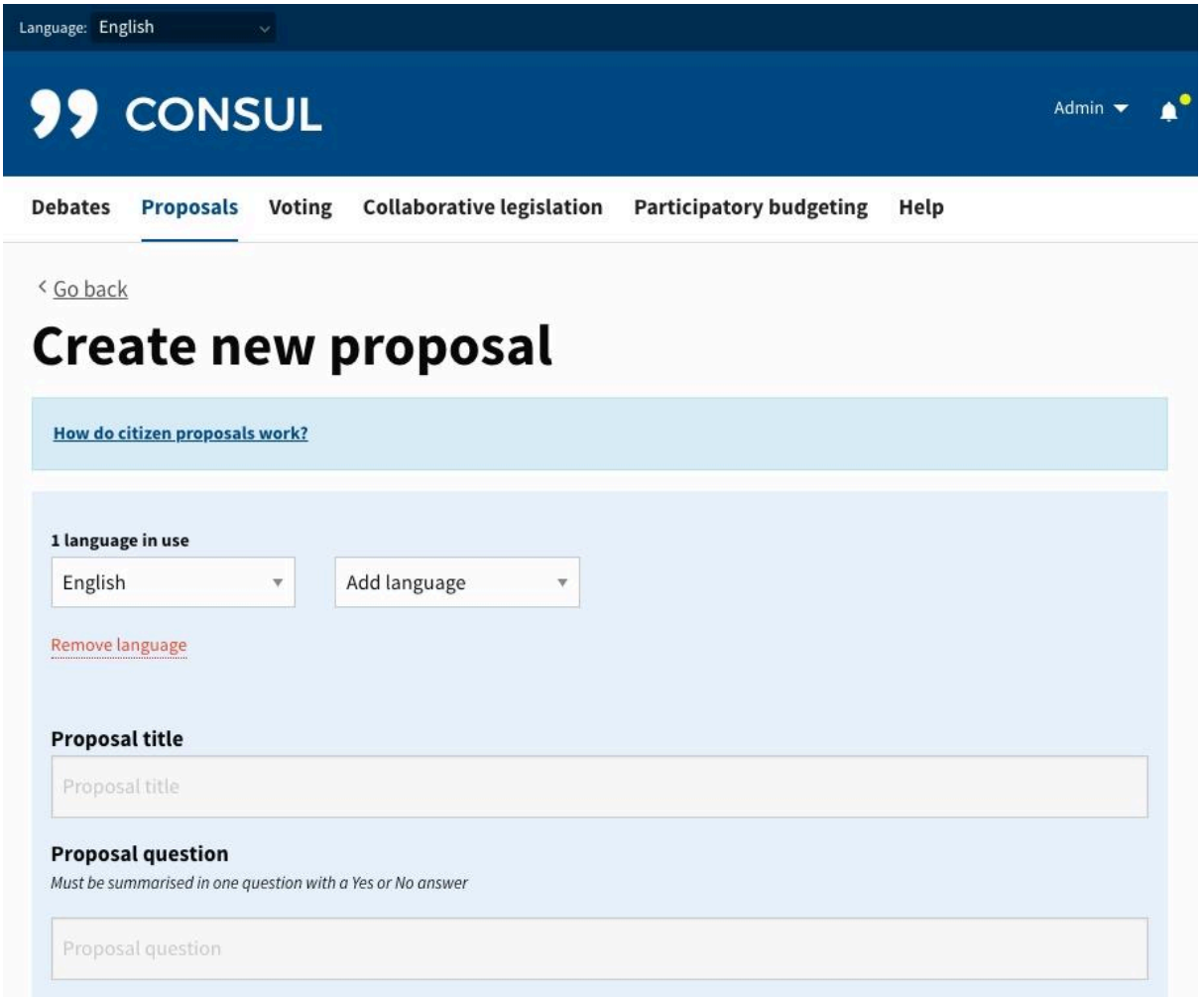
Enabling the feature

To enable this feature you must access from the administration panel to the section **Configuration > Global configuration > Features** and enable the feature called **Translation Interface**.

Use Cases

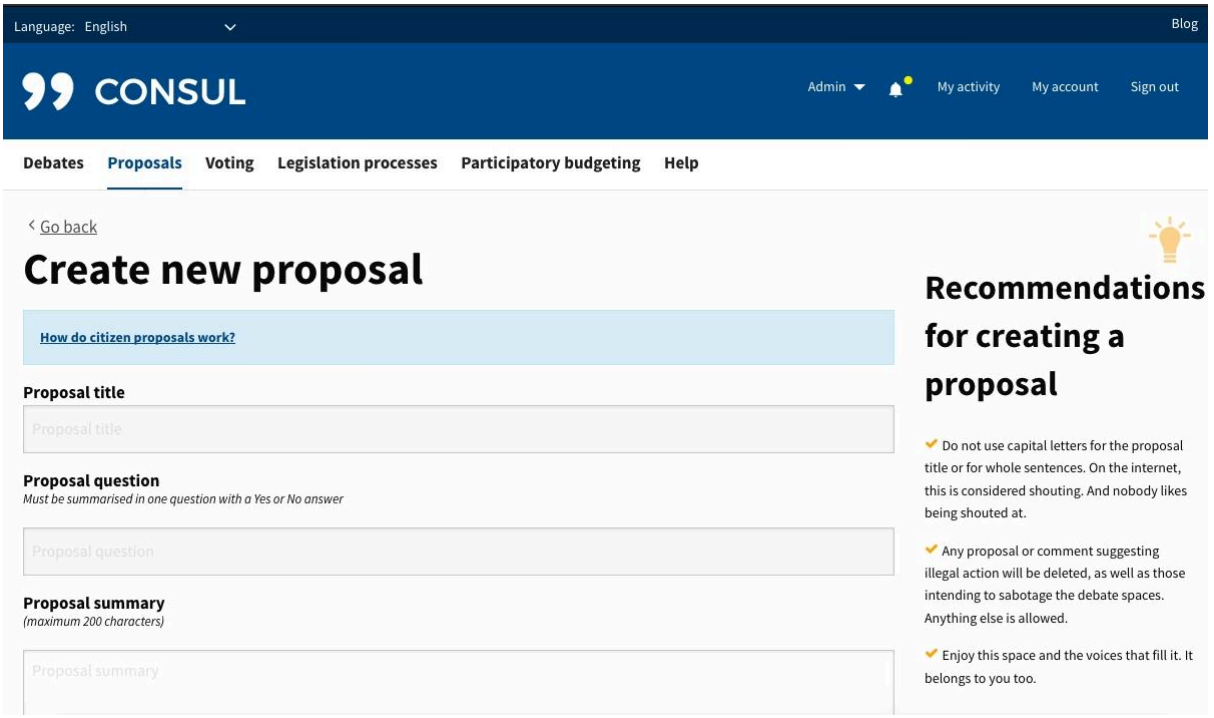
Depending on whether we enable or disable the **Translation Interface** feature we will see the forms as follows:

When the translation interface is active: As you can see in the image below, the translation interface has two selectors, the first one "Select language" is to switch between enabled languages and the second one "Add language" is to add new languages to the form. Translatable fields appears with a blue background to facilitate users to distinguish between translatable and not translatable fields. Additionally, the interface provides a link `Remove language` to delete the current language shown at "Select language". If a user accidentally removes a translation they can recover it by re-adding it to the form.
This feature is visible during the creation and edition of translatable resources.



Translations interface enabled

When the translation interface is disabled: When this feature is disabled users will see standard forms without the translation interface and without highlighted translation fields.



Translations interface disabled

Open Source project

Contributing

We appreciate you want to help us by contributing to Consul Democracy. Here's a guide we made describing how to contribute changes to the project.

Reporting an issue

If you have seen anything wrong in the platform performance or directly in the code, we encourage you to open an issue in the Consul Democracy Github repository.

Before doing it, **please take some time to check the existing issues and make sure what you are about to report isn't already reported** by another person. In case someone else reported the same problem before, if you have more details about it you can write a comment in the issue page -a little more help can make a huge difference!

In order to write a new issue, take into account these few tips to make it easy to read and comprehend:

Try to use a descriptive and to-the-point title.

It's a good idea to include some sections -in case they're needed- such as: steps to reproduce the bug, expected behaviour/response, actual response or screenshots.

Also it could be helpful to provide your operating system, browser version and installed plugins.

Resolving an issue

Issues in Consul Democracy labeled with `PRs-welcome` are well defined features ready to be implemented by whoever wants to do it. In the other hand, the `not-ready` label marks features or changes not well defined yet or subject to an internal decision, so we recommend not to try to resolve them until the admins come to a resolution.

We suggest to follow these steps to keep a good track of the changes you're about to make:

First of all, add a comment to the issue to make everyone know you are going to work on it. If the issue has someone assigned it means that person is already solving it.
Fork the project.
Create a feature branch based on the `master` branch. To make it easier to identify, you can name it with the issue number followed by a concise and descriptive name (e.g. `123-fix_proposals_link`).
Check our coding conventions to help you decide how to write your code.
Work in your branch committing there your changes.
Make sure all tests are passing. In case you're extending or creating a new feature, consider adding its own specs.
Once you've finished, send a **pull request** to the Consul Democracy repository describing your solution to help us understand it. It's also important to tell what issue you're addressing, so specify it in the pull request description's first line (e.g. `Fixes #123`).
Our core team will review your PR and suggest changes if necessary. If everything looks good, your changes will be merged :)

Working on your first Pull Request? You can learn how from this *free* series How to Contribute to an Open Source Project on GitHub.

Other ways of contributing

We'll appreciate any kind of contribution to Consul Democracy. Even if you can't contribute to it coding, you still can:

Create issues about any problem or error you've encountered.

Help translate the platform to other languages you master at Consul Democracy's Crowdin.

Help with Consul Democracy's documentation.

Open Source project

Coding conventions

Linters

Consul Democracy includes linters that define code conventions for Ruby, JavaScript, SCSS and Markdown. Following these conventions makes the code consistent, easier to read and easier to maintain.

When you open a pull request, our continuous integration (CI) system automatically checks that the code in the pull request follows these conventions (you can have a look at the `.github/workflows/linters.yml` file in order to check the exact commands the CI executes). Please check the pull request report generated by these linters.

Since you probably want to check these conventions at all times during development instead of waiting until you've already opened a pull request, we recommend you check whether your text editor has support for these linters. When an editor supports these linters, every time you save a file you'll get immediate feedback about whether you're following these conventions.

A current limitation is that there are two Rubocop (the linter we use for Ruby) rules that aren't followed by old code and one rule that contains false positives. These rules are marked with `Severity: refactor` in the `.rubocop.yml` file; you might want to configure your editor so it ignores them.

If your editor doesn't support these linters, you can also run the `bundle exec pronto run` command, which will only analyze the changes in the current development branch, meaning that you'll get feedback much faster than you'd do if you ran the commands to analyze every file in the project.

Beyond linters

There are code conventions we follow that can't be analyzed by linters. Moreover, old code doesn't always follow these conventions, so sometimes it'll be hard to figure out which way you should proceed.

Here are a few hints that will hopefully help. Don't try to memorize them all at once; instead, check whether these conventions are related to the code you are currently writing.

Only add English and Spanish texts

Most Consul Democracy developers are fluent in English and Spanish but don't know much about other languages. On the other hand, there are many people who are fluent in other languages but don't have the technical knowledge to work with Consul Democracy's source code. That's why we use Consul Democracy's Crowdin to manage translations for languages other than English and Spanish.

Currently, the downside of this approach is that if you include translations for other languages in your pull request, they will be overwritten by the content in Crowdin. So don't bother including texts in other languages in your pull request; once your pull request is merged, use Crowdin to add translations to other languages instead.

Use components instead of views and helpers

You might notice that HTML/ERB code appears in both `app/views/` and `app/components/`. For new code, use `app/components/`. We're gradually moving existing code from `app/views/` to `app/components/`.

There's one scenario when you still need to add code to `app/views/`. Suppose you're creating a new controller named `AwesomeThingsController` that contains the action `index`. In this case, you need to create the `app/views/awesome_things/views/index.html.erb` file, with the following content (note that, in the code below, you might need to pass one or more parameters to the `new` method if the component needs them):

Copy

```
<%= render AwesomeThings::IndexComponent.new %>
```

Then you would create the `app/components/awesome_things/index_component.rb` and `app/components/awesome_things/index_component.html.erb` files.

Similarly, whenever possible, avoid adding helper methods and add methods to a component instead. Since helper methods are globally available, writing a helper method with the same name as an existing one (even if the existing one is present in a different module or even in one of our gem dependencies) will silently overwrite the existing helper. Component methods, on the other hand, are only available inside one component class, which makes them more reliable and easier to refactor.

Structure (S)CSS and JavaScript like components

If you're adding some CSS or JavaScript code that only affects one component, create a file following the same structure as the component. For example, the CSS related to the code in `app/components/admin/budgets/form_component.html.erb` is located at `app/assets/stylesheets/admin/budgets/form.scss`.

Also note that the HTML classes follow a similar structure; in this case, the `app/components/admin/budgets/form_component.html.erb` file contains an HTML

element with the `budgets-form` class (note that we aren't always consistent regarding when to add an `admin-` prefix, though).

Use the Header component concern in new admin pages

When writing a new page in the admin area , using the `Header` concern in components makes it easier to add a proper title and heading to the page. Define a `title` method like:

Copy

```
class Admin::AwesomeThings::IndexComponent < ApplicationComponent

 include Header



 private



   def title

     t("admin.awesome_things.index.title")

   end

end
```

Then you can add `<%= header %>` to the `.html.erb` file, which will set the correct `<title>` tag (which is very important for accessibility) as well as generating the page heading.

Use buttons for non-GET HTTP requests

By default, clicking an HTML link generates a GET request. However, the `link_to` helper provided by Rails accepts a `method:` option; using it will allow you to render links that generate non-GET requests (POST, PUT/PATCH, DELETE, ...). **Do not do this**.

Instead, use the `button_to` helper, which generates a form with a button with native support for non-GET requests. Buttons offer many advantages over links in this scenario:

> People using screen readers will know what the button is supposed to do, but might be confused about what the link is supposed to do.
> Buttons work when JavaScript hasn't loaded or is disabled.
> Most browsers allow opening links in a new tab; for non-GET requests, that will usually generate an error.

Similarly, use buttons instead of links for controls that don't generate any HTTP requests but modify the content of the page using JavaScript (for example, to hide or show certain content).

Use Tenant secrets instead of Rails secrets

Rails manages confidential information by storing it in the `config/secrets.yml` file. Usually, Rails applications access the contents of this file using the `Rails.application.secrets` method.

Consul Democracy, however, is a multitenant application. With a few exceptions, most secrets allow different values for different tenants. In this case, using `Tenant.current_secrets` instead of `Rails.application.secrets` will correctly find different values depending on the current tenant.

Browser support

We try to support 100% of the browsers whenever possible. That means that we don't use ECMAScript 2015 (or later) but use the ECMAScript 5 syntax instead.

However, sometimes, particularly when styling with CSS, supporting 100% of the browsers isn't feasible. In this case, we aim to make the page look as expected on browser versions that are 7 years old or newer (about 98% of the browsers out there), while making sure things don't look too broken everywhere else.

Right-To-Left support

In order to support Right-To-Left (RTL) languages, like Arabic or Hebrew, when writing CSS, use properties like `margin-#{$global-left}` or `margin-#{$global-right}` instead of `margin-left` or `margin-right`. Same with padding and other positional properties. The `$global-left` variable is set to `left` on languages written from left to right and to `right` on languages written from right to left; the `$global-right` variable takes the opposite value.

Until there's universal browser support for them, don't use logical properties like `margin-inline`.

Use rem units instead of rem-calc

In our (S)CSS code, you'll probably find many places using Foundation's `rem-calc` function to convert pixels to rems. Other places directly use `rem` to define rems. At some point, we'll remove every usage of `rem-calc`, so use `rem` instead.

Write component tests for scenarios with no user interaction

For years, we almost exclusively wrote system tests to check the content displayed on a page. However, system tests are very slow and, when Consul Democracy started to grow,

our test suite reached a point where it takes too long to run it on a development machine and we need to rely on a continuous integration system. To prevent the test suite from becoming even slower, only write system tests when testing some kind of user interaction like clicking links or filling in forms. To test that, for example, certain content is rendered for verified users but not rendered for unverified ones, write a component test instead.

Don't check the database after a system test

In general, system tests have four parts:

1. Setting up the database
2. Starting the browser using the `visit` method
3. Doing some kind of interaction
4. Checking the results of the interaction

When checking the results of the interaction, don't check the content of the database (for example, by checking `Proposal.count` after creating a proposal) because it might result in a flaky database exception when the process running the test and the process that started the browser both access the database. Instead, check the content of the page as seen from the user's point of view (for example, check the number of proposals appearing on the proposals index page instead of checking `Proposal.count`).

Checking the database instead of checking the user's point of view can also lead to usability and accessibility issues. For example, a button which modifies the database without giving users any feedback is a usability issue that will be detected when checking the user's point of view. For the same reason, you should try to use reference texts instead of HTML attributes; for example, instead of writing `fill_in "residence_document_number", with: "12345678Z"`, you should write `fill_in "Document number", with: "12345678Z"`. The latter checks that a label is correctly associated with the input, while the former does not.

Avoid simultaneous requests in system tests

When using methods like `click_link`, make sure the request has finished before generating a new one. Getting this point right is really hard (sometimes we don't do it right either), so we really appreciate your efforts regarding this point.

Here's an example:

Copy

```
scenario "maintains the navigation link" do

 visit admin_root_path

 click_link "Proposals"
```

```
 within("#side_menu") { expect(page).to have_link "Proposals" }

end
```

The problem of this test is that the expectation is also true without clicking the "Proposals" link. When writing this code, we probably want the browser to click on the "Proposals" link, wait for the request to finish, and then check the content of the page. That is not what's happening here, though.

Instead, what's happening is: the browser clicks on the link and the test immediately checks the content of the page. If the expectation is met, the test doesn't wait for the request to finish. That means that the request might finish when a different test has started and, after that, anything can happen. For example, the data from the two different tests might get mixed up.

This sometimes results in a so-called "flaky spec", which is a test that fails sometimes but not 100% of the time. Flaky specs are really harmful because, when facing a test suite with a failure during a pull request, you no longer know the cause of the failure, which might not even be related to the pull request.

The example above would be solved with:

Copy

```
scenario "maintains the navigation link" do

 visit admin_root_path

 click_link "Proposals"



 within("#side_menu") { expect(page).to have_css "[aria-current]",
exact_text: "Proposals" }

end
```

This time, the expectation isn't true before clicking the "Proposals" link, so the test waits until the request has finished.

By the way, you might notice that here we're checking an HTML attribute, which seems to be the opposite of what we recommended in the don't check the database after a system test section. However, people using screen readers will be notified about the `aria-current` attribute, so we're actually testing the page from the user's point of view.

Open Source project

Open source forks and modifications

The software ecosystem around the Consul Democracy project is large and wide and emcompasses many software companies, local governments, civic tech non-profits and ngo's who work with the code, modify it to meet their special needs, and, sometimes, bring crucial features back to make their modifications available to the wider community of users and developers.

Here is a preliminary and growing list of open source forks and modifications of our software and its maintainers, that we know exist:

Brasil

- Prefeitura de São Paulo platform 'Participe+'
  - platform: https://participemais.prefeitura.sp.gov.br/
  - code base: https://codigoaberto.prefeitura.sp.gov.br/casacivil/participemais/-/tree/prodam_12_08_24
  - maitainer: Prefeitura de São Paulo (https://capital.sp.gov.br/)

Denmark

- Aarhus Kommune platform 'Sammen om Aarhus'
  - platform: https://www.sammenomaarhus.dk/
  - code base: https://github.com/bellcom/consul
  - maintainer: Bellcom (https://www.bellcom.dk/)

Germany

- Landeshauptstadt München platform 'Unser Muenchen'
  - platform: https://unser.muenchen.de/
  - code base: https://github.com/it-at-m/consul-lhm-dev
- Landeshauptstadt Würzburg platform 'Würzburg Mitmachen'
  - platform: https://wuerzburg-mitmachen.de/
  - code base: https://github.com/StadtWuerzburg/wuerzburg-mitmachen
- Regensburg platform 'Mein Regensburg'
  - platform: https://mein.regensburg.de/
  - code base: https://github.com/StadtRegensburg/consul
  - maintainer all German repos: Demokratie Today (https://demokratie.today/)

Italy

- Comune di Bari platform 'Partecipa Bari'

platform: https://web.archive.org/web/20240805175301/https://partecipa.ba.it/

code base: https://github.com/comunedibari/consul

Mexico

San Pedro Garca-Garcia platform 'Decide San Pedro'

platform: https://decide.sanpedro.gob.mx/

code base: https://github.com/sanpgg/decide

maintainer: San Pedro Garca-Garcia municipality (https://www.sanpedro.gob.mx/)

Netherlands

Provincie Fryslan platform 'Stim fan Fryslan'

platform: https://stimfanfryslan.frl/

Gemeente Oldenzaal platform 'Stem van Oldenzaal'

platform: https://www.stemvanoldenzaal.nl/

Gemeente Hogeland platform 'Stem van Ons Hogeland'

platform: https://stemvanonshogeland.nl/

Gemeente Waadhoeke platform 'Stem van Waadhoeke'

platform: https://stemvanwaadhoeke.nl/

Gemeente Noard-East Fryslan platform 'Stem van Noardeast'

platform: https://stemvannoardeast.nl/

Gemeente Midden-Groningen platform 'Stem van Midden-Groningen'

platform: https://stemvan.midden-groningen.nl/

Gemeente Dantumadiel platform 'Stem van Dantumadiel'

platform: https://stemvandantumadiel.frl/\

shared code base: https://github.com/consul-nl/consul-nl

maintainer of all Dutch repos: Buro Radstake

Romania

Primaria Sinaia platform 'Bugetare Participativ'

platform: https://bugetparticipativ.primaria-sinaia.ro/

Primaria Brasov Elinor platform 'Bugetare Participativ'

platform: https://bugetareparticipativa.primariabrasovenilor.ro/

Primaria Dumbravita platform 'Dumbravita Decide'

platform: https://dumbravitadecide.ro/

shared code base: https://github.com/code4romania/consul

maintainer of all Romanian repos: Code4Romania (https://www.code4.ro/ro)

Slovenia

City of Maribor platform 'Maribor Sodeluj'

Obcine Medvode platform 'Sodeluj in Glasuj'

platform: https://www.mariborsodeluj.si/, https://www.sodelujinglasuj.si/

same code base: https://github.com/danesjenovdan/consul

Obcina Hrpelje-Kozina

platform: https://pp.hrpelje-kozina.si/

code base:

maintainer of all Slovenian repos: Danes Je Nov Dan (https://danesjenovdan.si/en/)

Spain

Ayuntamiento de Madrid platform 'Decide Madrid'

platform: https://decide.madrid.es/

code base: https://github.com/AyuntamientoMadrid/consul

maintainer: Ayuntamiento de Madrid

Ajuntament de Castelló platform 'Decide Castello'

platform: https://decidim.castello.es/

code base: https://github.com/AjuntamentdeCastello/consul

Diputacion de Valladolid platform 'Participa Valladolid'

platform: http://participa.diputaciondevalladolid.es/

code base: https://github.com/DipVa/consul

Ayuntamiento de Zamora platform 'Zamora Participa'

platform: https://zamoraparticipa.com/

code base: https://github.com/jaam1974/consul

Ayuntamiento de Arucas platform 'Participa Arucas'

platform: http://participa.arucas.org/

code base: https://github.com/Usabi/consul_arucas

Generalitat Valenciana platform 'Pla Recuperem Valencia'

platform: https://gvaparticipa.gva.es/

code base: https://github.com/Usabi/consul_gva

maintainer: Usabi (https://usabi.es/)

Ayuntamiento de Las Palmas de Gran Canaria platform 'LPGC Decide'

platform: https://decide.laspalmasgc.es/

code base: https://github.com/LauraConcepcion/consulLPA

maintainer: Usabi (https://usabi.es/)

Scotland

The City of Edinburgh Council platform '

platform: https://yourvoice.edinburgh.gov.uk/

Westlothian Council platform 'Community Choices':

platform: https://westlothiancouncil.communitychoices.scot/

North Ayrshire Council platform 'Community Choices':

platform: https://northayrshire.communitychoices.scot/

Glasgow Council platform 'Community Choices':

platform: https://glasgow.communitychoices.scot/

East Lothian Council platform 'Community Choices':

platform: https://eastlothian.communitychoices.scot/

code base:

maintainer of all Scottish repos: Cosla (https://www.cosla.gov.uk/)

Uruguay

Intendencia de Montevideo platform 'Montevideo Participa':

platform: https://participa.montevideo.gub.uy/

code base:

maintainer: Intendencia de Montevideo (https://montevideo.gub.uy/)

Documentation for Administrators

Type of users

1.1 Users without access to special interfaces

Registered users: They have only provided an email, but their account has not been verified. These users can

Participate in discussions

Create proposals

Verified users: They have verified that they are registered and have the required age. These users can

Participate in discussions
Create proposals
Support proposals
Participate in voting

Public Officers: Any individual user (registered or verified) can additionally be given the role of “Public officer”. These users stand out with a small title banner next to their username in the content they create. For example "Environment Area Technician". Their content is also displayed in a different style to make it stand out. Different levels of "Public officer" can be defined.

Collective: When registering on the platform, you can choose to create an organization user instead of a normal user. By default organisation users, when verified, are highlighted with a small "Collective" banner next to their username in the content they create. They can create debates, proposals or comments, but they cannot support or vote on proposals or projects. In order to verify Collective users, a telephone number and a contact person are required in addition to an e-mail address.

1.2 Users with access to special interfaces

The Consul platform features several possible interfaces associated with different types of users. A more detailed explanation of these interfaces can be found in the Interfaces section.

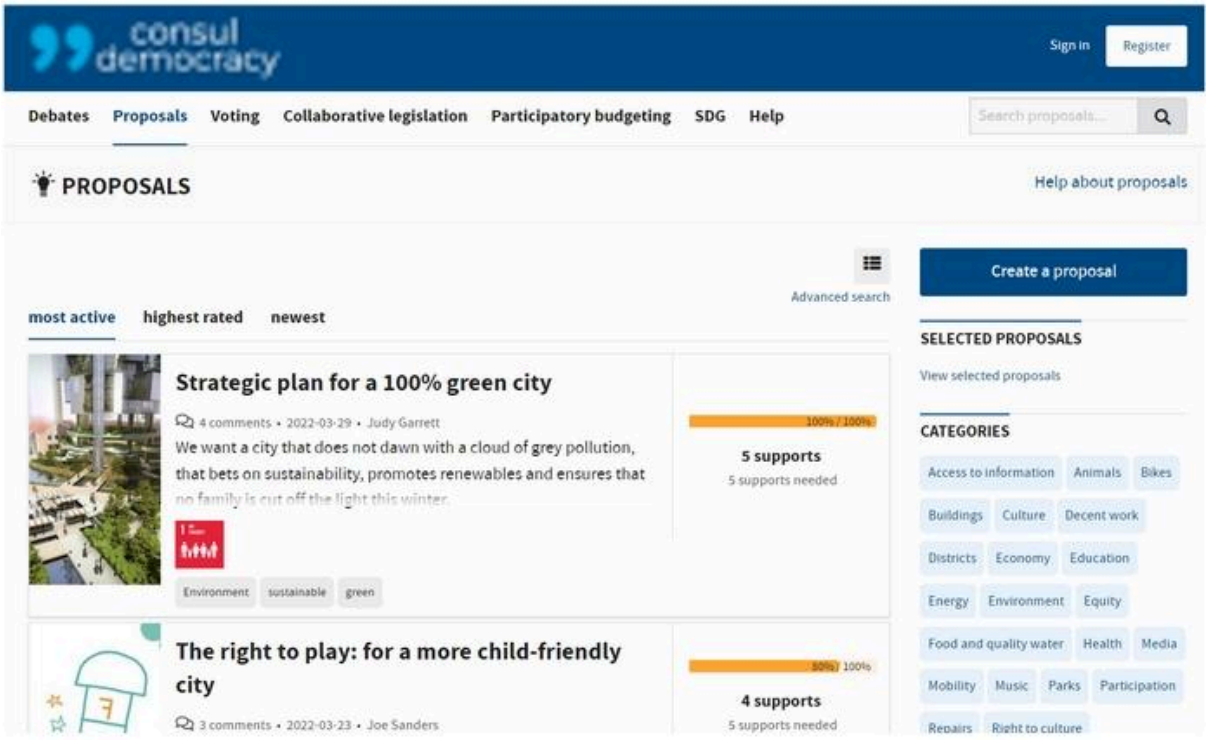
Administrators. These users can access the administrator interface and the rest of the interfaces (management, moderation, evaluation). All platform processes and settings are managed through the administrator interface.
Moderators. Users with this role can access the moderation interface, where all platform content marked as 'inappropriate' by users is listed. In this interface Moderators have options to hide or confirm this content and block users.
Evaluators. This type of user allows access to the evaluation interface for participatory budgets. Through this interface, Evaluators see the proposals that administrators assign to each specific evaluator and can complete the evaluation reports.
Managers. These users have permission to access the managers' interface. In this interface, managers can create and verify user accounts and perform tasks for users to create or support proposals. The most common use of this functionality is by public

workers in the institution's citizen service offices who can help citizens interact with the participation process.

SDG Managers: The software facilitates aligning the content of the platform with the 2030 UN Sustainable Development Goals Agenda as well as correcting the alignment of the content made by users. It also allows creating localized goals, as well as adding content to the SDG section.

Citizen proposals

2.1 Citizen proposals



Overview of citizen proposals from the frontend

2.1.1 What is it for?

The proposals module is designed to collect citizen proposals that require a specific number of "supports" to be processed.

This module is the most complete for collecting proposals since it allows users to include images, documents, videos, locations and categories. It offers a "control panel" of the proposal that allows them to follow how it evolves, alongside the resources and recommendations of a specific project.

Additionally, in each proposal, a "community of users" of the proposal is created to help improve its content, as well as to stimulate its dissemination to obtain more support.

Once the proposal is approved, users can track its development through progress bars and milestones.

2.1.2 Permissions and Tips

Permissions to participate in “Proposals”

Create proposals

Registered users

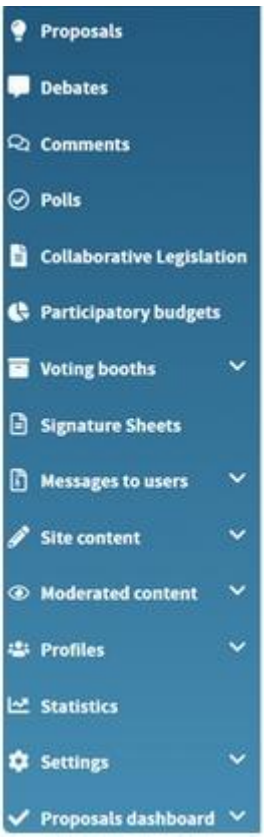
Support proposals

Verified users

CONSUL DEMOCRACY TIPS - citizen proposals

> The default configuration lets proposals go to a vote when the necessary number of supports is reached. It can also be used to gain the required support, and for many other things (discussed in plenary session, studied, etc.).
> You can configure the number of required supports adapting it to your needs. Generally, a high number (such as 5% of the census) is used if reaching that number has important consequences (how to bring the proposal to a vote). If the consequences are less dramatic (how should the proposal be studied by the technicians) a lower number of supports is usually requested.
> Some institutions do not need something specific to happen when reaching a certain number of supports, but want to collect proposals through this module -since it is the most complete- so they modify the code to hide that specific functionality (See good practices)

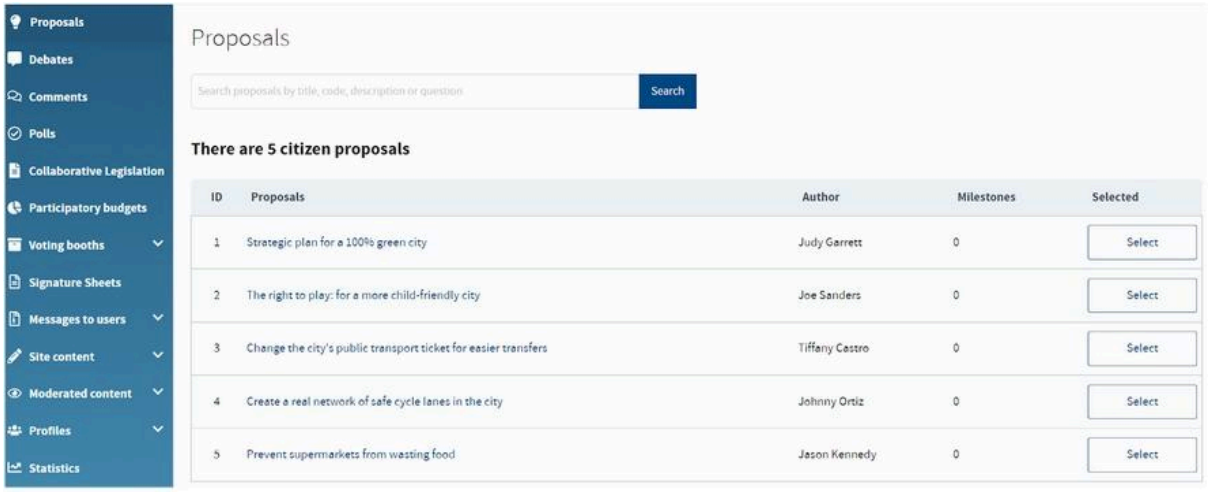
2.1.3 How do I configure it?



The main configurations of the proposals module are made through the administration interface, in the following sections:

- Proposal
- Settings global settings Proposal topics
- Proposal dashboard

Administration> Proposal



Proposal overview within the admin interface

In this "Proposals" section, we see all the proposals created by users (which we should not confuse with the participatory budget projects that have their own section).

We see all the proposals listed, with their basic information and a search engine.

By clicking on the title of each proposal we can see more detailed information about it.

Next to each proposal, we see the "Select" button. Selected proposals will stop collecting support and will be displayed in the "Selected proposals" menu of the "Proposals" section that users see. It is the way to mark that a proposal has received sufficient support or has been selected for another reason.

At the bottom of this information page, we find the Milestones section. This section is used to publish the evolution of the proposals that have been selected. A timeline will be published on the public page of the proposal listing all the milestones.

By clicking on "Create a new milestone", the following information can be added: status (to categorize the proposals, for example: "studying the proposal, contracting, building the proposal, finished"; possible statuses can be defined in the link below), milestone description, milestone post date, images and documents to display in the milestone. The date, status and a description are the only required fields.

Administration >Proposal progress panel.

When creating a proposal, the author can access a progress panel that allows them to follow how it evolves, as well as to collect and make available resources and recommendations needed to make their proposal successful. The first section of the panel is called "Resources and actions" which lists all the available resources and recommendations. The second section is called "Requested resources" which is where notifications will appear when a user requests a resource.

- Resources and actions.

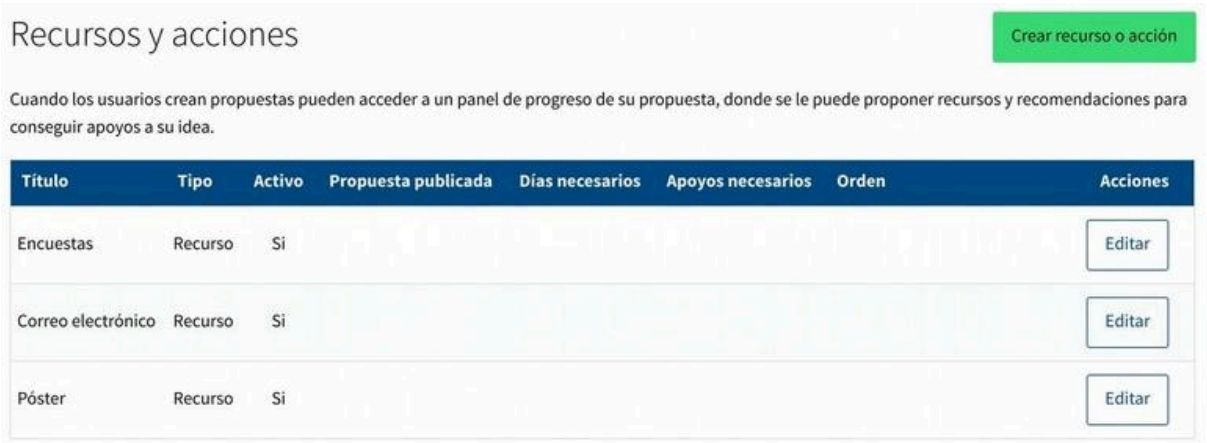
The actions are recommendations that are proposed to the user in order to make his/her proposal successful. Examples of these recommendations are: "Choose a suitable and eye-catching image for the proposal", "Explain the proposal in person to friends and family", "Prepare a social media outreach campaign", etc.

The resources are additional elements that are offered to users that can help the proposal be succesful. Some of these are default elements of the platform itself, like the possibility of carrying out surveys on the proposal or an email and poster format with a certain design to make your proposal visible), Some other resources may be provided by the administration of the platform (for example, resources for recording a short video about the proposal or announcing the proposal on social networks, etc.).

Both categories of resources can be configured to be offered only when the proposal reaches a certain number of supports. In this way, offering the resources can be customized to apply to a different number of users.

Other requirements can also be taken configured when activating the resources, such as whether the proposal is already published or still in draft mode, or that it has been published for a minimum number of days.

If we click on the "Create resource or action" button, it will show us a form where we can define the recommended resource or action.



In the case of resources, the form has a checkbox "Include a button in the resource to request the resource from the administrators". If this box is checked, the user will have to request the resource when it becomes available. Upon request, a notification will be activated in the following section "Resources requested".

- Resources requested.

When a user clicks on the "Request resource" button of one of the resources that include it, a notification will appear in this section. In this way, administrators will be able to manage the resource and contact the user. Subsequently, by clicking on the "Mark as resolved" button, the request will disappear from this list and will go to the "resolved resources" list.

Administration> Global Settings>Global Settings

Number of supports in which a Proposal ceases to be editable

After this number of supports, the author of a Proposal will no longer be able to edit it

Prefix for Proposal codes

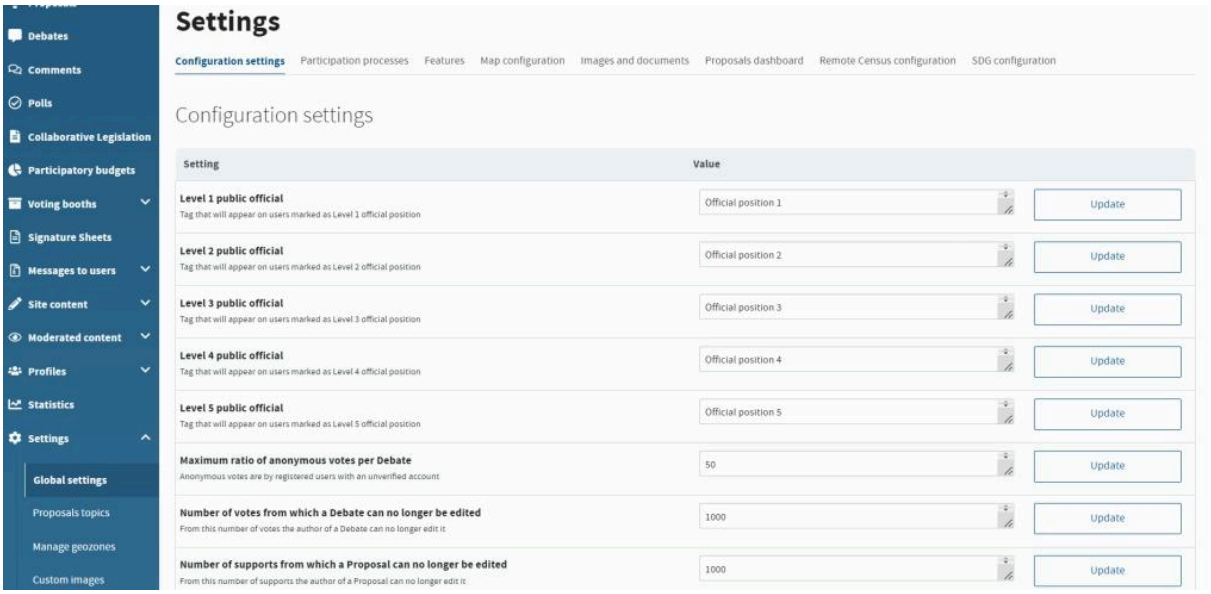
This prefix will appear in Proposals before the creation date and the user ID

Number of supports needed to approve a Proposal

When a proposal reaches this number of supports, it will no longer be able to receive more and it is considered successful.

Administration> Global Settings>Proposal Progress Dashboard

As we have seen, most of the functions of the proposal progress panel are configured in their own section. From the global configuration tab you can define the description of different aspects like manager and evaluator user names, numbers of required votes, etc.



2.1.4 GOOD PRACTICES CITIZEN PROPOSALS

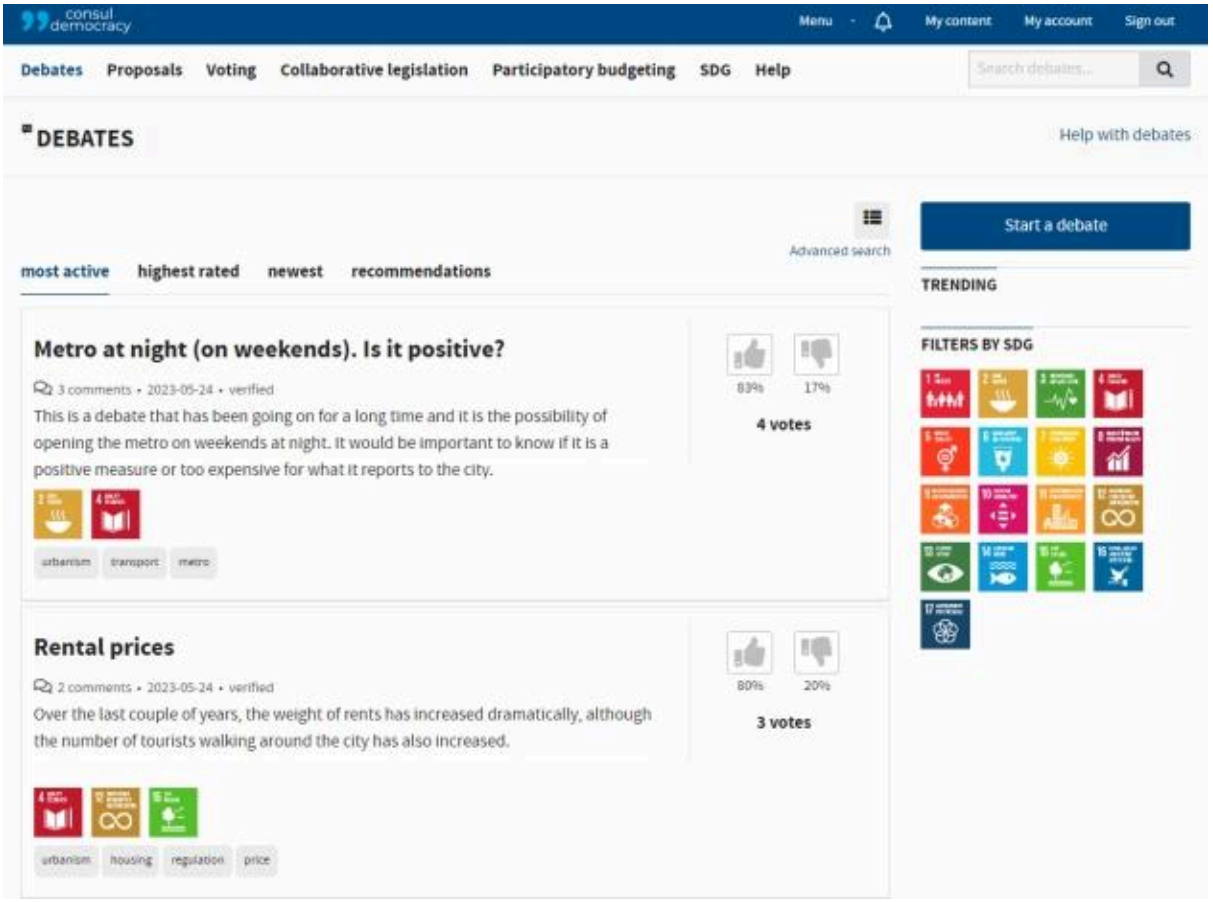
Redpública - Perú *redpublica.pe*

UNDP Peru developed the “Redpública” project to build a citizen agenda through CONSUL's "Proposals” module. The module was modified to remove the need to receive a certain number of supports.

Decide Madrid- Madrid *decide.madrid.es*

The Madrid City Council uses the module to search for proposals that obtain the support of 1% of the enfranchised population (in Madrid: 28,564) on the web. The proposals that receive this support go to a citizen vote.

Debates



2.2.1 What is it for?

The "Debates" module enables citizens to make visible the topics that are important to them, and to find each other to debate or collaborate around these topics. It is a space for listening, but also for meeting and discussion.

Users can vote Debates up or down, which makes the top-rated Debates easily viewed.

The "Debates" module features a comment section, where users can talk and exchange messages and comments. Comments are also voted up or down ranking them organically.

Institutional representatives (administrators or evaluators on the platform) have verified profiles to be able to open the debates they deem appropriate, as well as to respond to any comments. These interventions are highlighted in order to make them stand out from user comments.

In practice, it is a module similar to the "Proposals" module, but it is not focused on collecting a specific number of supports, and it has fewer functionalities (see section 2.2.5 Differences between "Debates" and "Proposals")

2.2.2 Permissions and tipTipss

Permissions to participate in "Debates"

Create debate

Registered users

Support debate

Registered users

CONSUL DEMOCRACY TIPS - DEBATES

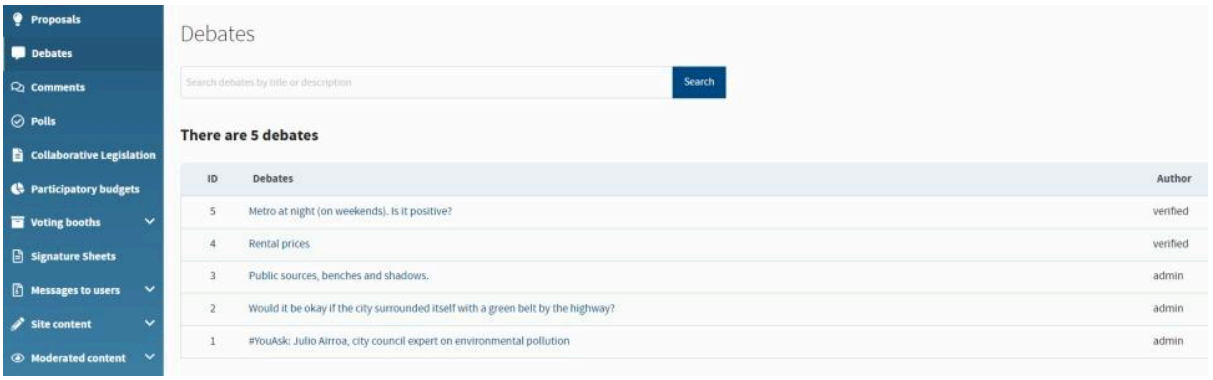
> The functionality of the “Discussions” and “Proposals” modules are very similar. Study your needs well in order to make an informed choice about which process to start.
> In case of opening both, explain well in the help sections what type of content you expect to receive from citizens in each section.
> Take advantage of the verified "Public officer" accounts (see 1.1) to establish direct contact between citizens and the technical and political managers of your organization.

2.2.3 How do I configure it?

The "Debates" module is a simple module that does not require any type of specific configuration.

Administration> Discussions

The "Debates" section of the administration interface shows a list of debates created by users.

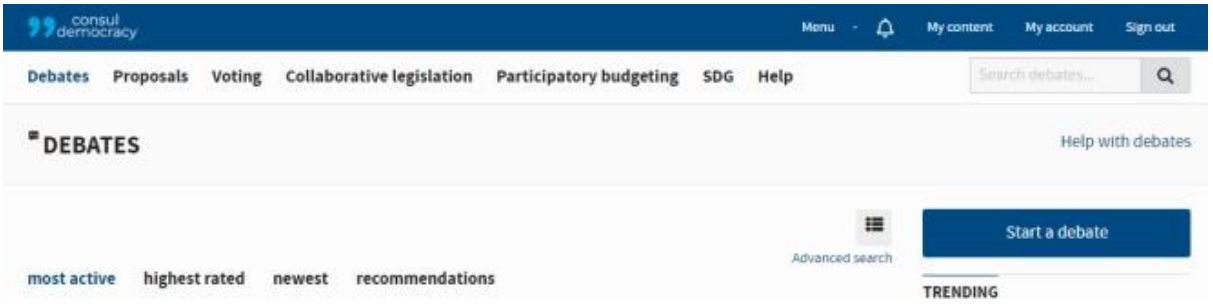


Overview of “Discussions” within the administration interface

Administration > Settings > Global Settings > Features

The only issue to configure in “Debates” is to decide if we want to include a “recommendations” view. If enabled, the user can sort the debates (in addition to newest,

most active, and top-rated) by a new tab, which displays the debates to users in a personalized way, based on their activity.



Overview of “Discussions” within the administration interface

2.2.4 GOOD PRACTICES - DEBATES

2.2.5 Differences between “Debates and “Proposals”

Debates

Proposals

Allows you to establish a specific number of supports required for realization

X

V

Allows you to vote for or against

V

X

Allows the author to include photos, documents or videos

X

V

Administrators can provide the author with resources and actions to disseminate their content

X

V

A follow-up phase with various milestones can be included

X

V

Participatory budgeting

2.4.1 What is it for?

The "Participatory budgeting" module allows citizens to propose and decide directly how to spend part of the institution's budget. Each person can make proposals for projects to spend the budget on and vote on the proposals of others. The proposals most voted for will be carried out.

The budget can be divided into different items or groups of items, and each item can allow a different type of participation. For example, in a city three budget groups are created: one for projects that affect the entire city and two for projects that only affect the North and South districts of the city. In such a case, it can be configured that users can vote for projects from both districts or one of them.

2.4.2 Permissions and Tips

Module of proposals of "Participatory Budgets"
Create proposals
Registered users
Support proposals
Verified users
Vote proposals

Verified users

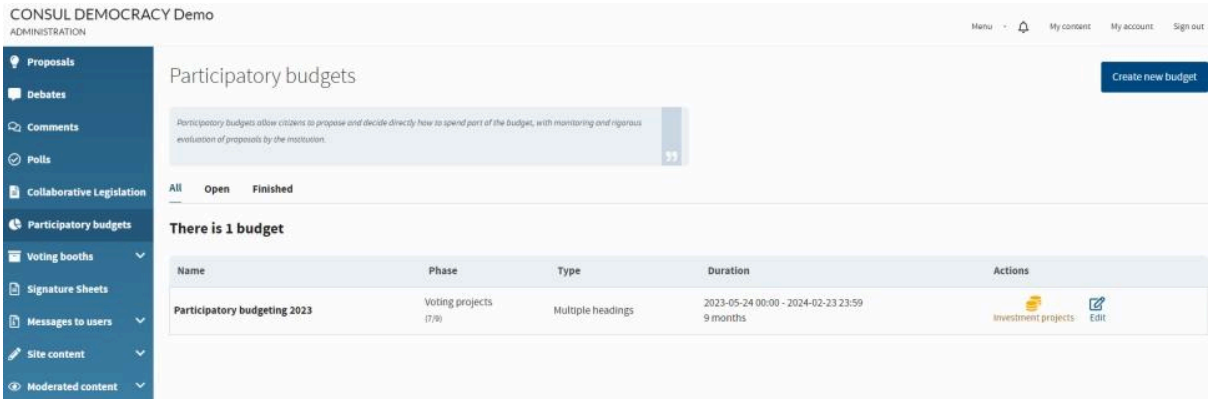
CONSUL DEMOCRACY TIPS - PARTICIPATORY BUDGETS

> Participatory Budgeting is the most complex participatory processes at CONSUL DEMOCRACY. The platform is prepared for a large process that can have up to 9 phases. Reflect on the needs of your process and remember that it is not necessary to activate all of them.
> Keep in mind that this process has automated the sending of certain emails to users, for example when their proposal is qualified as unfeasible. Please check if the messages fit your participation needs. You'll find more information in "Messages to users" in the section "3.1 Administration Interfaces".
> The Participatory Budgeting process allows you to combine face-to-face and online participation. Take advantage of its functionalities to overcome the digital divide. More information in the "Signature sheets" and "Ballot boxes" in section "3.1 Administration Interfaces".

2.4.3 How do I configure it?

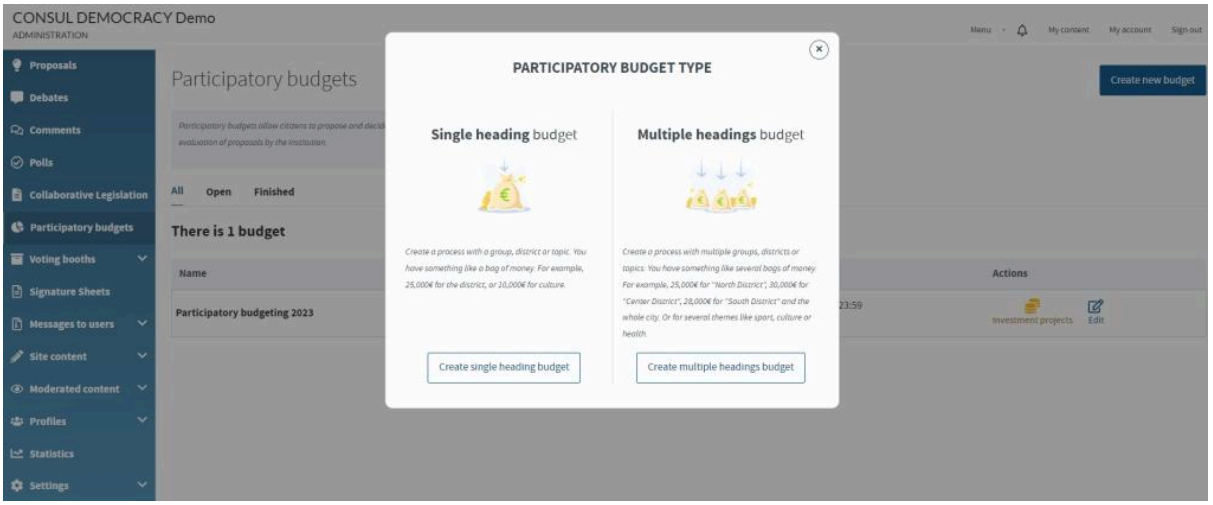
The processes of "Participatory Budgets" are created and configured from their own space within the administrator interface:

Administration> Participatory budgets



Overview of "Participatory Budgets" from the administration interface

We can create a Participatory Budget process from this section by clicking on the “Create new budget” button at the top right.



Window when clicking "Create budget"

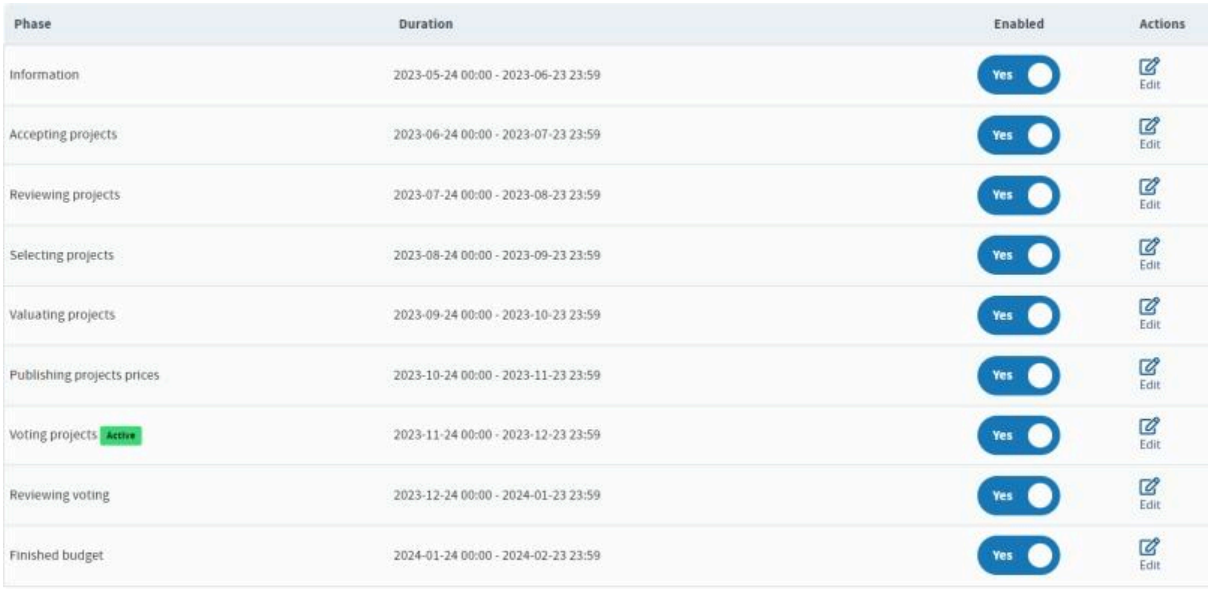
The first thing we have to configure is if our budget is going to be divided into different items (e.g. a budget for each neighbourhood or area) or if it will have a single item for the entire city or region.

The next steps are adding the basic information of the budget (name, currency, type of vote) and creating and configuring different groups and items of the budget. Groups refer to the type of budget, not to the items in a group itself (e.g. the district budgets).

Example of groups

In the image above, "City projects" and "Districts project" both are a "Group" and each of the four districts would be a "Headings".

Once we added and customized the "Headings", we will configure the different "Phases" of our budget. The process can include the following phases (all optional):



Phase	Duration	Enabled	Actions
Information	2023-05-24 00:00 - 2023-06-23 23:59	Yes	Edit
Accepting projects	2023-06-24 00:00 - 2023-07-23 23:59	Yes	Edit
Reviewing projects	2023-07-24 00:00 - 2023-08-23 23:59	Yes	Edit
Selecting projects	2023-08-24 00:00 - 2023-09-23 23:59	Yes	Edit
Valuating projects	2023-09-24 00:00 - 2023-10-23 23:59	Yes	Edit
Publishing projects prices	2023-10-24 00:00 - 2023-11-23 23:59	Yes	Edit
Voting projects Active	2023-11-24 00:00 - 2023-12-23 23:59	Yes	Edit
Reviewing voting	2023-12-24 00:00 - 2024-01-23 23:59	Yes	Edit
Finished budget	2024-01-24 00:00 - 2024-02-23 23:59	Yes	Edit

Phases Participatory Budgets

Information. Publishing the basic information about the process before the participation phase begins.

Accepting projects. Citizens can submit budget spending projects.

Reviewing projects. During this phase, it is no longer possible to present projects, but they cannot be supported or voted for yet. It is a useful phase so that all users can see their published projects, and any errors that exist can be detected and corrected.

Selecting projects. This phase allows users to support projects and prioritize them for evaluation in the next phase. It is common to receive more projects than the ones that can plausibly be evaluated by the number of active participants.

Evaluating projects. During the evaluation period, the projects can be evaluated by the institution and its administrators. This evaluation enables marking the projects as feasible or unfeasible, and to assign execution costs. The evaluated feasible projects can proceed to the voting phase. The evaluation phase is carried out from the evaluation interface (Section 3.3)

Publishing project costs. During this phase, the selection of projects that pass to the final vote is publishd, including the costs assigned in the previous phase, giving users the opportunity to detect possible errors.

Voting projects. Users can vote on the projects to decide the final selection of projects to be carried out using the participatory budget.
Reviewing voting. Phase to verify the results.
Results. Phase to present the final results once the process has finished.

2.4.4 GOOD PRACTICE

PARTICIPATORY BUDGETS

"Digital Participatory Budget" - Porto Alegre (Brazil), *https://opdigital.prefeitura.poa.br*

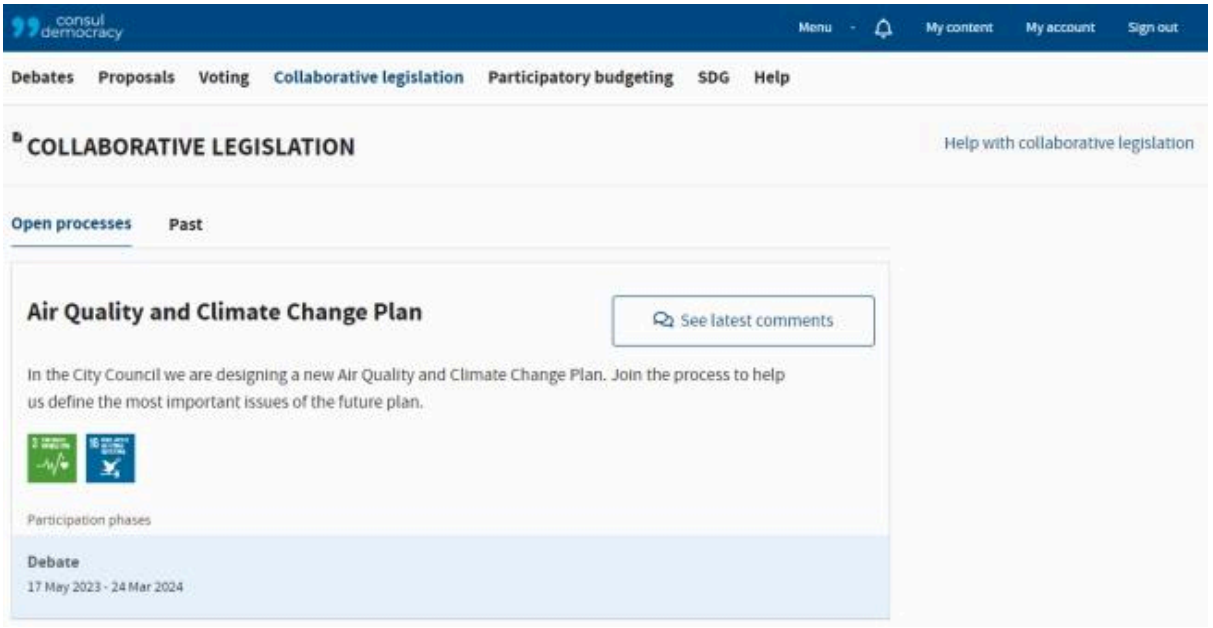
The City of Porto Alegre was the first public institution in the world to implement a Participatory Budget, in 1989. Its 30th edition in 2019 was carried out through this CONSUL DEMOCRACY module. Ten public policy priorities (instead of Projects) were defined within six specific themes:

Circulation, Transport and Urban Mobility
Health and Social Assistance
Education, Sport and Leisure
Culture and Youth
Economic Development, Taxation, Tourism and Labor
Housing, City Planning, Urban and Environmental Development

"Let's make a city with your ideas" - San Pedro (Mexico), https://decide.sanpedro.gob.mx

San Pedro carries out, through CONSUL DEMOCRACY's "Participatory Budgeting" module, a highly detailed citizen budget process including 177 budget items ("Headers") corresponding to the 177 Neighborhood Councils divided over San Pedro's six city districts.

Collaborative legislation

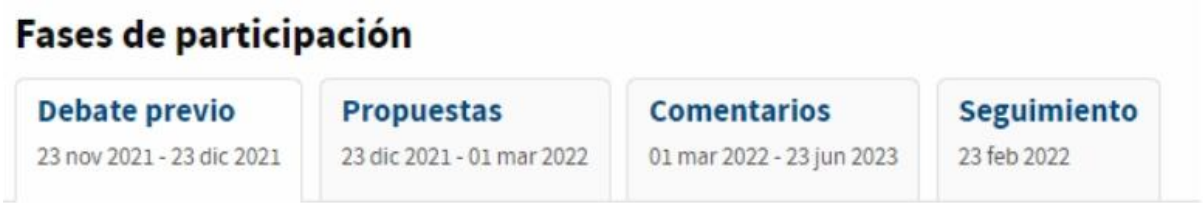


Overview of “Collaborative Legislation” from the frontend

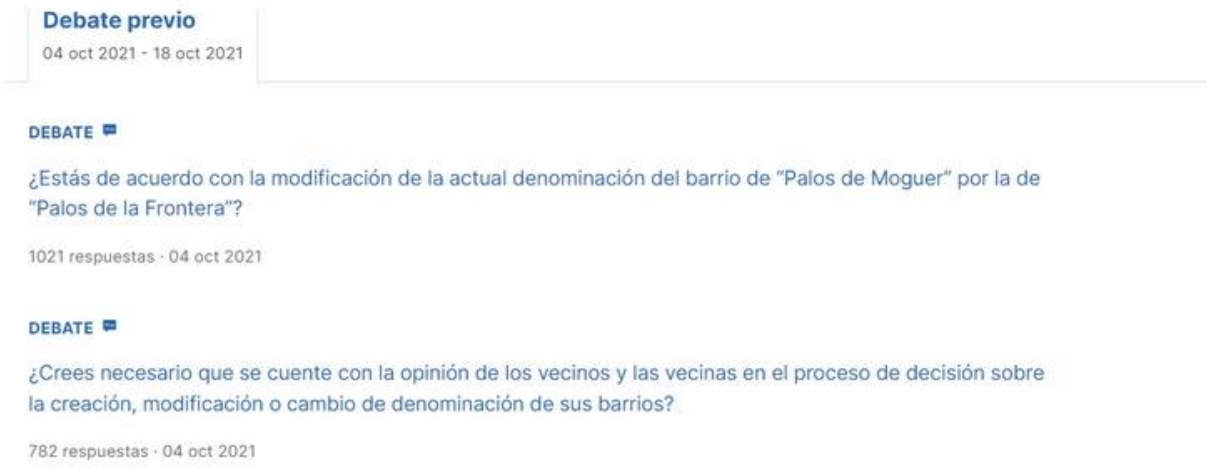
2.3.1 What is it for?

The "Collaborative legislation" module is the platform's most versatile one. Although specifically designed to organize participative processes around creating formal rules, laws or regulations, the module is also fully equipped to open any type of text to citizen contributions and, more generally, even to receive contributions to any (non-legislative) initiative that is launched by the public institution.

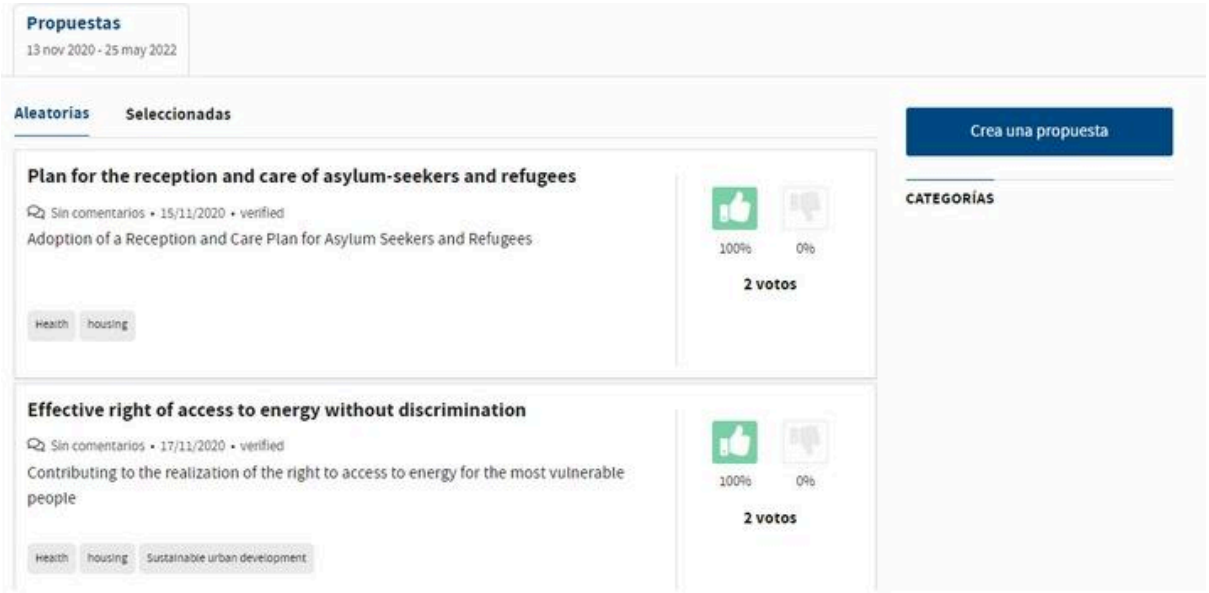
The module allows administrators to configure each "Collaborative Legislation" process with up to five phases (all optional). Once the participative process is finished, a fourth "Follow-up" phase can be added too.



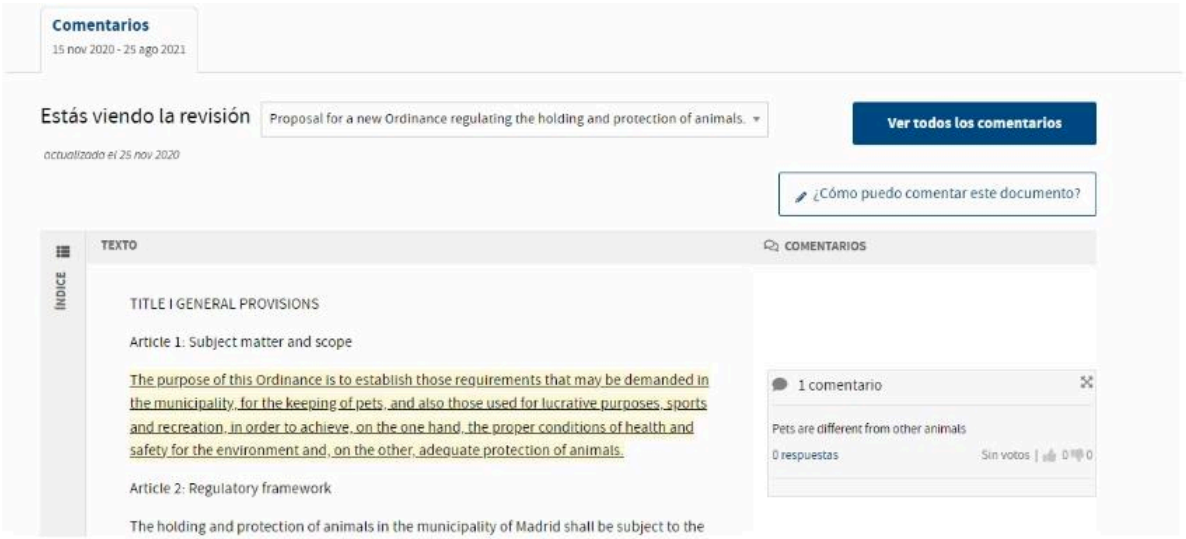
> The **Debate phase** allows opening several questions. Users can give open-ended answers to questions or select possible pre-existing answers, and vote on other users' answers.



The **Proposals phase** allows users to write proposals for the document's text and support proposals from other users.



The **Drafting phase** allows you to publish a text. Users can select any part of the text, make comments and vote on other users' comments.



Follow-up phase this section is used to publish information about what has happened after the participation process. It allows administrators to include milestones and percentage progress bars.

Additionally, a "Draft" phase where the content is not yet public, and which allows selecting a "presentation" phase where the process is published, but no proposals or comments are allowed.

2.3.2 Permissions and Tips

Government proposals module

Discussion phase

Answer questions posed

Verified users

Proposal phase

Create proposals

Registered users

Support proposals

Verified users

Feedback phase

Comment text

Registered users

Support comments

Registered users

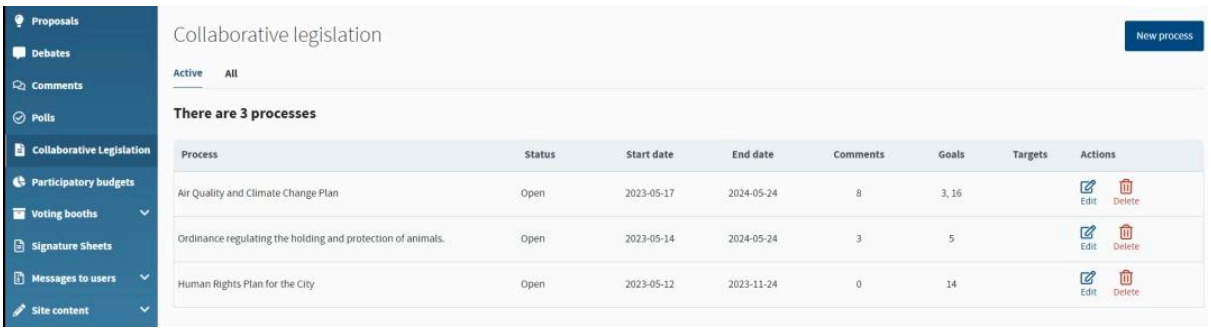
CONSUL DEMOCRACY TIPS - COLLABORATIVE LEGISLATION

> This module is very versatile and allows opening diverse collaborative drafting processes. Consider whether the name “collaborative legislation” is the one that best suits the participation process you're looking to create. You may opt for a more generic name such as “Municipal proposal” or just focus on the topic at hand. The module's name cannot be modified from the backend.
> Although usually the different phases are correlative (one begins when the previous ends), all the phases can be open at the same time.
> Note that the "Debates" and "Proposals" phases are reminiscent of the similarly names modules. However, they differ in two important respects:
>
> - Discussions here are opened by the administration (not users).
> - The proposals here do not seek a specific number of supports.

2.3.3 How do I configure it?

The "Collaborative Legislation" processes are created and configured from their own space within the administrator interface:

Administration> Collaborative Legislation



Overview of “Collaborative Legislation” within the administration interface

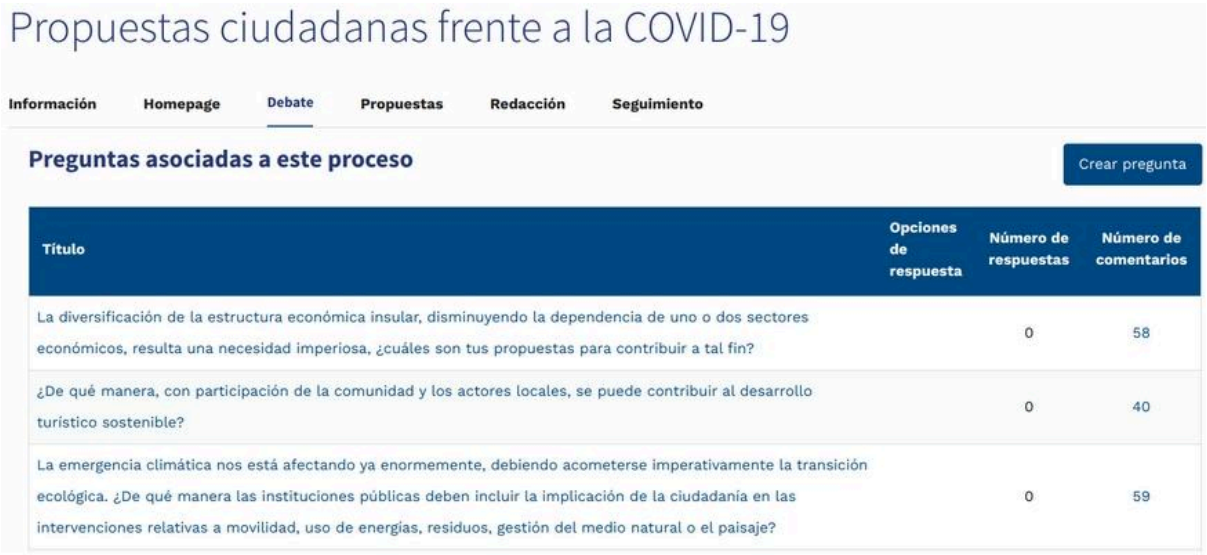
By clicking on the "New process" button (top right) you can create a new Collaborative Legislation process. Next, you can define which phases will be active (by clicking on the "enabled" box for each phase), the start and end dates of each phase, the information that will be displayed on the process page (title, process summary, basic description and additional information), the basic style of the process page (header and text colours) and additional documents to be added in case it is useful in that same public page as additional information. Finally, we can also align the process with one or more of the objectives of the UN 2030 SDG Agenda.

Once the process is created, by clicking on it you can see different tabs to define each phase:

Tabs of the collaborative Legislation process "Citizen Proposals against COVID" once created

Information. Features all the basic information entered while creating the process in the previous step.

Homepage. In addition to the general header with information on the process, it can have a first 'home' page where we give more detailed information before entering any of the phases. This tab shows a box to tick and activate the homepage and a field where we can enter the page's content: text, images, links and tables.

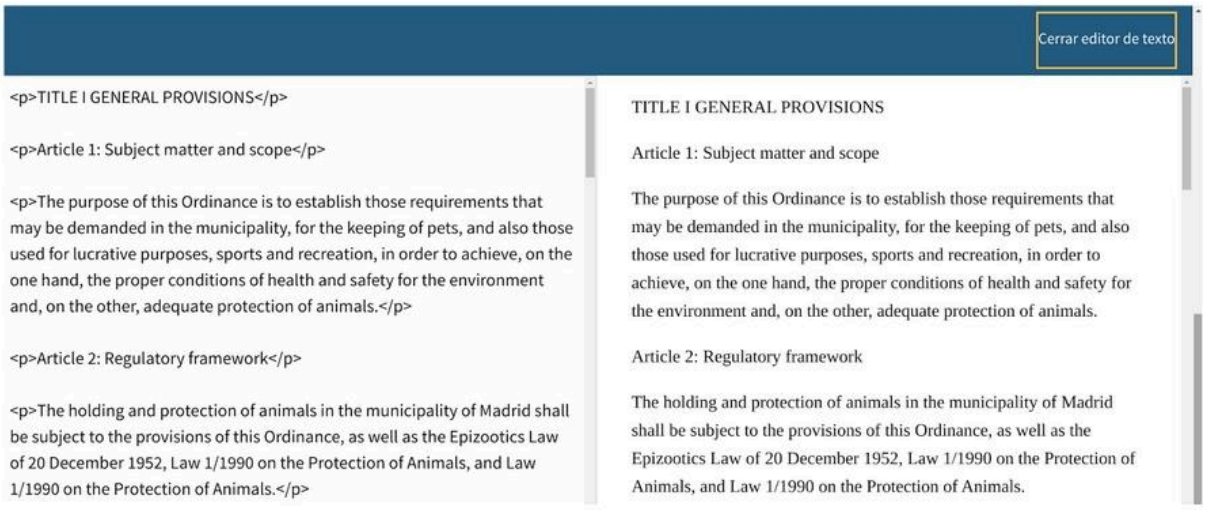


“Discussion” tab

Debate. The "Create question" button allows you to add new questions to the phase. It is possible to define closed answers to the question, but this is optional. By default, answers will be open.

“Proposals” tab

Proposals. The "Categories" field defines which categories will be suggested to the user when creating a new proposal. The categories will be listed during the process in order to filter proposals in those categories.



“Composition” tab

> **Drafting**. The Drafting phase allows you to upload the first version of the text and then modify it and publish new versions. To start the process, click the "Create Version" button. At the top, you can write the title of the text and there is an optional field to write a summary of the changes compared to the last version. The "Status" option selects between "draft" (the text will not be public, only admins can preview it) and "published". Once the process is finished, the "Final version" checkbox allows you to publish the final text (comments will not be allowed here).

The "Launch Text Editor" link can be used to load the text. Clicking on it will split the window into two parts. The left part is used to enter the text, and the right part will show how it will look on the web page. To add styles to the text (such as bold letters, italics, etc.) you must use the "markdown" format. It may look very different from traditional text editors, but it's easier than it sounds. When you have the text (or part of it) ready, click on the "close text editor" which will save your changes automatically and then on the "Create version" button.

> **Follow-up**. This section is used to post information about what has happened after the participation process.

First of all, a text field is displayed where a generic summary text that will appear at the top of the Follow Up page. This text will only be visible if there is a defined milestone, as we will explain below.

By clicking on "Create a new milestone", the following information can be added: status (to categorize the projects, for example: "studying the project, contracting, building the project,

finished"; possible statuses can be defined following the "Manage statuses" link), milestone description, milestone post date, images and documents to display in the milestone. The date and description (or status instead) are the only required fields.

We will also find the "Manage progress bars" button that allows us to add percentage bars to the Milestones that indicate how much of each has been completed.

The Milestones section will display a timeline on the project's public page with all the milestones.

2.3.4 GOOD PRACTICE

COLLABORATIVE LEGISLATION

"Citizen proposals against COVID" - Gran Canaria *participa.grancanaria.com*

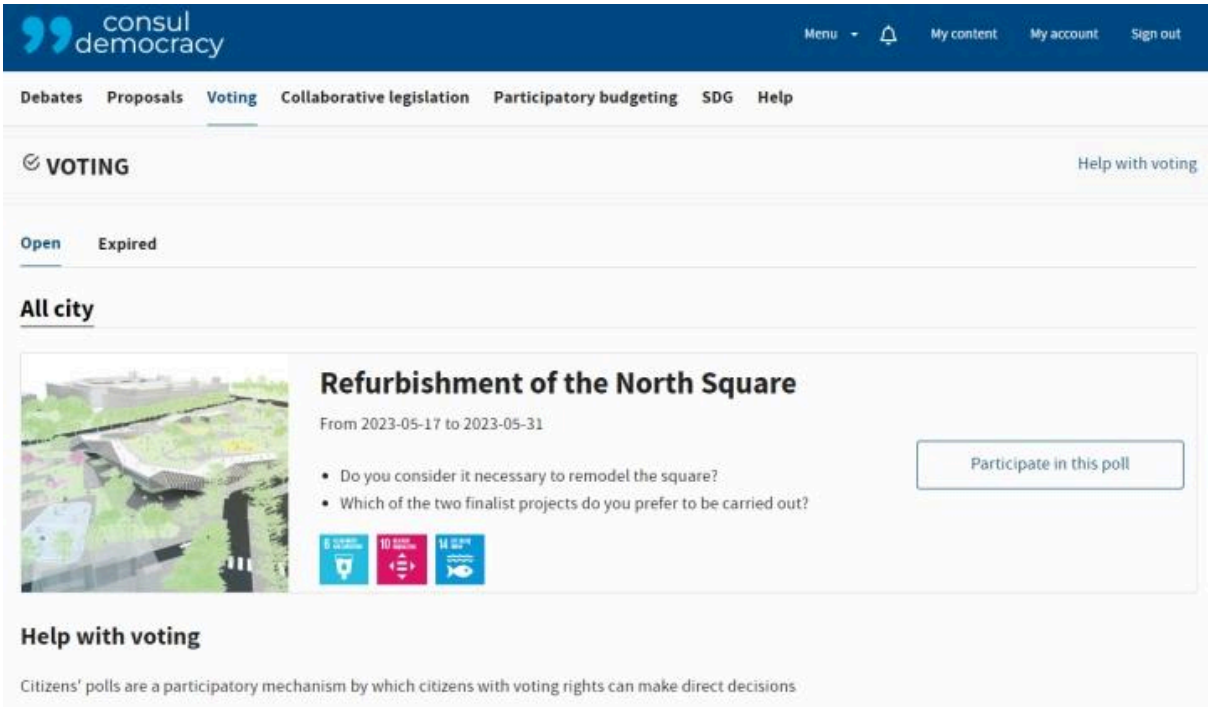
The Cabildo used this module to develop a process in which citizens made proposals on how to combat COVID-19, which were later discussed in a Workshop-Lab for Participatory Design of Public Policies using a simultaneous face-to-face and virtual format. The proposals were worked on and debates arose from the consultation, the result of which was evaluated by the insular government.

"Influencing Europe"- Carlos Antwerp Foundation

decidefcamberes.org

The Carlos Antwerp Foundation used the "Collaborative Legislation" module to collect proposals from citizens on how to modify the Digital Services Law (DSA) and the Digital Market Law (DMA) and take them to the European Parliament.

Voting



Overview of “voting” from the frontend

2.5.1 What is it for?

The voting module allows users to vote for citizen proposals or specific questions that the institution wants to raise.

In each vote we can include as many closed-answer questions as we see fit.

2.5.2 Permissions and Tips

“Voting” proposal module

Vote

Verified Users

CONSUL DEMOCRACY TIPS - VOTING

Given the ease of configuration of the "Voting" module, it can be set up rapidly and be used for many different purposes, ranging from more 'provocative' questions associated with referenda, to more everyday issues (such as the type of light bulbs that the City Council should acquire) or to organize contests to be resolved by popular vote.

Make it clear from the beginning whether or not the vote is binding to avoid misunderstandings.

Keep in mind that this module is designed to carry out voting, not to carry out surveys. Since it only includes closed answers, the results are shown aggregated.

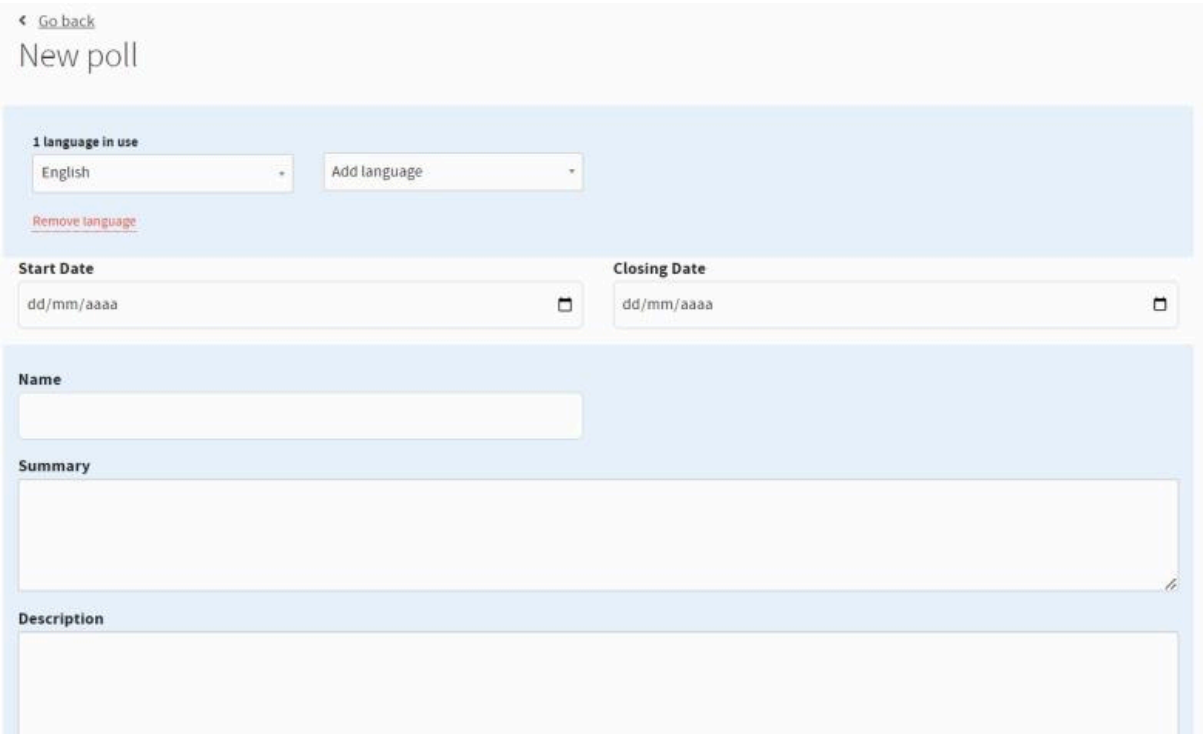
2.5.3 How do I configure it?

The "Voting" is created and configured from its own space within the administrator interface:

Administration> Voting

Overview of "voting" from the admin interface

We can create a vote from this section by clicking the "Create poll" button on the top right.



Window when clicking "Create vote"

The first thing we have to configure is the basic information of our vote. "Name", "Summary" and "Description" will be the public information presented on the voting page (each voting has its own web page; all of them are listed in the voting section of the platform). Online voting will be possible between the "Start Date" and the "End Date". You can upload an "Image", which will be displayed in the voting list. The "restricted by geozone" checkbox allows you to restrict the users who can vote to a specific area. We can also align our voting with the 2030 UN SDG Agenda.

Once a vote has been defined, it is presented with tabs to the subsections 'questions', 'booths', 'officers', 'recounts' and 'results' (the last two once the vote has started).

Creation of voting questions”

The "Create question" button allows you to create a new question. In the first form, only the question itself is needed. Once created, we will see an "Edit answers" button next to the question. In the following form, you can add the possible answers with the "Add answer" button. Each response can include a description, images, documents, and videos. All this information will accompany the answers and be displayed on the voting page. To upload material, click on the "Image List", "Document List" or "Video List" links to the right of each created answer.

(If you want to enable voting booths in addition to digital voting, this can be configured through the "Voting booths" section, which we will explain later.)

In the 'recounts' and 'results' section, you can find links to see the (provisional) results of the votes for each question. Keep in mind that the 'recount' section differentiates between two types of recounts: recounts (carried out by voting officers) and votes (automatically recorded by the platform). This comparison helps verify whether the process was successful. Once the voting is completed, there are two boxes in the results section to make public the results and the statistics of the voting. Those results will be displayed on the public voting page.

2.5.4 GOOD PRACTICE

VOTING

“How do you want the new lighting of the Old Town to be?” - Arcos de la Frontera
http://decidearcos.es/

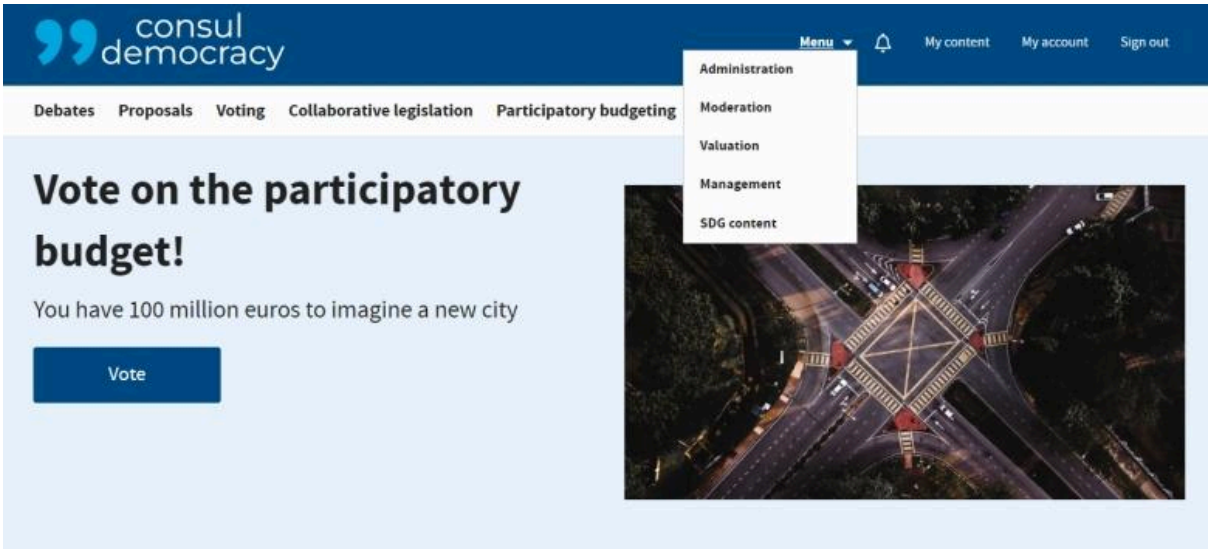
The Town of Arcos de la Frontera inaugurated its CONSUL DEMOCRACY platform by voting on the type of light bulbs that illuminate its historic centre, an issue of singular importance in the historic Andalusian town.

“Concurso de repostería” Palma de Mallorca

https://tufas.palma.cat/

The City Council of Palma de Mallorca organizes through CONSUL DEMOCRACY a contest to choose the most representative cake for its festivities. To do this, it opened a vote for citizens to choose between the different proposals made by the city's pastry shops.

Administration



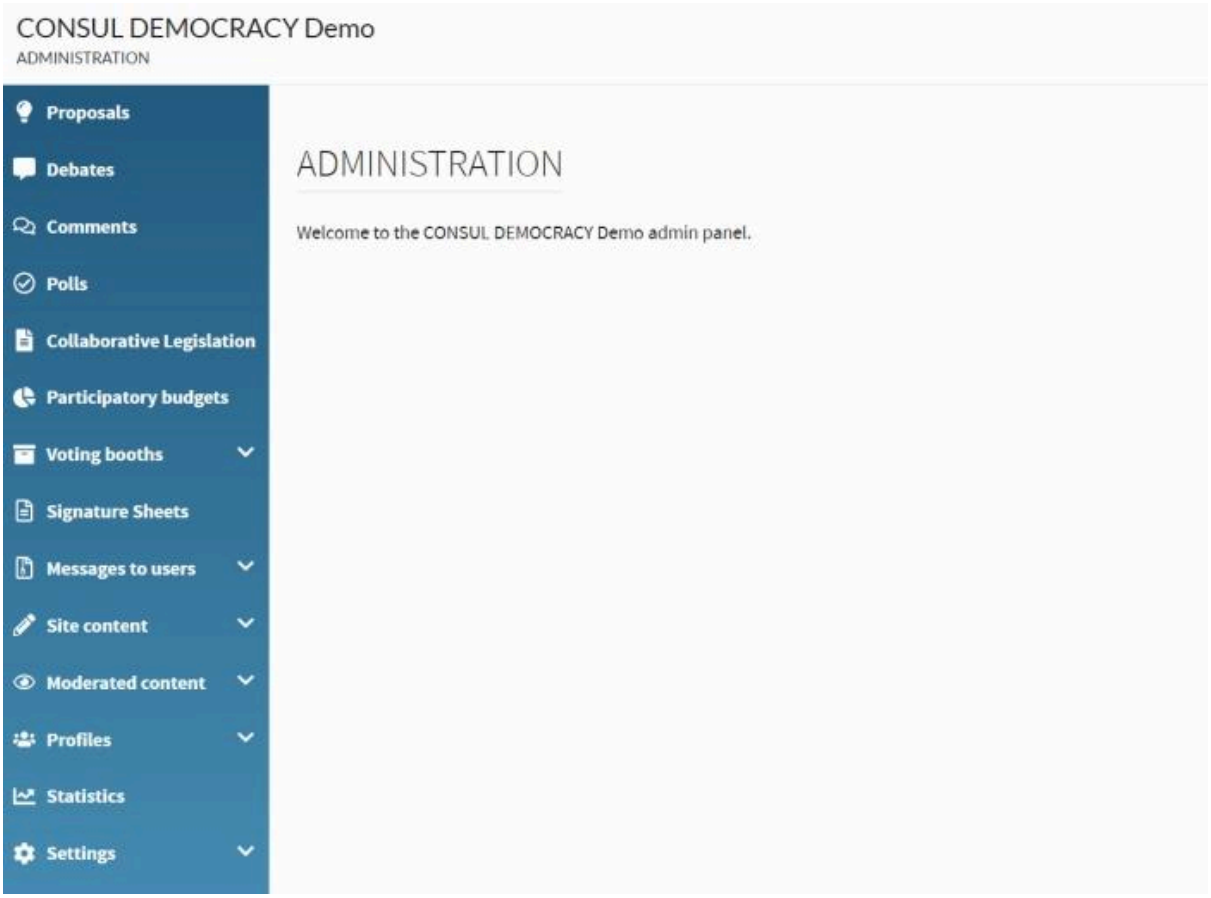
Interface menu (only visible to users with administration permissions)

We are now going to see the different independent interfaces that CONSUL DEMOCRACY has, from which everything that happens on the platform can be easily administered and managed. There are six separate interfaces: Administration, Moderation, Evaluation, Management, Poll Officer and SDG Content.

Administrators can access all of them and additionally assign permissions to other users to access specific interfaces. For example, a moderator of a given participation process can be assigned permission to access the Moderation interface.

In the following sections, we will explain the use of each of the interfaces.

3.1 Administration



administration interface

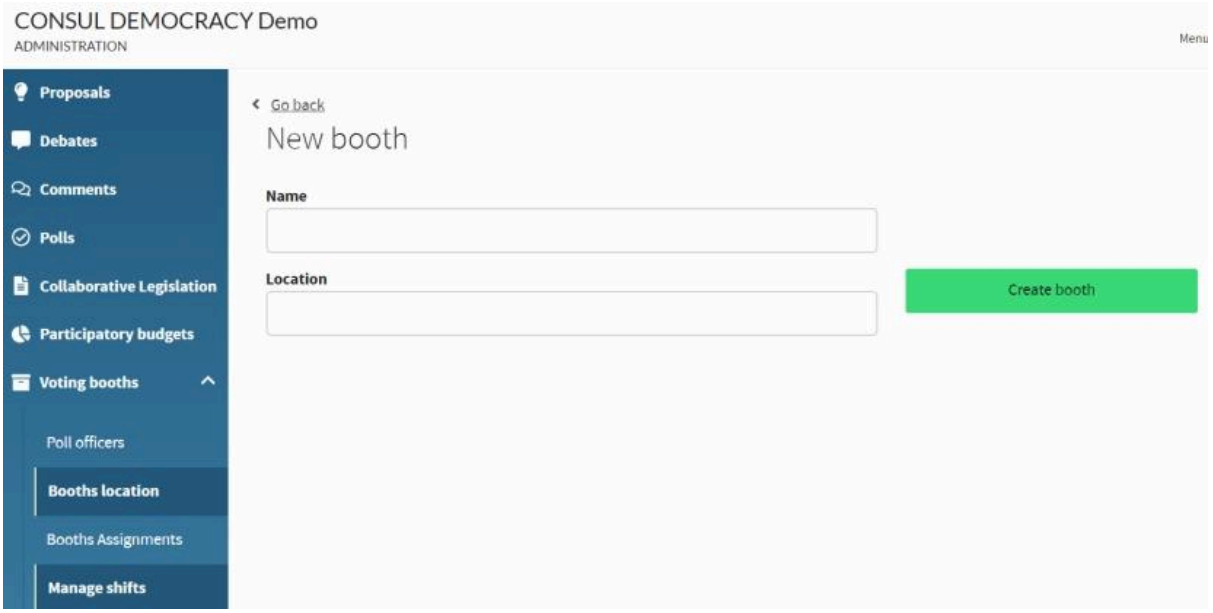
In the previous sections, we have already discussed the sections of the Administration interfaces related to the five participation modules (proposals, debates, voting, participatory budgets and collaborative legislation).

Let's see the rest now:

3.1.1 Voting booths

This section allows you to enable offline voting booths both for the "Voting" module and the "Participatory Budgets" module. The Voting booths are overseen through the "Poll Officer" interface to ensure that no person can vote twice (at the ballot box and/or on the digital platform). It also allows monitoring of the evolution of a voting in a decentralized manner.

To use offline voting booths (and to be able to see the Poll Officer interface), we must first create a voting. This is done in the "Polls" menu (Create Poll) or in the "Participatory Budgets" menu (enabling the 'Voting projects' phase). Once the voting is created, you can add Voting booths, Poll Officers and assign shifts.



Ballot box creation interface

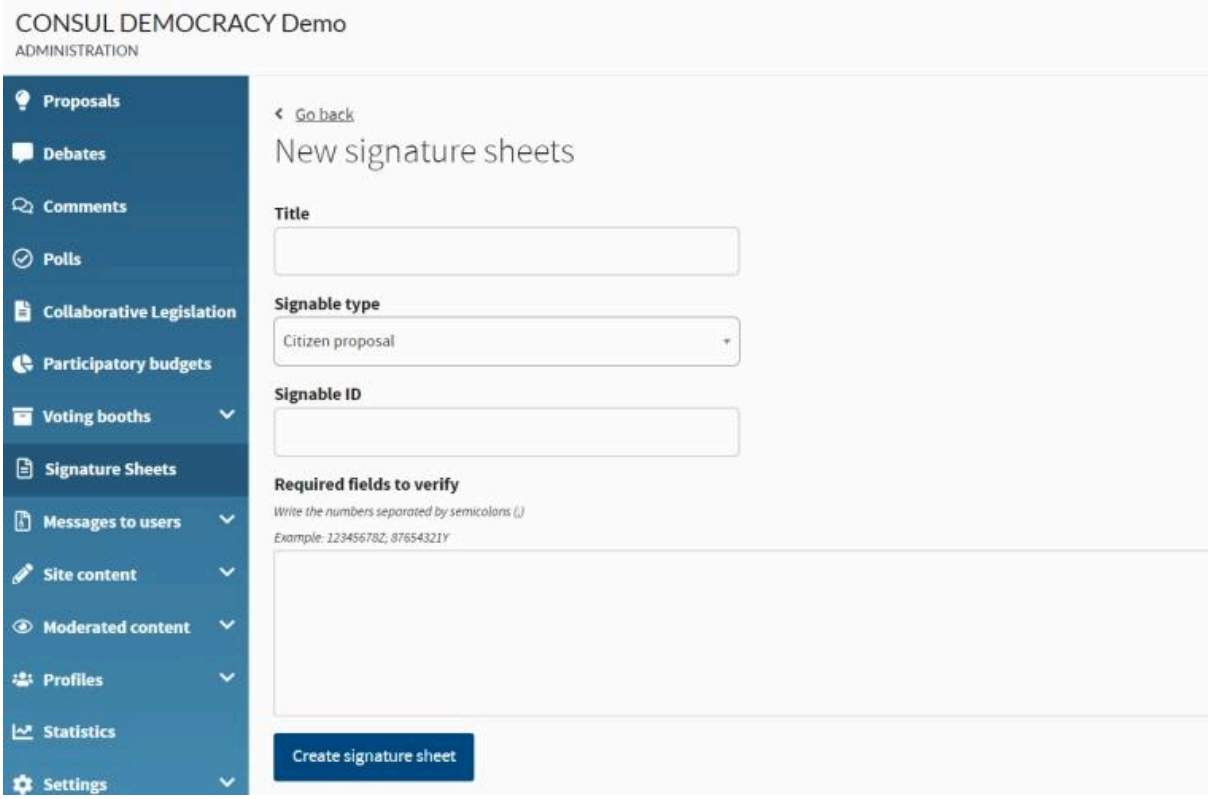
Poll officers. List of users with the role of "Poll officers". These users are allowed to access the polling officer interface (which we will explain later) only when the voting is active. The Poll officers verify that people vote correctly, that no one votes twice, and verify the results of the voting. To assign the role of "Poll officer" to a user, search for the user in the search box on top of the page, and click the button next to their user name.

Booth location. Voting booths can be given a name and a location, so they can be easily recognized. Click on "Add booth" to create a new voting booth and define name and location.

Booths assignment. Here Voting booths can be assigned to specific Votings. The button "Manage assignments" allows you to select which voting booths will be used for each vote, by clicking on "assign booth" next to the desired booth.

Manage shifts. By clicking on "Manage shifts" for each voting booth, you can search for a user with Poll Officer status and assign them with a shift by clicking on the 'Edit shifts' button. Shifts can either be "Collect Votes" for days when voting is open or "Recounting" for days after voting. These two tasks allow different options for the officer in the "Poll Officer" interface during the voting process. Multiple shifts can be assigned to each Poll Officer.

3.1.2 Signature sheets



Signature sheet creation interface

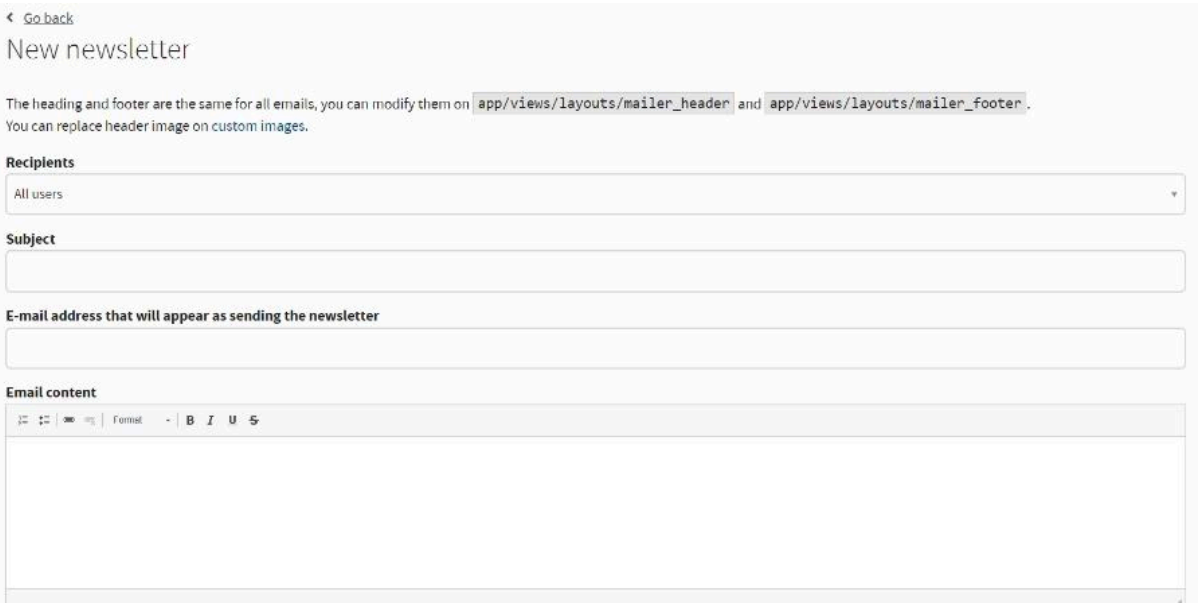
If you allow support proposals or projects of participatory budgets through traditional signature sheets, that part of the process can be managed here. This section allows you to

upload the signatures so that it can be verified if the citizens can support the proposal/project and not have duplicate or unverified signatures.

To upload new signatures, click on "New signature sheet". Select if it is a proposal or a participatory budget project, write its ID (it is shown on the proposal or project's web page) and then add the citizen document numbers (passport or ID) separated by commas. By clicking on "Create signature sheet", the system will verify each signature and add the supports to the proposal or project. Please note that the verification system may be slow. All the signature sheets will be displayed in this section, with a link for each one that shows all the information.

3.1.3 Messages to Users

This section manages communication with users. The following possibilities are available:



Newsletter creation interface

Newsletters. Informative newsletters are emails sent to users. For each one, you can define the content of the email and which users will receive it. Click on "New newsletter" to define the subject, the sender email address, the text and the recipients (the group of users that will receive the e-mail). Once created, you can check its "Preview" and then send it with the "Send" button on the preview page.
Notifications. Notifications are messages that appear in the users' notification panel (the bell on the top right of the screen). You can select the recipients, define the title

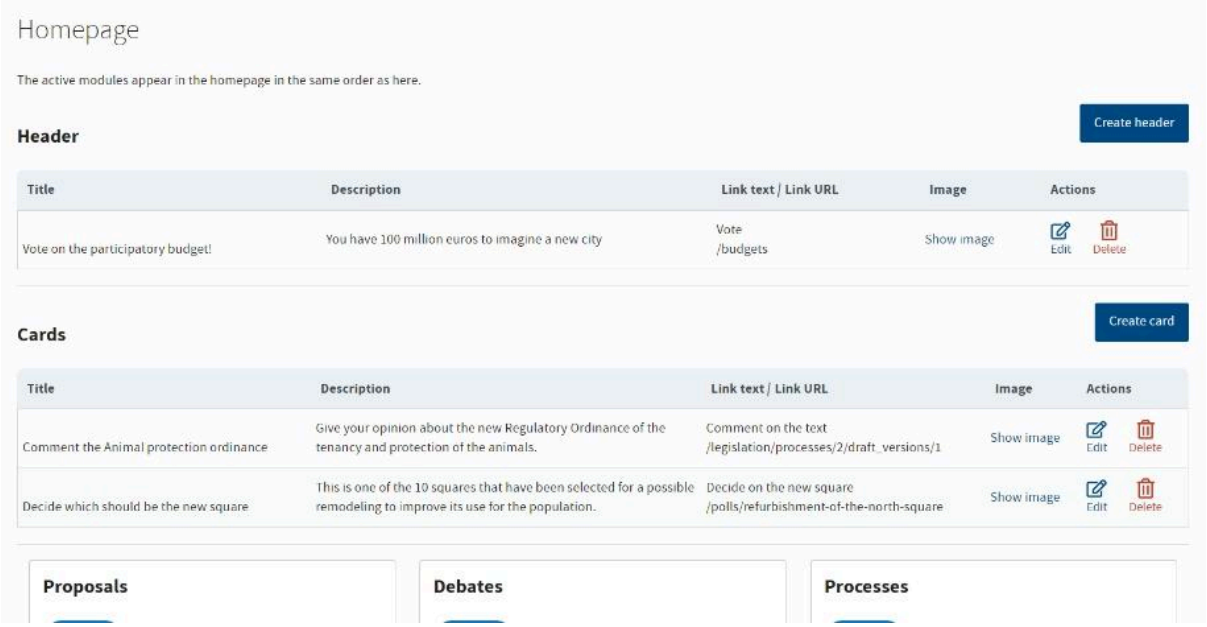
and the text (as in the previous section) and also include a link. When the user clicks on the notification, that link will open.

System emails. In this section, you can find all the emails that are sent to users. For example, the emails that are sent to the user when registering, when verifying the account, etc. A special type of email from the system is the "Proposal Notification Summary". Proposal authors can write proposal notifications that appear on the proposal web page. Those notifications are sent to all users who support the proposal. Here you can preview the abstract, check the content to avoid spam, and send it.

Email download. This last option allows you to select a group of users and download their emails in a CSV file.

3.1.4 Site content

Homepage. In this section, you can configure what is shown on the initial screen of the platform when entering it.



homepage interface

The first menu allows you to define the "Header", by pressing the "Create header" button. This is the first section that appears just below the menu bar, and that allows us to give the greatest visibility to what seems most relevant to us. The section can be configured with a title that will appear large, a small description, an additional image, and a link that will be accessed by clicking the button that will appear on the section.

The next element to configure is the "Cards". These cards are rectangles smaller than the header section, and located below, which will allow us to highlight other relevant processes or links, although less than the one in the header. We can create all the cards we want with the "Create card" button, which will be placed one after the other. Cards are defined in the same way with title, description, image and link.

Next, we can activate four different elements, with the “Activate” buttons: Proposals, Debates, Processes and Recommendations. Proposals will show us the proposals that appear at the head of the "Proposals" section of the platform. Debates consists of the same thing, but refer to the general section of the "Debates" platform. Processes will activate a format similar to the previous Cards, but with the content of the latest processes that are active in the "Processes" or "Collaborative Legislation" section of the platform. In these three cases, we can select the number of elements that will be displayed for each one. The last option "Recommendations" activates a module at the bottom of the page, which will only be seen when you are logged in as a user on the platform.

When this is the case, it will show three discussions and three proposals recommended to our user, based on the discussions and proposals we have supported before. This automatic recommendation system selects tags from discussions and proposals we've supported and looks for other items with the same tags. If we enter with our user in the Debates or Proposals section at the top of the debates or proposals we find different filters to order the elements: "Most active / Best valued / New / Recommended". This last filter is the same one used to display recommended items on the home page.

> **Custom pages**. In this subsection, you can create information pages for users. Click the "Create New Page" button to define it. The page includes a title, subtitle, and text. You can define the language of the page and also the "slug" (the URL that will be displayed in the web browser). The status can be chosen between "draft" (while we're creating it) and "published" (visible to users). And two additional options can be selected: display it in the "help" section of the platform and include a "print" button to make it easier for users to print.

In the list of pages, in addition to the link "See page" we will find another link "See cards". This allows you to add cards to highlight content at the bottom of the page, which will work the same as the cards explained in the previous Homepage section.

Manage banners. This section allows you to configure banners that will be displayed at the top of the Debates and Proposals sections of the platform. They will allow us to highlight other processes or links that we find especially relevant.

By clicking on the "Create a banner" button we can fill in the following information:

-Style. That will define the style of fonts and colours of the banner.

-Image. The background image of the banner.

-Start and end of publication. The banner will be displayed automatically during the selected period.

-Title and description. The text to be displayed on the banner.

-Link. The URL to which we will be directed when clicking on the banner.

The banners remain saved and can be reused later.

Customize texts. This section shows all the texts that are on the platform, both those seen by users and administrators or other roles. From the names of the menus, through the text of the buttons, to the informative texts.

Changing any text, and pressing the "Save changes" button we will see how it changes on the platform.

The texts are grouped in different tabs, to make it easier to find them. In particular, it is advisable to review the first tab titled "Basic customization". There we will find some text strings that we will most likely want to configure specifically for our platform.

Documents. This section allows us to upload files to the platform, which we can then link from any page or content that we create. To upload a file you only have to click on the "Upload document" button, select the file you want and click on "Upload document". In the list of documents the uploaded files will be shown, including the URL, which if we click on it will take us to the URL in the file browser. This will be the address that we can use in any section of the web.

3.1.5 Moderate content

The moderation of the platform is done from the moderation interface, which we will explain later. This section, however, allows you to track it. It shows all the content moderated by moderators or administrators of the platform, as well as the activity of the moderators to verify that it is being done correctly. All moderation activities are logged and can be verified by other administrators.

The subsections 'hidden proposals', 'hidden discussions', 'hidden comments' and 'hidden proposal notifications' work the same way. Each displays the list of moderated content, with "redisplay" and "confirm moderation" buttons next to each content.

> The first undo the moderation action, making the content public again.
> The second confirms that the moderation was successful and moves the content to the "confirmed" tab. The commit action doesn't change anything publicly, it's only useful for internal moderation review purposes.

The 'blocked users' subsection shows all users who have been blocked on the platform. In the same way, the blocking action can be undone or confirmed.

The subsection 'activity of the moderators' contains a record of each moderation action carried out and the date and the user who carried it out, to ensure that it was carried out correctly.

3.1.6 Configuration

In this section, you can define the general configuration options of the platform which, in general, do not affect the configuration of each specific participation process. These other configurations, as we have seen, are adjusted from the sections of each of the processes.

> **Global setting**. This is the main configuration section. It is divided into the following tabs: Global Settings, Participation Processes, Features, Map Settings, Images and Documents, and Proposal Progress Panel.

Global Configuration Tabs

- Global setting. Here we find the general parameters of the platform. Some parameters that are important to configure are the following: Number of supports necessary to approve a Proposal, Months to archive the Proposals, General URL of the web, Site name, Sender email name, Sender email address, and Minimum age to participate.

- Participation processes. This section allows you to activate and deactivate the different participation modules: Debates, Proposals, Voting, Participatory Budgets, Collaborative Legislation and SDGs.

- Features. By clicking on the “activate/deactivate” buttons, specific functionalities such as registration with Facebook or Twitter, the Communities spaces in the proposals, Recommendations, Geolocation, etc. will appear or disappear from the platform.

- Map settings: If geolocation is activated in the Functions section and latitude, longitude and zoom are correctly defined in the "Global configuration" section, we will see a map in this section, which will be the one displayed by default in participatory budget proposals and projects in order to place proposals.

- Images and documents. From this tab, you configure the type of images and documents that are allowed to be uploaded to the platform when creating proposals or other types of content, or the participation processes themselves. In particular, it is important to specify the type of files and the minimum size of the images.

- Proposal progress panel. As we have seen, when creating a proposal, its author can access a progress panel that allows him to follow how it evolves, as well as have resources and recommendations for his proposal to be successful. The resources and recommendations are defined in the following menu of the administration interface that we explained in point 2.1.3 but here you can define some of the parameters of this panel.

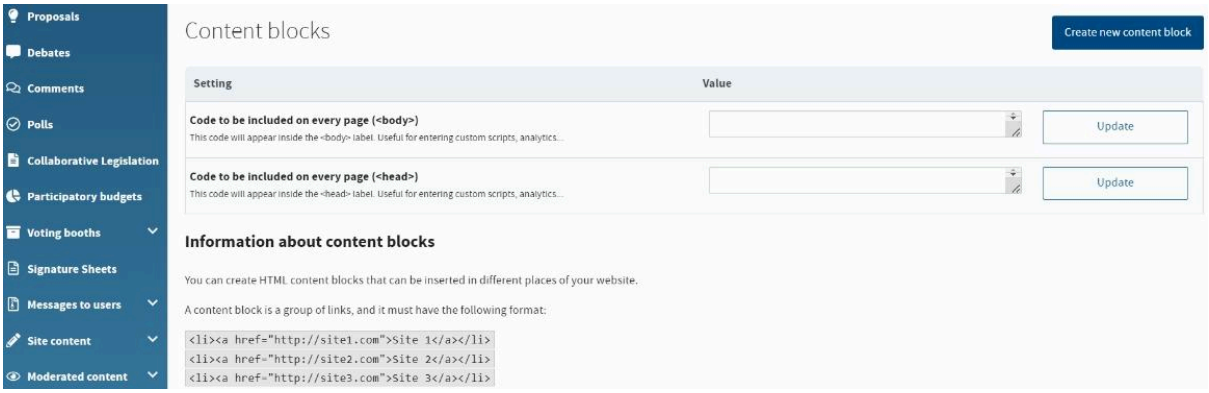
- SDG Configuration: This section will enable and disable the possibility of aligning the content of the different modules to the Goals and Goals of the 2030 Agenda.

> **Themes of proposals**. As we have seen, here when a user creates a proposal, some default labels are suggested for his proposal. These categories will also be displayed in the sidebar of the proposals section. In this section, you can define those labels by writing them in the upper field and clicking on the "Create theme" button.

Manage districts. When creating proposals from the general section of Proposals, they can be categorized as belonging to a specific geographical area. To define the geographical areas, click on the "Create an area" button and add the name and coordinates (the other fields are not necessary). The coordinates refer to the coordinates in the image that we defined in the installation of the platform (see the installation manual), they do not refer to geographic coordinates of latitude and longitude.

Customize images. Here we will include basic images of the platform, such as the logo that will appear on it in the top bar.

Customize blocks. This option allows us to define HTML blocks that will be embedded in the header or footer of the platform on all pages. This space helps us, for example, to integrate tracking tags for promotional campaigns. It is important to note that one of the options of these blocks is to change the main menus of the page by adding new menus, by adding "Main Navigation" blocks.



block section

3.1.7 Proposal Progress Dashboard

Everything related to this section is explained in detail in point 2.1.3, together with the rest of the configuration options of the proposals.

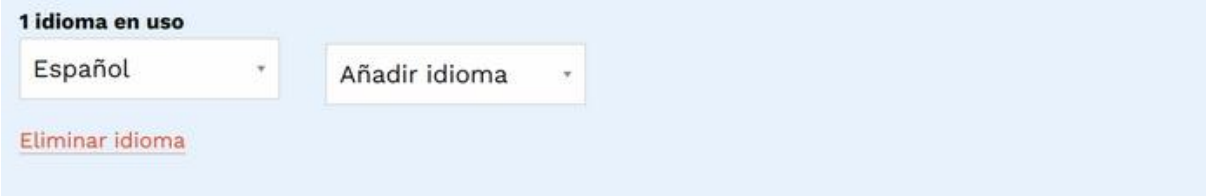
3.1.8 Multi-language

CONSUL DEMOCRACY is a multi-language platform. This corresponds to different functionalities:

The text of the platform itself (menus, buttons, informative text, etc.) is multi-language.

When administrators create participation processes, the content of the process is multi-language.
The content created by users (proposals, debates, comments) is multi-language.
There is an automatic translation system that allows the above contents to be translated into other languages when the administrators or users themselves have not done so.

In any of the forms where administrators or users can write some text, we will see a language bar at the top



block section

This bar shows a tab with the default language, and below a selector with the text "Add language". Pressing this selector will show the other available languages. When choosing a new language, a new tab will appear with the chosen language.

By clicking between tabs we can create the text in those languages. When we save the text, it will be saved in the languages that we have added. In case we want to delete any language, we can click on the link on the right side "Delete language".

Moderation

moderation interface

Moderators and administrators can access this interface. Here all the content of the platform indicated as 'inappropriate' by the users is listed. Moderators have the option to hide or confirm content and block users.

When a user flags a Proposal / Discussion / Comment with the "report as inappropriate" option, it will appear in this list. For each inappropriate item, the title, date, number of complaints (how many different users have marked the complaint option) and the text of the Proposal / Discussion / Comment will appear.

To the right of each element, there is a box that we can mark to select several from the list at the same time. Once we have selected one or more, we find three buttons at the bottom of the page to perform actions on them:

> Hide: It will hide those elements on the platform.
> Block authors: it will stop the authors of these elements from being able to access the web and will also hide all the Proposals / Debates / Comments of those users on the platform.
> Mark as Reviewed: Used when we believe these items should not be moderated, that their content is correct, and therefore should no longer appear on this inappropriate items list.

To facilitate the management, above we find a filter with the sections:

Pending: Proposals / Discussions / Comments that have not yet been hit "hide", "block" or "mark as reviewed", and therefore still need to be verified

All: Showing all Proposals / Discussions / Comments on the web, and not just those marked as inappropriate.

Marked as reviewed: those that a moderator has marked as seen and therefore appear correct.

It is advisable to check the "pending" section regularly.

block users

The search form allows us to find any user by entering their username or email, and block them once found. By blocking it, the user will not be able to access the web again, and all

their Proposals / Debates / Comments will be hidden and will no longer be visible on the web.

Evaluation

Evaluation Interface

Evaluators and administrators can access this interface to evaluate participatory budgeting projects. Through the administration interface, projects are assigned to different evaluators or groups of evaluators. Each evaluator can see in this interface only the projects assigned to him and complete the evaluation report for each project. With this, the evaluation can be done in a decentralized way, simplifying the work for each evaluator.

To be displayed here, projects must be assigned to the evaluator and also check the "show to evaluator" box in the admin interface. In this way, if this option is considered, administrators can ration the work that each evaluator does each time, and not show all the projects from the beginning.

When accessing the interface, the participatory budget in progress is displayed, and an "Evaluate" button to start the evaluation of the projects. The next window at the top shows the number of projects for each item, and then two tabs to organize the projects: "In evaluation" and "Evaluation completed"

By clicking on each project, you can see the public information about the project, the assigned administrators and evaluators, the current status of the evaluation report and a space for internal comments between evaluators and administrators. This comments section is not public, it is used only for internal debate and clarification among the people in charge of the project evaluation.

By clicking on the "Edit report" link, the evaluation report form is displayed. First, the feasibility of the project must be selected: undecided (default), viable or unfeasible.

By clicking on "viable", the following fields will be filled in:

Cost (mandatory). The cost of the project will be displayed during the final voting phase and will be used to calculate how many projects each user can vote on.
Cost report (optional). If this field is filled in, the content will be displayed on the project page. This field can be used to explain how the price was calculated or how the project can be executed.
Cost during the first year (optional, not public). This field is only used for internal purposes. It will not be published at any time.
Execution time (optional, not public). This field is only used for internal purposes. It will not be published at any time.

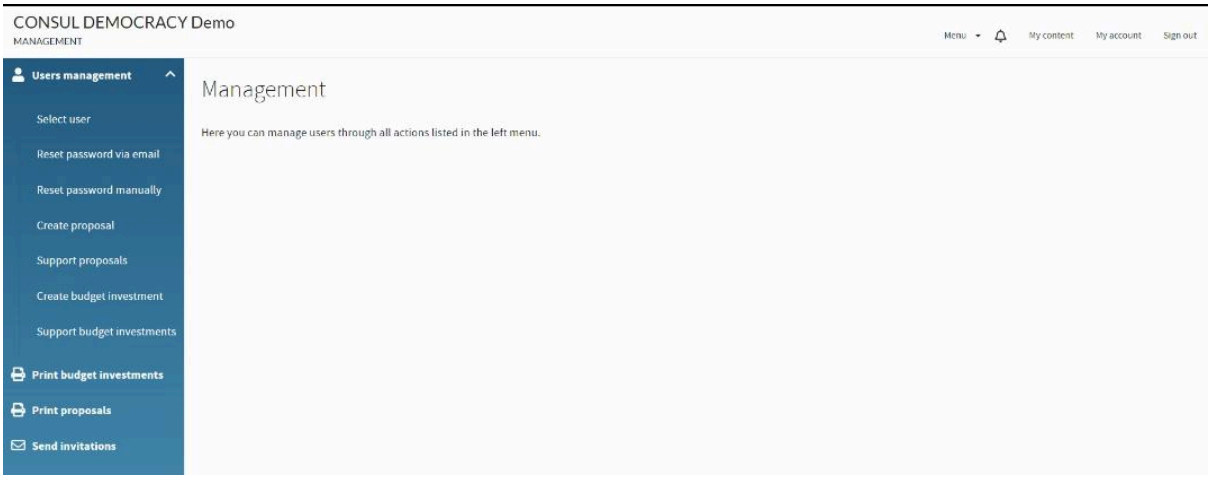
When clicking on the "unfeasible" option, the only field required is the following:

Unfeasibility report (mandatory). Here is why the project has been declared unfeasible.

When the report is marked as "Report Complete" and changes are saved, the project will appear publicly in the "Unfeasible projects" list and the unfeasible explanation will be posted on the project page. An email will also be sent to the author of the project, with the explanation of the unfeasibility. Keep in mind that this email is sent and the explanation is published on the platform at the same time that the evaluation is established as finished and the changes are saved. By clicking on the "evaluation finished" box and the "save changes" button, the report will be closed. The evaluator will not be able to make any further changes (except if an administrator unchecks that box again).

If you click on the "save changes" button without checking the "evaluation finished" box, the information on the fields will be saved, to continue working on them at another time.

Management



Management Interface

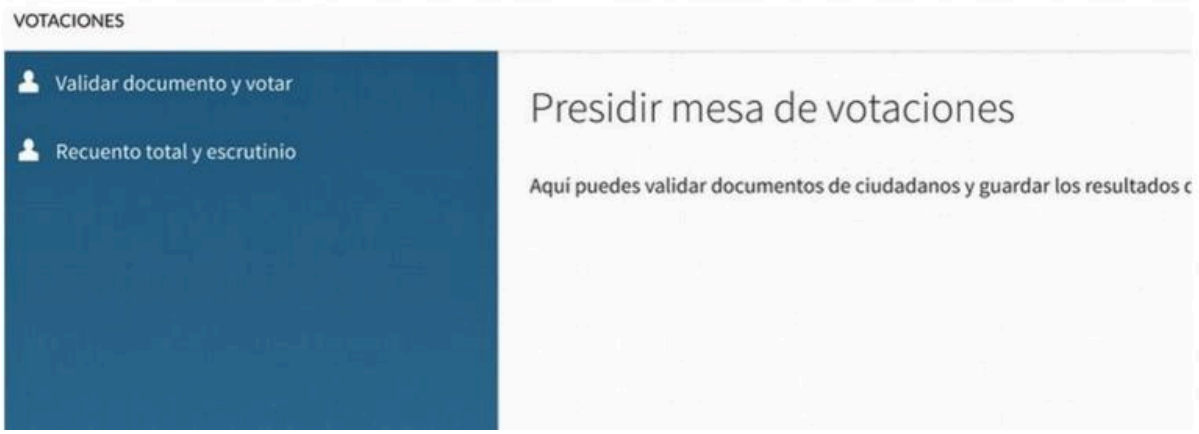
Administrators and Managers can access this interface. Here managers can create and verify user accounts and perform tasks for users to create or support proposals, etc. The most common destination for this functionality is public workers in the institution's citizen service offices, to help citizens interact with the participation process.

The following options are available:

Users. Managers can create or verify user accounts through this option. All other options require first access through this subsection, to select the user whose actions will be performed.
Edit user account. This option allows you to reset a user's password in two ways: by email (this is the most common way, the user will receive an email with a password reset link), manually (in case the user does not have access to your email, a random password can be generated or written by the user himself through this interface)
Create proposal. Here is the form to create a proposal. When the proposal is created, it will appear on the platform as created by the user.
Support proposals. A text field is displayed at the top of the page to search for proposals. Once the proposal is found, if you click on the "Support" button, the proposal will register support from the user. The final vote on proposals or participatory budgets, being a much more delicate process, is not managed from this interface. To do this, from the administration interface, you have the option of 'Voting boxes'.

Create spending projects. The form for creating projects for the current participatory budgeting process is shown here if the 'project submission' phase is open. When the project is created it appears on the platform as created by the user.
Support spending projects. At the top of the page, a text field is displayed to search for projects for current participatory budgets, if the 'support phase' of projects is open. Once the project is found, if you click on the "Support" button, the project will register support from the user.
Print proposals. This section makes it easy to print a part of the proposal list.
Print expense projects. This section makes it easy to print part of the list of participatory budget projects.
Send invitations. The field can be filled with a list of emails separated by commas. By clicking on the "send invitations" button, an email will be sent to each address inviting the person to register on the platform.

Poll officer



The table presidents can access this interface during a general vote or the final vote of the participatory budgets. This interface is used by those responsible for the face-to-face voting stations, to verify that the person who wants to vote can do so, and if they do, to confirm the vote so that they cannot vote a second time at another table or through the digital platform.

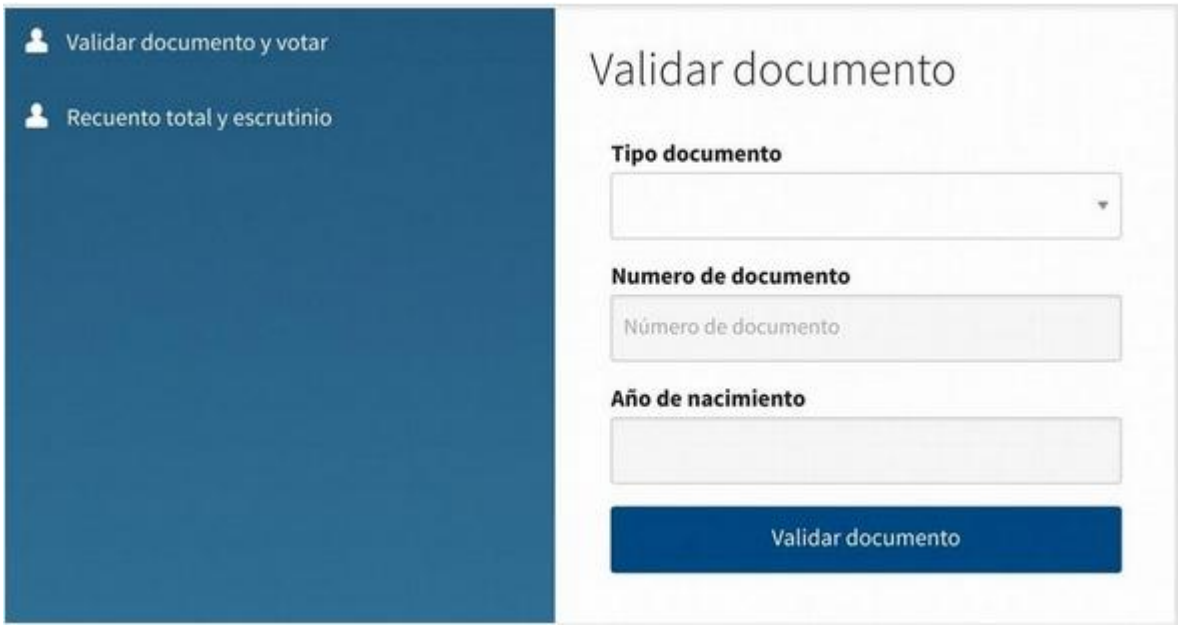
It also allows us to enter the results of the vote once it has finished.

To our left, we find a menu with two options (we will explain each one in a separate section)

Validate document and vote
Total count and scrutiny

3.5.1. Validate document and vote

In this section, we introduce the voter's identity document and manage their vote.



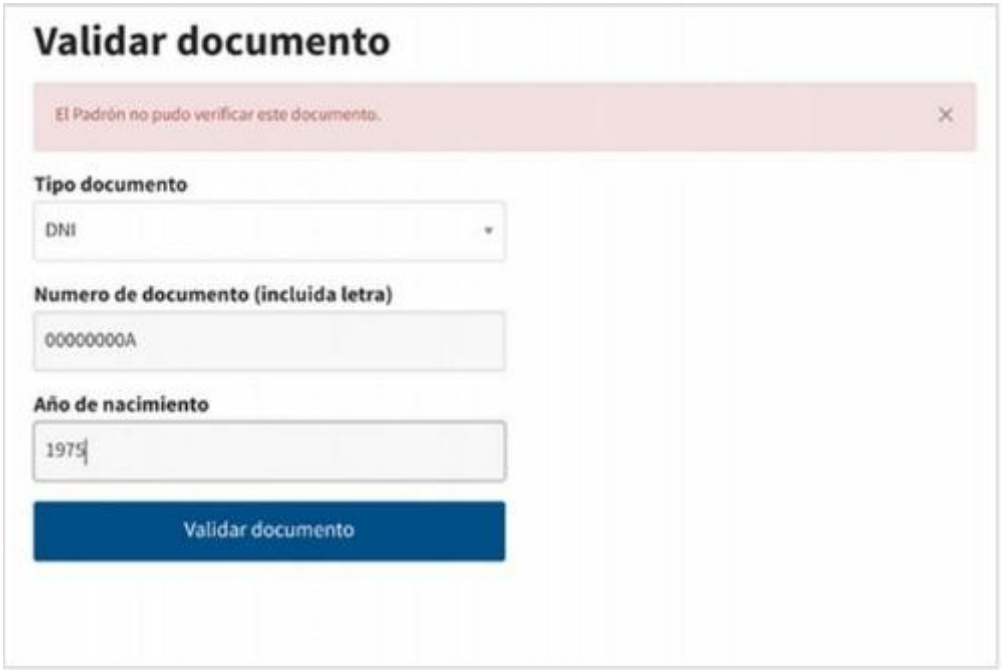
VOTER IDENTITY:

In the first place, we will find the form to enter the voter's data. The type of document, the document number and the year of birth: Different types of documents can be selected:

Select the type of document and enter the requested data: and click on the "Validate document" button.

If the person is not in the user verification registry with that document, or is under the age configured as the minimum voting age, or the information is not correct, it will show us the message "This document could not be verified".

If everything is correct, it will take us to the following screen:



It will show us the list of active votes in our polling station, indicating to us with the message "You can vote" in case the voter can vote, or a message "You have already participated in this vote" in case the voter has already voted on the internet or at another polling station.

In the event that they can vote, the voter will be told that they will proceed to introduce the envelope with the voting papers into the ballot box.

Once the corresponding envelope has been inserted, the President of the table will mark the "Confirm vote" button.

The button for said vote will change to the message “Vote entered correctly”.

If the voter tried to vote again at this table (or in the event that they had voted online) the following message would appear, preventing the vote:

Once this process is finished, click again on the side menu "Validate document and vote" to go to the next voter.

3.5.2 Total count and scrutiny

This last section will only be used on the day of the final scrutiny of the vote.

By clicking on the menu "Final count and scrutiny" on the left we will see the following screen:

In the central part, we will see the completed voting in which we can carry out the final count and scrutiny. We will click on the "Add results" button of the vote that we choose, which will take us to the following screen:

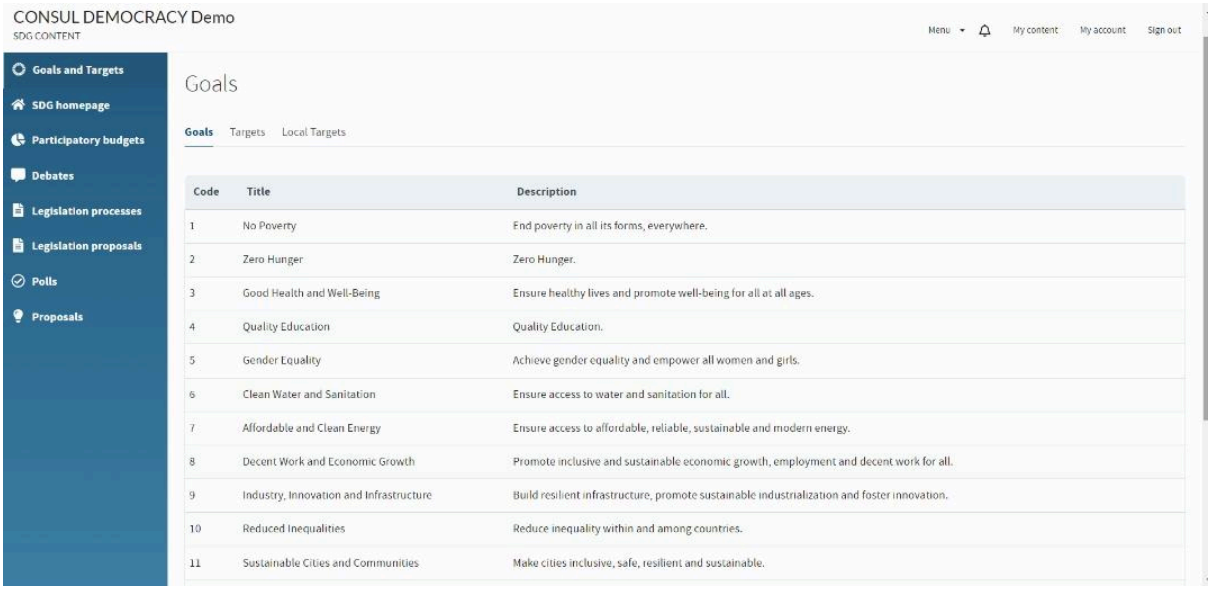
In the first drop-down, we will choose the ballot box corresponding to the scrutiny that we have carried out, and then we will fill in the requested fields.

Clicking on the "Save" button will reload the blank page, but at the bottom we will see a link with the results entered:

If we click on the link "See results" we can make sure that we have entered them correctly:

If we want to re-enter the result, because we made a mistake when entering it, we just have to repeat the process. The new result will overwrite the previous one.

SDG content



ODS content interface

The objective of this recently incorporated interface (2021) is to incorporate the data structure of the Sustainable Development Goals (SDGs) and goals of the 2030 Agenda into the CONSUL DEMOCRACY information architecture.

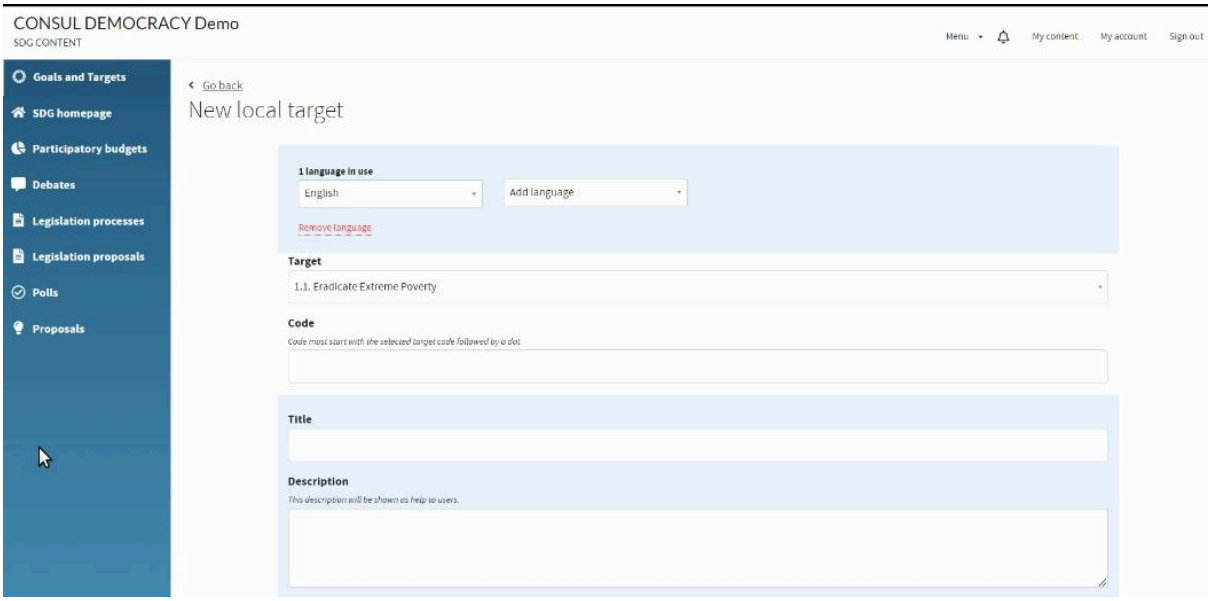
According to all the methodology developed by the United Nations that accompanies the SDGs, it is essential that the data structure is related to the goals. This is because the relationship with the SDG Indicators and with any other data is based on the corresponding goal. This data architecture is internationally used and is the one that allows a greater degree of interoperability between them. To apply this information architecture in CONSUL DEMOCRACY, a module has been developed that allows adding to all the data models of the system their relationship with the corresponding TARGETS and SDGs.

In the menu on the left, we find these sections:

3.6.1 Objectives and Goals:

In this section, we will find three tabs: "Objectives" where the 17 SDGs of the 2030 Agenda are listed; "Goals", where the different specific goals of each objective are listed, and "Local goals" where the local goals by the institution are listed and added.

To create a local goal, click on the "Create local goal" button that appears in the upper right part of the "located goals" tab, which will show us the following screen



The form to add a new Located Goal must be filled out taking into account the following indications.

> All Localized Goals must be related to a Global Goal.
> The Localized Target code must follow the related target code followed by “.” and the code you want below. Example: 1.1.3-MAD 2.
> It must have a title.
> It must have a description. The Localized Goals can be internationalized, so that it will be possible to register the Localized Goals in different languages

3.6.2 SDG home page

This interface allows you to customize the public page where all the information on the SDGs in the portal is collected and that allows navigation through all the content related to the SDGs.

To customize this page, according to the needs of each CONSUL DEMOCRACY installation, the following sections are available:

SDG header
phases

In the ODS header you can create a custom header with the following fields:

Label (optional)
Qualification
Description
link text
link URL
Image

In addition, the header is internationalizable, one can be defined for each language that is defined.

In the Phases section, you can add cards similar to those available on the CONSUL DEMOCRACY main page, associated with 3 phases.

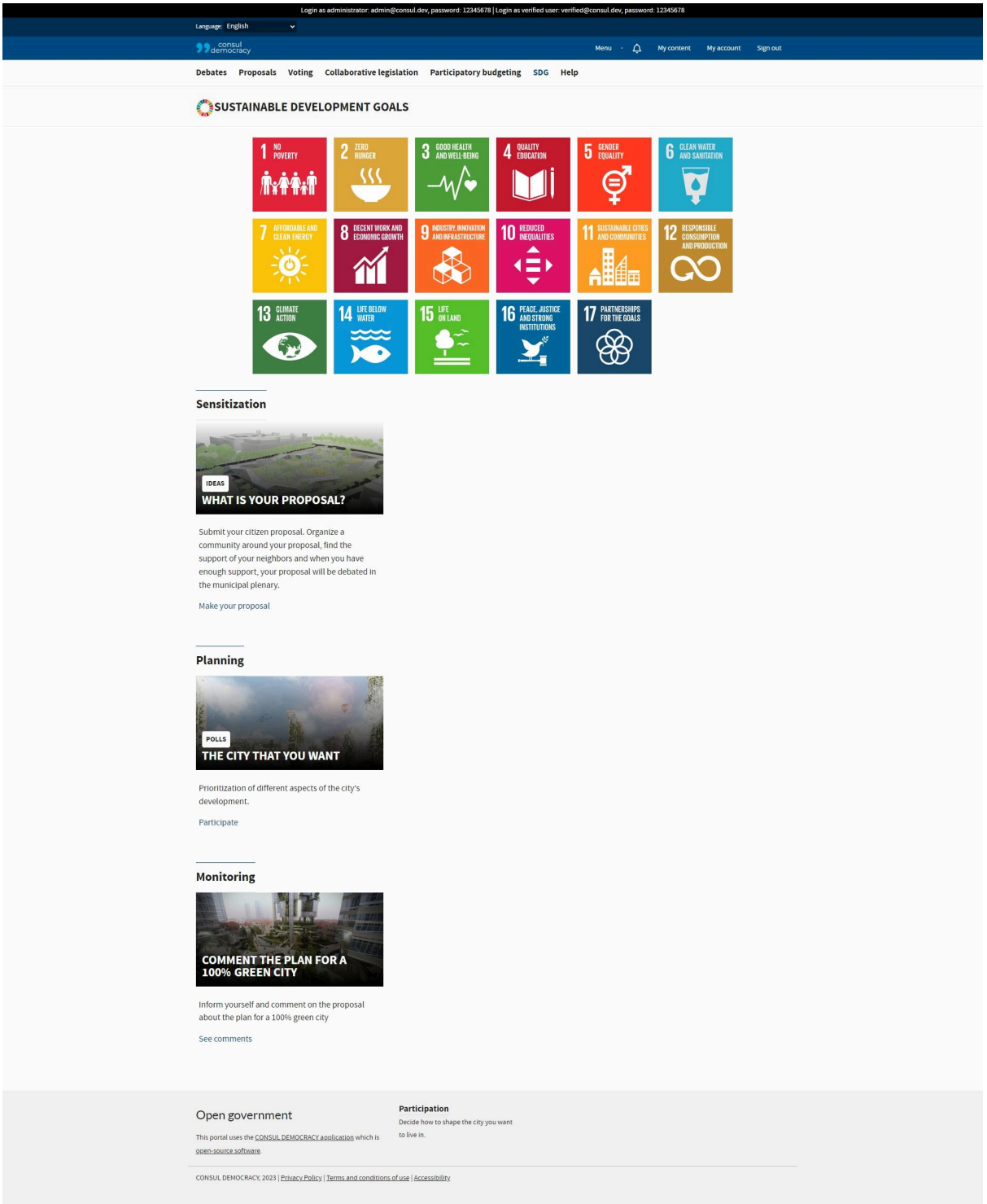
The name of the phases can be modified through the translations file.

The options that the cards allow you to add are the following:

- Label (optional)
- Qualification
- Description
- link text
- link URL
- Image

In addition, they are internationalizable, one can be defined for each language of the portal. They can also be adapted by defining columns to be able to customize the page according to the needs of the CONSUL DEMOCRACY installation.

The ODS page will look like this after the content has been added.



Users who access the SDG page (/sdg/goals) will be able to navigate by accessing a specific page for each of the SDGs. On each page, you will see the following information:

The name and icon of each ODS can be translated into each corresponding language of the CONSUL DEMOCRACY installation. There is an automatic

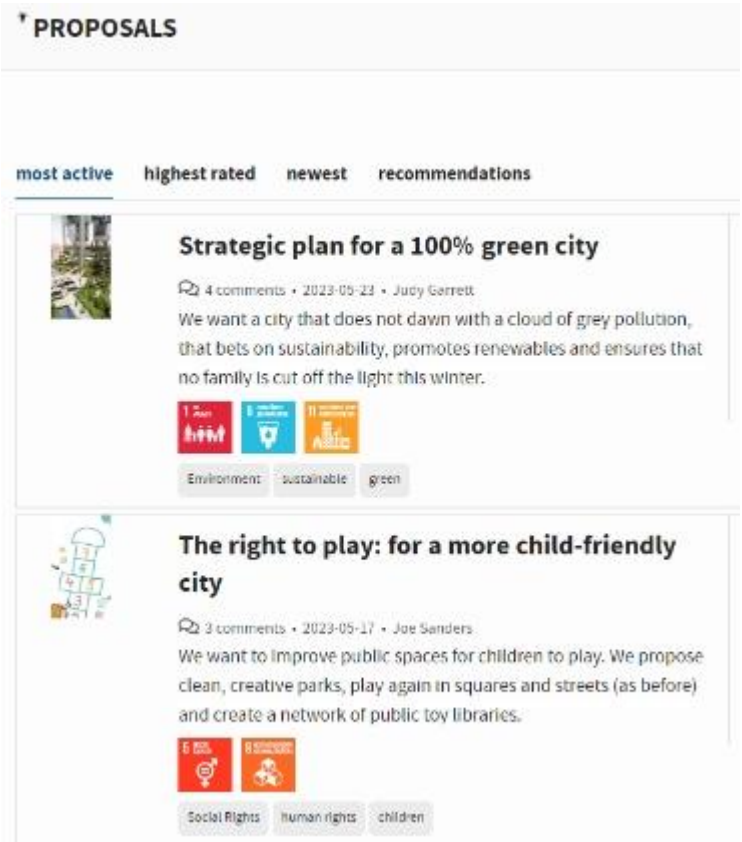
translation system that allows the above contents to be translated into other languages when the administrators or users themselves have not done so.
A descriptive text of each SDG. These texts have been adopted from the United Nations portal (www.un.org/sustainabledevelopment) or from other sources of the same body and all these texts are translatable.
More active proposals related to the corresponding SDG(s).
More active discussions related to the corresponding SDG(s).
Participatory processes related to the corresponding SDG(s).
The Global Goals corresponding to that SDG.
The Local Targets corresponding to that SDG.

Like other CONSUL DEMOCRACY views, this page can be adapted following the recommendations of the CONSUL DEMOCRACY documentation regarding customization of HTML View Customization.

If you want to modify the HTML of a page you can do it by copying the HTML from app/views and putting it in app/views/custom respecting the subdirectories you find there. For example, if you want to modify app/views/pages/conditions.html you must copy it and modify it in app/views/custom/pages/conditions.html.erb

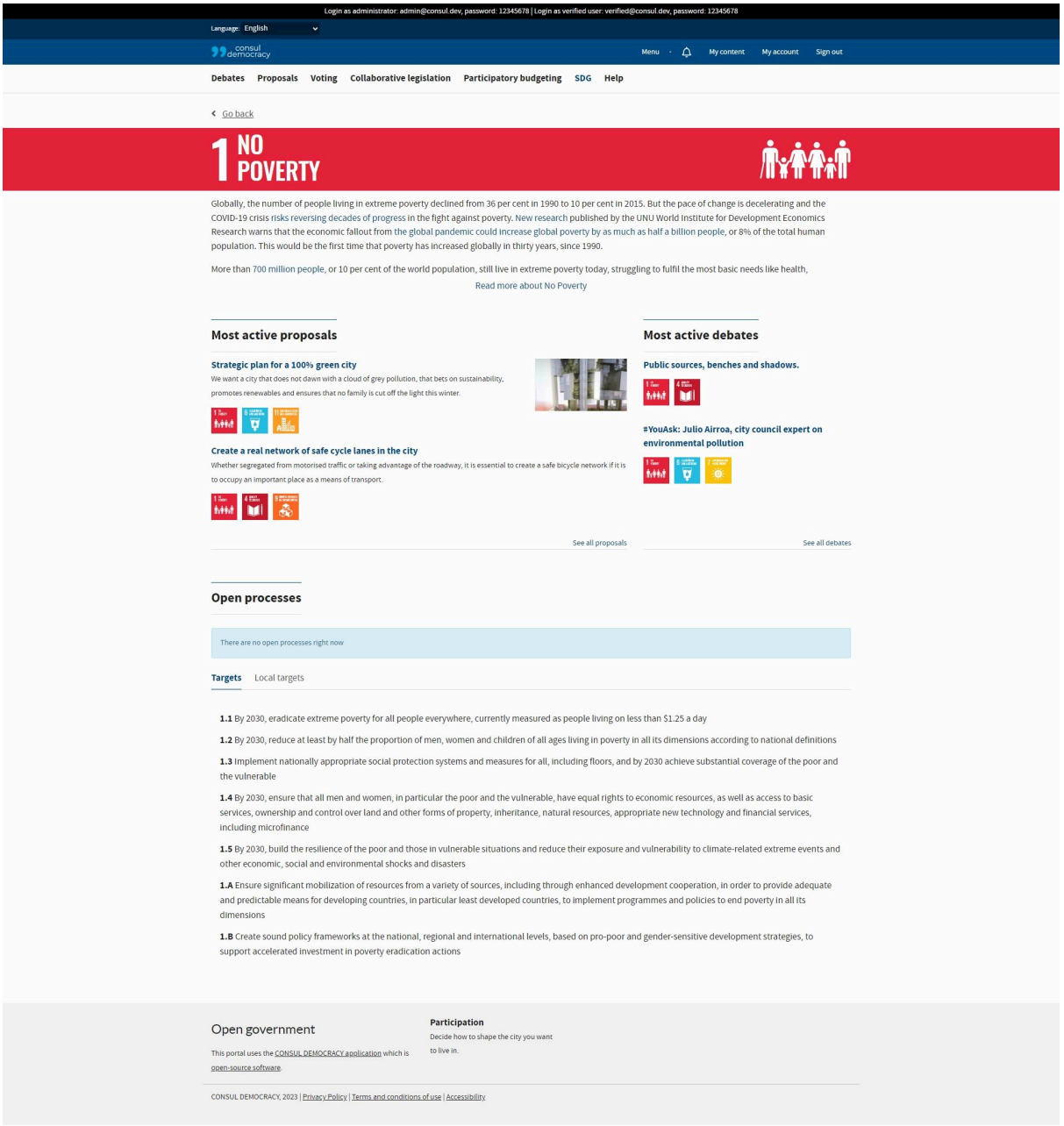
Each of the content blocks (most active Proposals, most active Debates and Processes) also allows direct access to the general view of each one with the filter corresponding to the SDG being consulted.

These blocks also show other SDGs in case the content is aligned with more than one SDG or Goal.



https://docs.consulproject.org/docs/spanish-documentation/customization/views_and_styles#vistas-html

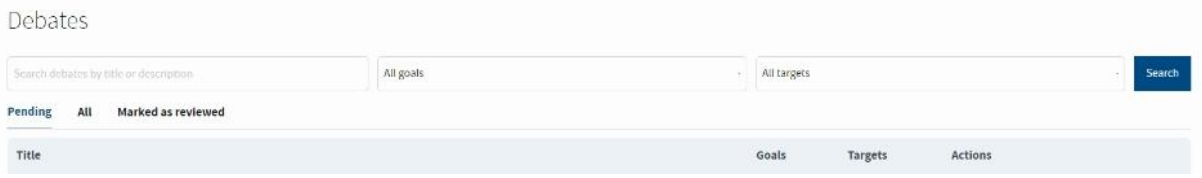
The view of the main page of an ODS will be the following:



3.6.3. discussions

From this management interface, the Administration profile or the profile with the ODS Management role will be able to moderate the content that has been created in this section, aligned with the ODS.

All open discussions on the portal are collected and managed.



All debates that have not been reviewed are classified as Pending. In the All tab there are all, those reviewed and not reviewed, and in the Marked as reviewed tab, there are those that have been reviewed.

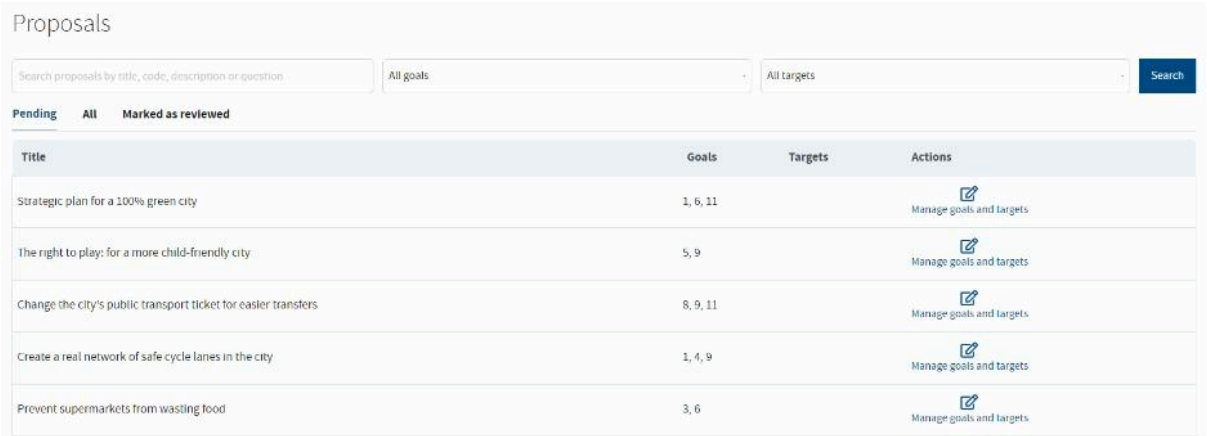
From the SDG Administration or Management profile, the SDGs or Goals aligned in each entry can be modified by clicking on the Actions icon and editing the SDGs and/or Goals assigned to each entry.

To facilitate the management, entries can be filtered by SDGs or Goals or search for specific content.

3.6.4. proposals

From this management interface, the Administration profile or the profile with the ODS Management role will be able to moderate the content that has been created in this functionality aligned with the ODS.

From this interface, it will be possible to manage all the proposals created through the CONSUL DEMOCRACY Proposals module.



All those that have not been reviewed are classified as Pending. On the All tab are all proposals (reviewed and unreviewed) and on the Marked as reviewed tab are those that have already been reviewed.

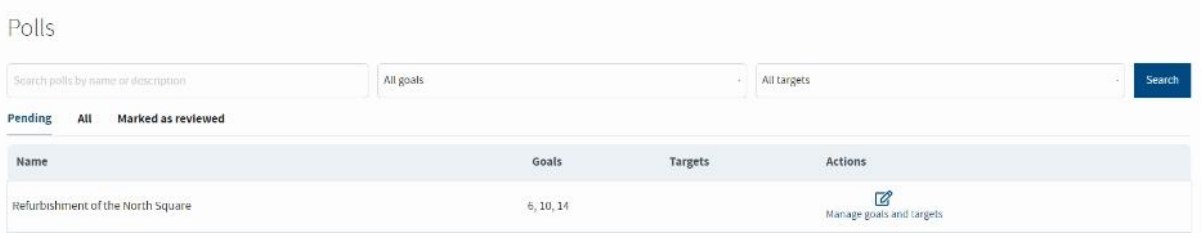
From the SDG Administration or Management profile, you can modify the SDGs or Goals aligned in each entry by clicking on the Actions icon and editing the SDGs and/or Goals assigned to each entry.

To facilitate the management, entries can be filtered by SDGs or Goals or search for specific content.

3.6.5. voting

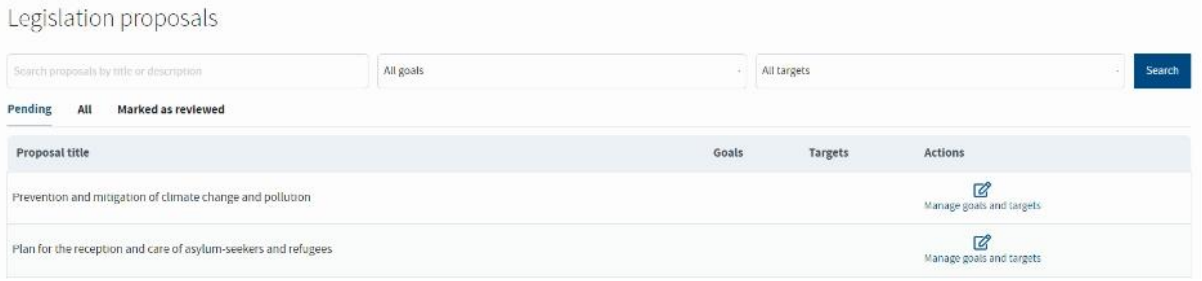
From this management interface, the Administration profile or profile with the ODS Management role will be able to moderate the content that has been created in this functionality aligned with the ODS.

This interface allows you to edit and manage the alignment of the Voting processes.



3.6.6. collaborative legislation

From this management interface, the Administration profile or the person with the SDG Management role will be able to moderate the content that has been created in this section, aligned with the SDGs.



3.6.7. Proposals within Collaborative Legislation

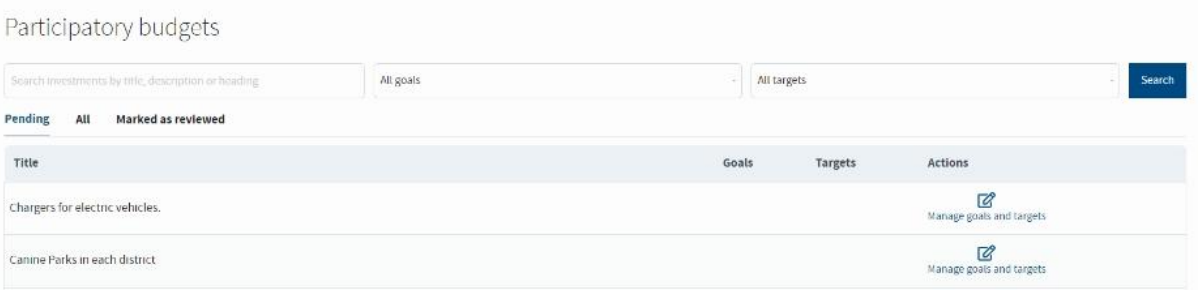
From this management interface, the administrator or the person with the SDG Management role will be able to moderate the content that has been created in this section, aligned with the SDGs.

Collaborative legislation includes complex processes that consist of different phases. One of them is the Proposal Phase, a period in which specific proposals to contribute to the Legislative process are compiled. In this interface, it is possible to manage these proposals to modify their alignment with the SDGs.

3.6.8. Participatory Budgets

From this management interface, the administrator or the person with the SDG Management role will be able to moderate the content that has been created in this section, aligned with the SDGs.

In this case, the spending proposals associated with the Participatory Budgets functionality can be managed from this interface.



All those that have not been reviewed are classified as Pending. In the All tab there are all the proposals (reviewed and not reviewed) and in the Marked as reviewed tab, there are those that have already been reviewed.