DIY Site

## Introduction

The DIY site is intended to be a repository of small-scale construction projects, designed so that anyone can carry them out by following the instructions posted on the repository in a genuine Do-It-Yourself style.

In this first phase of implementation, people of any age and skill level can explore the projects named "Things" that are publicly presented in this repository and use the project instructions to replicate them. The contents of the projects (things) are entered into the repository by the creators who require private access with certain privileges, assigned by an administrator, in order to access the repository management system.

Public access to this website can be made completely anonymous, allowing the users to browse the entire published repository, and encouraging them to register on a permanent basis, which will also allow users to mark which projects are their favourites.

From a technical standpoint, this site is implemented in Python, JavaScript, HTML and CSS languages, on the Django framework and using MVC design. It includes user authentication, email validation and full CRUD functionality to manage the content of the repository.

In future phases of implementation, the options to publish content by users from public access and the ability to sell their creations to other users will be implemented. Advanced functions will also be added, such as the search and filtering of projects based on content, or the use of shopping carts, user profiles, payment gateways, etc.

Access the [website deployed on Heroku](https://dys-1c0dad79f0a0.herokuapp.com/)

Table of Contents

[Introduction 1](#__RefHeading___Toc5131_1298562281)

[User experience (UX) 3](#__RefHeading___Toc5133_1298562281)

[Functional specifications 3](#__RefHeading___Toc3963_4273941663)

[The User 3](#__RefHeading___Toc3965_4273941663)

[Goals 3](#__RefHeading___Toc3967_4273941663)

[Customer requirements for users 3](#__RefHeading___Toc3969_4273941663)

[Other requirements 4](#__RefHeading___Toc4287_4273941663)

[Development plan 4](#__RefHeading___Toc4289_4273941663)

[Web Site Sketches 12](#__RefHeading___Toc4351_4273941663)

[Database 14](#__RefHeading___Toc5155_1298562281)

[Features 15](#__RefHeading___Toc5165_1298562281)

[Users Home page 15](#__RefHeading___Toc5167_1298562281)

[Footer 15](#__RefHeading___Toc3677_4205823145)

[Users Navigation Bar 15](#__RefHeading___Toc5169_1298562281)

[Sign Up 16](#__RefHeading___Toc3679_4205823145)

[Sign In 16](#__RefHeading___Toc3681_4205823145)

[Sign Out 16](#__RefHeading___Toc3683_4205823145)

[Email 16](#__RefHeading___Toc3685_4205823145)

[DIY Project details 17](#__RefHeading___Toc5177_1298562281)

[Administration pages 18](#__RefHeading___Toc3687_4205823145)

[Future Enhancements 24](#__RefHeading___Toc5203_1298562281)

[Testing 25](#__RefHeading___Toc5205_1298562281)

[Testing Strategy 25](#__RefHeading___Toc5207_1298562281)

[Validating the Source 25](#__RefHeading___Toc5538_970266643)

[Django Python code 25](#__RefHeading___Toc5540_970266643)

[CSS code 25](#__RefHeading___Toc5542_970266643)

[HTML code 26](#__RefHeading___Toc5544_970266643)

[Testing the application 26](#__RefHeading___Toc5546_970266643)

[The development environment 29](#__RefHeading___Toc5217_1298562281)

[Resources 29](#__RefHeading___Toc5219_1298562281)

[Deployment 30](#__RefHeading___Toc5223_1298562281)

[How to deploy the project on Heroku 30](#__RefHeading___Toc5225_1298562281)

[Create a clone of this repository 30](#__RefHeading___Toc5229_1298562281)

[Acknowledgements 31](#__RefHeading___Toc5231_1298562281)

## User experience (UX)

DIY Site aims to be a friendly public site for users to access DIY style projects that they can replicate. The presentation of the content is dynamic and attractive, and the information about the projects is clear and easy to read.

Registered project creators will also be able to add and edit their own creations. This administrative interface is geared towards a more functional purpose, so that project content can be easily edited and expanded.

### Functional specifications

#### The User

There are four types of users segregated by their role and purpose.

* **Anonymous:** A non registered user
* **Regular:** A registered user
* **Creator:** A user with privileges to add/edit/publish content
* **Administrator:** User that can validate users and perform all actions.

#### Goals

* For all users, the user interface should be minimalistic and intuitive
* For anonymous and regular users the user interface should be friendly, dynamic, clear and fun.
* For Creators and administrators, The user interface should be clear, flexible and functional.

#### Customer requirements for users

The following are the customer requirements that are part of the user stories list for the initial delivery iteration backlog.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epic | Story | Requirements for users | Anonymous | Regular | Creator | Admin |
| 38 | 12 | Register an account using e-mail and password | x |  |  |  |
| 38 | 13 | Validate own e-mail |  | x | x | x |
| 38 | 14 | Login in main page with e-mail and password | x | x | x | x |
| 38 | 15 | Change e-mail |  | x | x | x |
| 38 | 16 | Delete own e-mail account |  |  | x | x |
| 37 | 17 | Register own account using user and password |  |  |  | x |
| 37 | 17 | Login in admin page with user and password |  |  | x | x |
| 37 | 17 | Change own user data (name, lastname) |  |  | x | x |
| 38 | 18 | Change own password |  | x | x | x |
| 37 | 17 | Delete own user-account |  |  |  | x |
| 38 | 35 | Sign out/log out |  | x | x | x |
| 37 | 17 | Administrate users and password |  |  |  | x |
| 41 | 19 | view list of published DIY projects | x | x | x | x |
| 41 | 20 | View details of a published DIY project | x | x | x | x |
| 41 | 21 | View published components of a DIY project | x | x | x | x |
| 41 | 20 | View instructions of a DIY project | x | x | x | x |
| 38 | 22 | Mark a DIY project as own favourite |  | x | x | x |
| 39 | 23/24 | Create a DIY project and its details |  |  | x | x |
| 39 | 26 | Create components (DIY subprojects) |  |  | x | x |
| 39 | 25 | Create instructions |  |  | x | x |
| 39 | 27 | Modify a DIY project and its details |  |  | x | x |
| 39 | 28 | Modify a component and its details |  |  | x | x |
| 39 | 29 | Modify the instructions of a DIY project |  |  | x | x |
| 39 | 30 | Delete a DIY project and its details |  |  | x | x |
| 39 | 31 | Delete a component |  |  | x | x |
| 39 | 32 | Delete instructions |  |  | x | x |
| 39 | 33 | Change state published/Draft a DIY project |  |  | x | x |
| 39 | 34 | Change state published/Draft a component |  |  | x | x |

#### Other requirements

* There should be a main landing page to display the content.
* There should be an administrator page for manage users and content.
* Web pages should be able to be used from desktops and mobile devices
* Content and user informations should be stored in Elephant PostgreSQL servers
* Multimedia content (Images, videos, data files, etc) should be stored in Cloudinary servers.
* The web site should be able to be deployed on Heroku servers from GitHub
* The project should be develop under Agile methodology.

#### Development plan

Since development must be managed under the agile methodology. `Epics`, and their corresponding `user stories` must be defined and, when needed, meet a consensus between the developers and the project owner through agreed acceptance criteria. Also, from a more technical point of view, development tasks and their corresponding `story points` are defined in order to better evaluate the development effort. The following information lists all these issues. Please follow the links to access the full information stored on github.

##### #37 EPIC: Setup initial development environment

As a \*\*developer\*\*, I can \*\*setup initial environment\*\*, so that \*\*The website can be developed\*\*

###### #36 USER STORY: Create Development Environment

As \*\*developer\*\*, I can \*\*setup the development environment\*\*, so that \*\*I can start developing the website\*\*

## Tasks

- [x] Task 1 – Install Visual Studio Code in local environment

- [x] Task 2 – Create website Github repository and start remote environment

- [x] Task 3 – Install Django and all required libraries for runtime and development of the website

- [x] Task 4 – Create Django project and application

- [x] Task 5 - Create database in ElepantSQL servers

- [x] Task 6 - Create media storage in Cloudinary servers

- [x] Task 7 – create env.py and add all security risk keys

- [x] Task 8 - Migrate Django project to use the new database and media storage

- [x] Task 9 - Setup Heroku service for the website

- [x] Task 10 – Setup automatic deployment from Github on Heroku servers

## Story Points: 8

###### #17 USER STORY: Administrate users

As \*\*Administrator\*\*, I can \*\*Create/modify/delete user accounts and their privileges\*\*, so that \*\*I can assign who signs in on the website and who can create or modify content\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in any website page

\*\*Then\*\* I can access the admin web page selecting the `Admin` option in the user menu

- Acceptance Criteria 2

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any user

- Acceptance Criteria 2

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any user group

- Acceptance Criteria 2

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any privilege of any user and any group

- Acceptance Criteria 3

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any privilege of any user and any group

## Tasks

- [x] Task 1 - Install and setup Django's allauth and its accounts and interfaces, extensions.

- [x] Task 2 - Create at least one administrator account

- [x] Task 3 - Customize allauth templates to the website base template

- [x] Task 4 - Create `Content Creators` user group with necessary privileges for content creators

## Story Points: 8

##### #39 EPIC: DIY Project creation and management

As a \*\*content creator\*\*, I can \*\*add/modify/delete DIY projects\*\*, so that \*\*the DIY Projects can be published and access by users in the website\*\*

###### #40 USER STORY: Register users as Content Creators

As \*\*Administrator\*\*, I can \*\*Give privileges to users to became Content Creators\*\*, so that \*\*This content creators can handle DIY projects\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am an administrator

\*\*When\*\* a user is designed as Content Creator

\*\*Then\*\* I can assign this user to the Content Creator user group

- Acceptance Criteria 2

\*\*Given\*\* that I am an administrator

\*\*When\*\* a user is no longer designed as Content Creator

\*\*Then\*\* I can revoke their Content Creator user privileges

- Acceptance Criteria 3

\*\*Given\*\* that I am a Content Creator

\*\*When\*\* I want to create/modify/delete a DIY Project

\*\*Then\*\* I have sufficient privileges to perform those tasks

## Tasks

- [x] Task 1 Create ` Content Creators` users group

- [x] Task 2 Assign sufficient privileges to the ` Content Creators` users group

- [x] Task 3 Assign designated users to the ` Content Creators` users group

## Story Points: 3

###### #23 USER STORY: Create a DIY project

As \*\*content creator\*\*, I can \*\*Create a DIY project\*\*, so that \*\*I can publish it and share it on the website\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am in any website page

\*\*Then\*\* I can access the admin web page selecting the `Admin` option in the user menu

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can create a new DIY Project

- Acceptance Criteria 3

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have created a new DIY Project

\*\*Then\*\* The Created date, Modified\_Date and Parent fields are automatically created for this DIY project

## Tasks

- [x] Task 1 - Allow the `Admin` option menu in the user menu in the base template for Staff users (from group `Content Creators`)

- [x] Task 2 - Create at least one user from the `Content Creator` user group and the `Staff` flag enabled

- [x] Task 3 - Create the DIY Project object model (`Thing`) that will be used to instantiate each `Thing` and each `Component`

- [x] Task 4 - Create the Admin view for the DIY Project (`ThingAdmin`)

## Story Points: 8

###### #24 USER STORY: Create DIY project details

As \*\*content creator\*\*, I can \*\*create the DIY project details\*\*, so that \*\*project can have a featured image and description\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have created a new DIY Project (`Thing`)

\*\*Then\*\* I can assign a Title, Author, Status, and Featured Image

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have created a new DIY Project (`Thing`)

\*\*Then\*\* I can edit a WSWG document to the description

- Acceptance Criteria 3

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have created a new DIY Project (`Thing`)

\*\*Then\*\* Slug field should be assign/updated automatically.

- Acceptance Criteria 4

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have created a new DIY Project (`Thing`)

\*\*Then\*\* I need to use a Save button to permanently store the changes in the database

## Tasks

- [x] Task 1 - Install Django Summernote extension for adding embedded WSWG editing functionality.

- [x] Task 2 - Modify the DIY Project Admin view (`ThingAdmin`) to accommodate all detail fields

- [x] Task 3 - Add the modifiable fields to the fieldsets in the DIY Project Admin view (`ThingAdmin`) to allow updating of these fields.

## Story Points: 8

###### #25 USER STORY: Create Instructions

As \*\*content creator\*\*, I can \*\*create a list of project instructions\*\*, so that \*\*the users can read and follow them\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can create a new `instructions` element that will be automatically assigned to the current DIY Project

## Tasks

- [x] Task 1 - Create the DIY Instructions class model (`Instructions`) that will be used to instantiate each `instructions` object

- [x] Task 2 - Modify the DIY Project object model 'Thing' to include the instructions model

- [x] Task 2 - Create the DIY Component Admin view (`InstructionsInLine`) to handle new instructions elements as a list.

- [x] Task 3 - Add inline admin view `InstructionsInLine` to the DIY Project Admin view (`ThingAdmin`) to allow new instructions elements to be added as a list.

## Story Points: 5

###### #26 USER STORY: Create component

As \*\*content creator\*\*, I can \*\*create a list of components\*\*, so that \*\*a user can view and gather the building blocks to fulfil the project\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can create a new component that will be automatically assigned to the current DIY Project

## Tasks

- [x] Task 1 - Modify the DIY Project object model 'Thing' to include a relation to itself as `components`

- [x] Task 1 - Create the DIY Component Admin view (`ComponentInLine`) to handle new components as a list.

- [x] Task 2 - Add inline admin view `ComponentInLine` to the DIY Component Admin view (`ComponentInLine`) to allow new components to be added as a list.

## Story Points: 5

###### #27 USER STORY: Modify a project details

As \*\*content creator\*\*, I can \*\*modify the DIY project details\*\*, so that \*\*the project can be corrected or upgraded\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (Thing)

\*\*Then\*\* I can change the Title, Author, Status, and Featured Image

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (Thing)

\*\*Then\*\* I can edit a WSWG document to the description

## Tasks

- Built in #24

## Story Points: 0

###### #28 USER STORY: Modify component

As \*\*content creator\*\*, I can \*\*modify the component details\*\*, so that \*\*the component can be corrected or upgraded\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can change the title of the Instructions element

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can change the content of Instructions with a WSWG built-in editor

- Acceptance Criteria

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I need to use a `Save` button to permanently store the changes in the database

## Tasks

- built in #24

- [x] Task 1 - Add Summernote edit functionality to the `description` field in inline DIY Component Admin view (`ComponentInLine') to allow for the WSWG edit features in the DIY Component list.

- [x] Task 4 - Add the modifiable fields to the fieldsets in the inline DIY Component Admin view (`ComponentInLine`) to allow updating of these fields.

## Story Points:3

###### #29 USER STORY: Modify instructions

As \*\*content creator\*\*, I can \*\*modify the project instructions\*\*, so that \*\*the project can be corrected or upgraded\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can change the title of the Instructions element

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can change the content of Instructions with a WSWG built-in editor

- Acceptance Criteria

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I need to use a `Save` button to permanently store the changes in the database

## Tasks

- [x] Task 1 - Add Summernote edit functionality to the `instructions` field in DIY Instructions Admin view (`InstructionsAdmin') to allow for the WSWG edit features.

- [x] Task 2 - Add the `title` and `instructions` fields to the fieldsets in the DIY Instructions Admin view (`InstructionsAdmin`) to allow updating of these fields.

- [x] Task 3 - Add Summernote edit functionality to the `instructions` field in inline DIY Instructions Admin view (`InstructionsInLine') to allow for the WSWG edit features in the DIY Project Details Instructions list.

- [x] Task 4 - Add the `title` and `instructions` fields to the fieldsets in the inline DIY Instructions Admin view (`InstructionsInLine`) to allow updating of these fields.

## Story Points: 5

###### #30 USER STORY: Delete a project

As \*\*content creator\*\*, I can \*\*delete a DIY project\*\*, so that \*\*the project does no longer exist on the website\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I select a DIY project from the DIY Projects list in the admin website page

\*\*Then\*\* I can delete all the selected projects

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project in admin website page

\*\*Then\*\* I can delete the current DIY Project

- Acceptance Criteria 3

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have requested to delete one or more DIY projects

\*\*Then\*\* I will have to explicitly confirm the deletion

## Tasks

Built in #23

## Story Points: 0

###### #31 USER STORY: Delete component

As \*\*content creator\*\*, I can \*\*delete a component\*\*, so that \*\*the component does no longer exist on the website\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can delete an existing DIY Component from the current DIY Project

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have requested to delete one or more DIY Components

\*\*Then\*\* I will have to explicitly confirm the deletion

## Tasks

Built in #26

## Story Points: 0

###### #32 USER STORY: Delete Instructions

As \*\*content creator\*\*, I can \*\*delete project instructions\*\*, so that \*\*those instructions do no longer exist on the website\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I am modifying a DIY Project (`Thing`)

\*\*Then\*\* I can delete an existing Instructions element from the current DIY Project

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have requested to delete one or more Instructions elements

\*\*Then\*\* I will have to explicitly confirm the deletion

## Tasks

Built in #25

## Story Points: 0

###### #33 USER STORY: Change project Published status

As \*\*content creator\*\*, I can \*\*change a project published status \*\*, so that \*\*the project is available to users when marked as published or hidden when it is marked as draft\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have selected one or more DIY Projects from the DIY projects list

\*\*Then\*\* I can toggle the published flag `Status` so that the DIY Project can be shown or hidden from viewing users.

## Tasks

- [x] Task 1 - Modify the DIY Project Admin view (`ThingAdmin`) to update the `status` field

- [x] Task 2 - Add `publish` and `set\_as\_draft` actions to the DIY Project Admin view

## Story Points: 5

###### #34 USER STORY: Change component published status

As \*\*content creator\*\*, I can \*\*change a component published status\*\*, so that \*\*the component information is available to users when marked as published or hidden when it is marked as draft\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Creator` user (member of the `Content Creator` group)

\*\*When\*\* I have selected a component from a DIY Project

\*\*Then\*\* I can toggle the published flag `Status` so that this Component from the DIY Project can be shown or hidden from viewing users.

## Tasks

- [x] Task 1 - Modify the DIY Component Admin view (`ComponentInLiner`) to update the `status` field

## Story Points: 1

##### #41 EPIC: View DIY Projects

As a \*\*user\*\*, I can \*\*access the published DIY Projects\*\*, so that \*\*I can follow their instructions and replicate them\*\*

###### #19 USER STORY: View list of published projects

As \*\*user\*\*, I can \*\*view a list of published projects\*\*, so that \*\*I can choose what projects to view in detail\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a user

\*\*When\*\* I am in any website page

\*\*Then\*\* I can access the website home page clicking on the website logo or brand name

- Acceptance Criteria 2

\*\*Given\*\* that I am a user

\*\*When\*\* I am in the home website page

\*\*Then\*\* I can view the list of all the published DIY Projects

- Acceptance Criteria 3

\*\*Given\*\* The list of all the published DIY Projects is displayed

\*\*When\*\* each DIY project is displayed

\*\*Then\*\* The Title, featured pictured, author, published date and number of likes should be displayed

- Acceptance Criteria 4

\*\*Given\*\* that I am a user

\*\*When\*\* I select a DIY Projects from the DIY Project list

\*\*Then\*\* I get redirected to the selected DIY Project details webpage

## Tasks

- [x] Task 1 - Add a paginated list of DIY Published projects to the index template

## Story Points: 5

###### #20 USER STORY: View details of a DIY project

As \*\*user\*\*, I can \*\*Access the details of the selected DIY project\*\*, so that \*\*I can view the components and the instructions\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a user

\*\*When\*\* I am redirected to the DIY Project details website page

\*\*Then\*\* I can view the DIY Project's Title, featured image, author, Published date, Description, number of components, number of Instructions, Number of likes, the components list and the Instructions list

- Acceptance Criteria 2

\*\*Given\*\* that I am a user

\*\*When\*\* viewing the components list

\*\*Then\*\* I can see each component's title, featured image, and description

- Acceptance Criteria 3

\*\*Given\*\* that I am a user

\*\*When\*\* viewing the instructions list

\*\*Then\*\* I can see each instruction's title, and its content

## Tasks

- [x] Task 1 - Create the Django view and controllers to handle DIY details

- [x] Task 2 - Create a template for the DIY Project details

- [x] Task 3 - Add to the details template a presentation header with all the DIY Project details

- [x] Task 4 - Add to the details template a list of the components related to the DIY Project

- [x] Task 5 - Add to the details template a list of the instructions related to the DIY Project

## Story Points: 8

###### #21 USER STORY: View Components

As \*\*user\*\*, I can \*\*select and view a project component\*\*, so that \*\*I can access the details of the component\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a user

\*\*When\*\* in the DIY Project details webpage, and selecting a component from the components list

\*\*Then\*\* if the component is also a DIY Project, I get redirected to this DIY Project detail website page

## Tasks

- [x] Task 3 - In the DIY Project details template, add a link to its DIY project detail for all the components related to the DIY Project and are DIY Projects themselves.

## Story Points: 3

##### #38 EPIC: User registration

As a \*\*user\*\*, I can \*\*register myself on the website\*\*, so that \*\*hold my information about the website\*\*

###### #12 USER STORY: Register an account

As \*\*anonymous user\*\*, I can \*\*register myself\*\*, so that \*\*I can select my favourite content\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am an `Anonymous` user (non registered)

\*\*When\*\* I am in the sign-in website page

\*\*Then\*\* I can access the sign-up page following the `sign up` link

- Acceptance Criteria 2

\*\*Given\*\* that I am an `Anonymous` user (non registered)

\*\*When\*\* I am in any website page

\*\*Then\*\* I can access the sign-up page selecting the sin-up option in the user menu

- Acceptance Criteria 3

\*\*Given\*\* that I am a non `Anonymous` user (user already registered)

\*\*When\*\* I am in the home page

\*\*Then\*\* the `join us ...` link should not be available

- Acceptance Criteria 4

\*\*Given\*\* that I am in the sign-up webpage

\*\*When\*\* I have entered a valid e-mail address and a valid password

\*\*Then\*\* The new account is created

## Tasks

- [x] Task 1 - Adapt Django's allauth accounts application to allow user registration via email.

- [x] Task 2 - Create the sign-up option menu in the user menu in the base template

## Story Points: 3

###### #13 USER STORY: Validate e-mail

As \*\*user\*\*, I can \*\*validate my own e-mail\*\*, so that \*\*I can sign in the website\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a new `Registered` user

\*\*When\*\* I have created an account using the sign-up webpage

\*\*Then\*\* an e-mail validation is sent to the new account e-mail address

\*\*Then\*\* we will be redirected to a webpage with instructions on how to verify the e-mail Address.

- Acceptance Criteria 2

\*\*Given\*\* that I am a new `Registered` user that has received an e-mail verification e-mail

\*\*When\*\* I follow the link contained in the e-mail verification e-mail

\*\*Then\*\* I will be directed to the `confirm-Email` webpage

- Acceptance Criteria 3

\*\*Given\*\* that I am a new `Registered` user redirected to the `confirm-Email` webpage

\*\*When\*\* I click the confirm button

\*\*Then\*\* the e-mail is confirmed

\*\*Then\*\* I am redirected to the sign-in webpage

## Tasks

- [x] Task 1 - Create an e-mail account for Django's automated e-mail (on Gmail, Yahoo, etc)

- [x] Task 1 - Adapt Django's allauth accounts application to allow e-mail validation.

## Story Points: 3

###### #14 USER STORY: Sign in

As \*\*user\*\*, I can \*\*sign on the website\*\*, so that \*\*I can mark my favourite content\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Registered` user not logged in

\*\*When\*\* I am in any website page

\*\*Then\*\* I can access the sign-in web page selecting the `sign in` option in the user menu

- Acceptance Criteria 2

\*\*Given\*\* that I am a `Registered` user not logged in

\*\*When\*\* I am in the home page

\*\*Then\*\* I can access the sign-in web page following the `join us ...` link

- Acceptance Criteria 3

\*\*Given\*\* that I am in the sign-in webpage

\*\*When\*\* I have entered a valid e-mail address and a valid password

\*\*Then\*\* I become logged in the website

\*\*Then\*\* The user menu button should contain my username

- Acceptance Criteria 4

\*\*Given\*\* that I am a user not logged in

\*\*When\*\* I am in any website page

\*\*Then\*\* The user menu button should contain `Anonymous`

## Tasks

- [x] Task 1 - Adapt Django's allauth accounts application to allow user sign-in via email.

- [x] Task 2 - Create the `sign in` option menu in the user menu in the base template

## Story Points: 3

###### #15 USER STORY: Change e-mail

As \*\*user\*\*, I can \*\*change my own e-mail address\*\*, so that \*\*I can sign in on the website with new e-mail address\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a `Registered` user

\*\*When\*\* I am in any website page

\*\*Then\*\* I can access the E-mail web page selecting the `E-mail` option in the user menu

- Acceptance Criteria 2

\*\*Given\*\* that I am in the e-mail webpage

\*\*When\*\* I fill the email field with a valid e-mail address and use the button `Add Email`

\*\*Then\*\* The e-mail address is added to user's list of addresses

- Acceptance Criteria 3

\*\*Given\*\* that I am in the e-mail webpage and I have more than one e-mail addresses

\*\*When\*\* I select an e-mail address from the list and use the button `Make Primary`

\*\*Then\*\* The selected e-mail address will be the new `Primary` that will be used for sign-in

## Tasks

- [x] Task 1 - Adapt Django's allauth accounts application to allow users to manage their e-mail addresses.

## Story Points: 1

###### #16 USER STORY: Delete own account

As \*\*user\*\*, I can \*\*Delete my own account\*\*, so that \*\*I can erase my information from the website\*\*

Note: This user story is not included in the [DIY Site Backlog](https://github.com/Juanma1313/dys/milestone/1) iteration.

###### #17 USER STORY: Administrate users

As \*\*Administrator\*\*, I can \*\*Create/modify/delete user accounts and their privileges\*\*, so that \*\*I can assign who signs in on the website and who can create or modify content\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in any website page

\*\*Then\*\* I can access the admin web page selecting the `Admin` option in the user menu

- Acceptance Criteria 2

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any user

- Acceptance Criteria 2

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any user group

- Acceptance Criteria 2

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any privilege of any user and any group

- Acceptance Criteria 3

\*\*Given\*\* that I am an `Administrator` user

\*\*When\*\* I am in admin website page

\*\*Then\*\* I can add/delete/modify any privilege of any user and any group

## Tasks

- [x] Task 1 - Install and setup Django's allauth and its accounts and interfaces, extensions.

- [x] Task 2 - Create at least one administrator account

- [x] Task 3 - Customize allauth templates to the website base template

- [x] Task 4 - Create `Content Creators` user group with necessary privileges for content creators

## Story Points: 8

###### #18 USER STORY: Reset/Change password

As \*\*user\*\*, I can \*\*Reset and change my own password\*\*, so that \*\*I can recover my account if I forget my password or enforce security to my account\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am in the sign-in webpage

\*\*When\*\* following the link `Forgot your password?'

\*\*Then\*\* I get redirected to the Password/reset webpage

- Acceptance Criteria 2

\*\*Given\*\* that I am in the Password/reset webpage

\*\*When\*\* I fill my user e-mail address and use the `Reset My Password` button

\*\*Then\*\* a password reset e-mail is sent to the e-mail address

\*\*Then\*\* we will be redirected to a webpage with instructions on how to reset the password.

- Acceptance Criteria 3

\*\*Given\*\* that I am a Registered user that has received a password reset e-mail

\*\*When\*\* I follow the link contained in the password reset e-mail

\*\*Then\*\* I will be redirected to the change password webpage

- Acceptance Criteria 4

\*\*Given\*\* that I am a Registered user that has been redirected to the change password webpage

\*\*When\*\* I fill in a valid password

\*\*Then\*\* The account password will be changed to the new password

\*\*Then\*\* I will be redirected to the change password done webpage

## Tasks

- [x] Task 1 - Adapt Django's allauth accounts application to allow user password reset.

## Story Points: 1

###### #22 USER STORY: Mark a project as favourite

As \*\*a registered user\*\*, I can \*\*mark a project as favourite\*\*, so that \*\*I can keep a collection of favourite projects and the projects get rated\*\*

---

## Acceptance Criteria

- Acceptance Criteria 1

\*\*Given\*\* that I am a signed-in user

\*\*When\*\* I am viewing a DIY Project details

\*\*Then\*\* I can click on the favourite icon to toggle the current DIY Project as favourite for me.

## Tasks

- [x] Task 1 - Modify the DIY Project class model to handle DIY likes.

- [x] Task 2 - Create the Django view and controllers to handle DIY likes

- [x] Task 3 - Modify DIY Project detail template to handle DIY likes

## Story Points: 5

#### Web Site Sketches

To outline the design of the website, I have created the following sketches to guide the development of the user interface. But, since for the administration of the website I will mainly rely on the Django administration interface, and even though it will be customized, I didn't find the need to design the wireframes of those pages, since the style is already defined in the framework and regular users will not access it anyways.

For this type of website content, I assume that the authors will produce descriptions and instructions with long texts, so the main design is also oriented towards large screens. I then adapt this content so that it can also be viewed by an eventual user of a small screen device.

##### Base Page

This design only sketches the header and footer of the pages. This should be consistent across all pages.

##### Home page and DIY projects list

This page is the landing page and displays a carousel of images and phrases that define the website and shows the list of available DIY projects as well as inviting the users to register to allow them to select their favourite projects.



##### DIY project Details

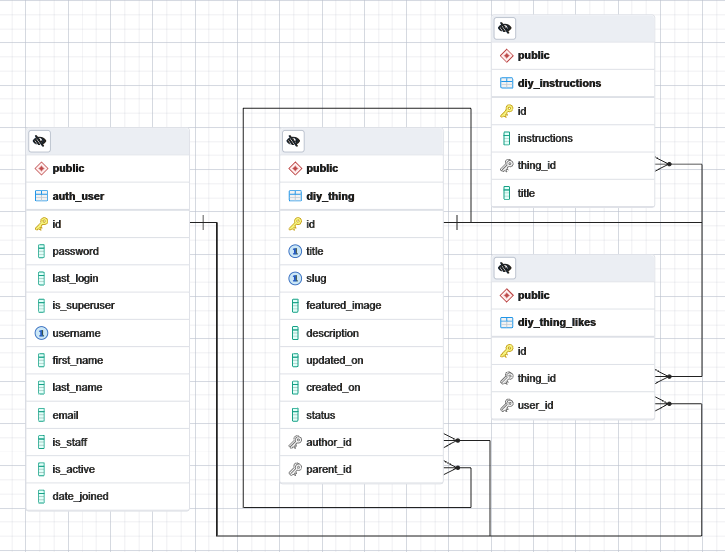


#### Database

It is a customer requirement to use the services of Elephant as an external SQL service to host the website database. This service provider has servers running PostgreSQL servers.

Since the development framework is based on Django, we will use its built in user management functionality from its allauth package.

Django's Allauth uses a database user table that its represented on the leeft of the following figure, which is included in this figure tu identify the relations with the website non built in database model.



The table named `diy\_thing` stores the details of DIY projects and DIY components. A special detail of the implementation is that this table has a cyclic reference to itself in order to accommodate a tree structure for the DIY components, which in turn are DIY projects themselves.

The table named 'diy\_instructions' stores the details of the DIY Project Instructions and a 'one-to-many' relationship with its corresponding DIY Project.

The table called `diy\_thing\_likes` functions as an intermediate table that stores references to all of each user's favourite DIY Projects as a `many to many` relationship.

All these tables are automatically created by Django from the application model classes `Thing` and `Instructions`, so with only two classes, all the necessary database structure is created.

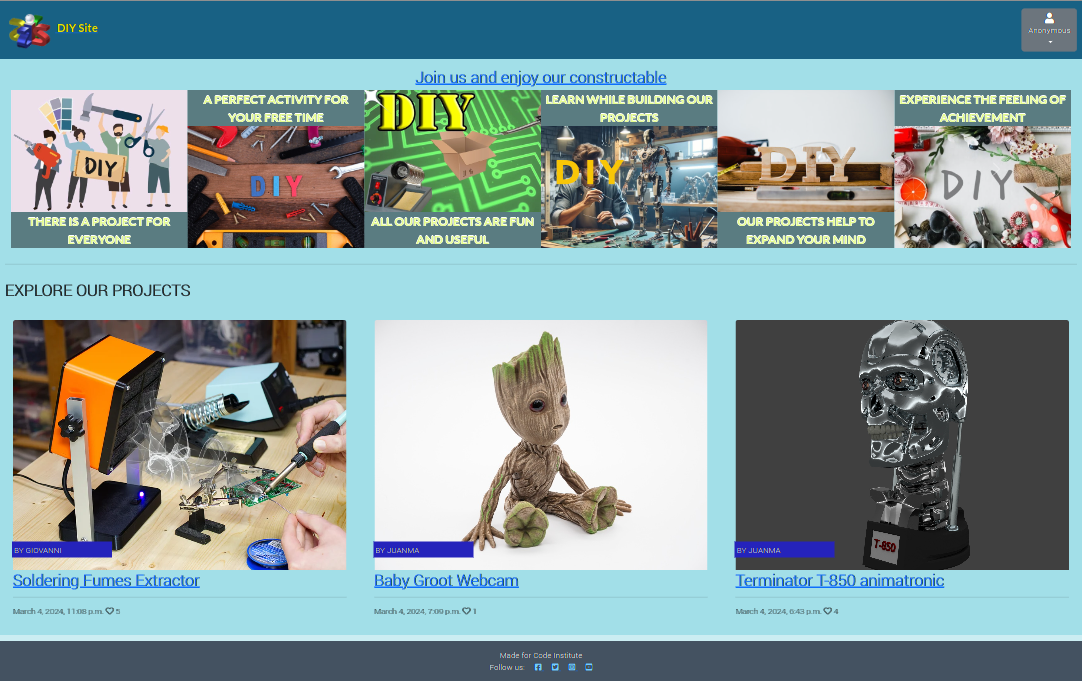
Although there is no specific class for the diy\_thing\_likes table, it is also produced by Django when it recognizes the `many-to-many` relationship in the `Thing` class in the application model.

## Features

#### Users Home page

The home page or landing page is presented on the root page of the site's domain. Since the sole purpose of this website is to share personal DIY projects, I designed it in a minimalist way with a single carousel banner that explains the purpose of the site and a paginated list of DIY projects available for sharing, where each of the DIY projects is represented by a link showing its title, featured image, author, time of publication and number of likes.

When an unregistered user browses the site, a link at the top of the page encourages them to register and redirects them to the registration page. On the other hand, if the user is already logged in, the link will encourage them to view and rate the DIY projects, and this link will redirect them to the list of DIY projects below..



#### Footer

The main navigation bar appears at the top of all user web pages containing links to the social media sites where the website owner have presence.



#### Users Navigation Bar

The main navigation bar appears at the top of all user web pages, and maintains the minimalist style. It displays only the site's logo and name, as a link to the home page, and a drop-down menu that allows the user to access registration, login, logout and other functions.

The drop-down menu is located in the top right corner and is represented by a user icon containing the user's name if signed in.



##### User Dropdown Menu

This menu is located at the top right of the user's navigation bar, and its drop-down content varies depending on the user's context.

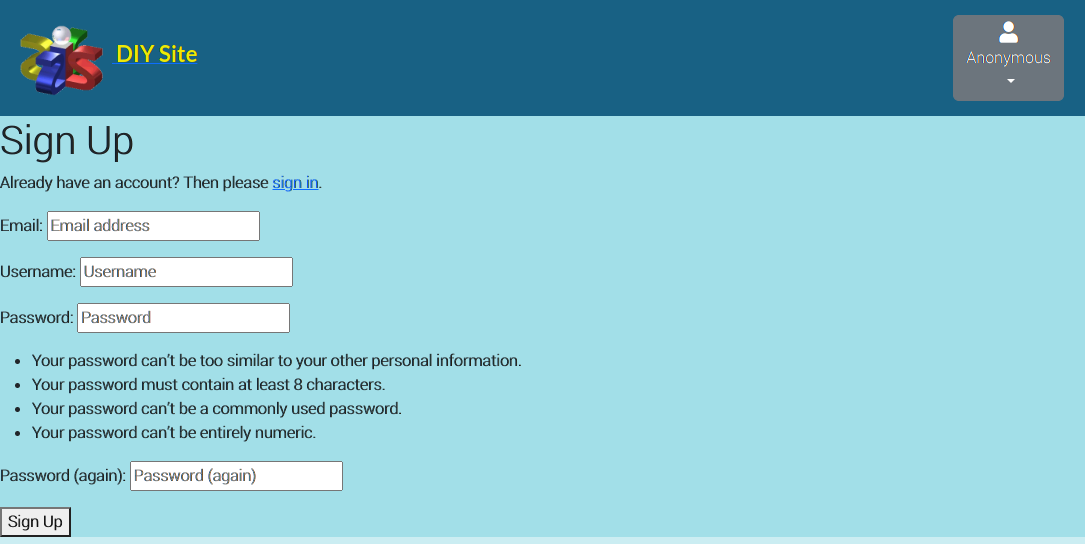
###### User Menu options:

* **Sign Up:** This menu option redirects the user to the accounts/registration web page so that he/she can self-register on the website.
* **Sign In:** This menu option redirects the user to the accounts/login webpage so that the user can sign in the website with an email address
* **Sign Out:** This menu option redirects the user to the accounts/logout webpage so that the user can sign out the website.
* **E-mail:** This menu option redirects the user to the accounts/email webpage so that the user can self-manage his/her email addresses.
* **Admin:** This menu option redirects the user to the admin web page so that he/she can perform administrative tasks.

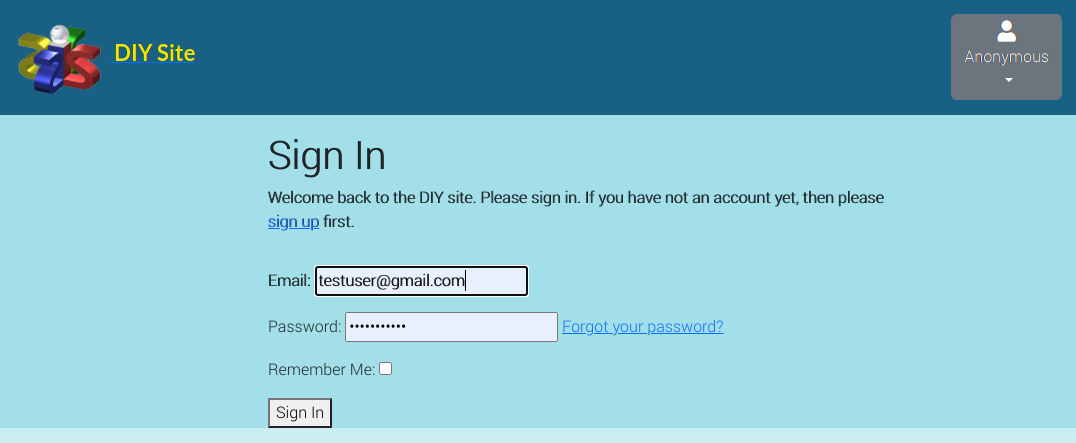
###### Drop-down Menu content.

| Anonimous User | Regular User | Administrators/Content Creators |
| --- | --- | --- |
|  |  |  |

#### Sign Up



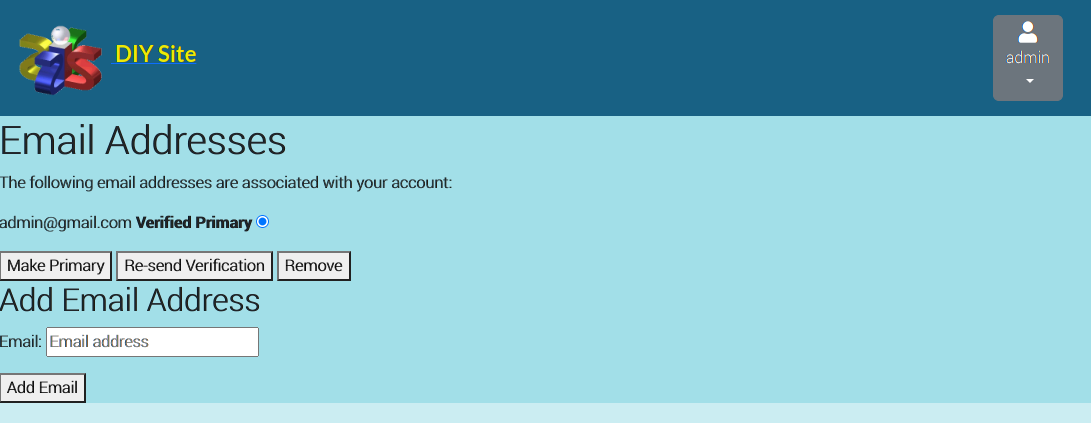
#### Sign In



#### Sign Out



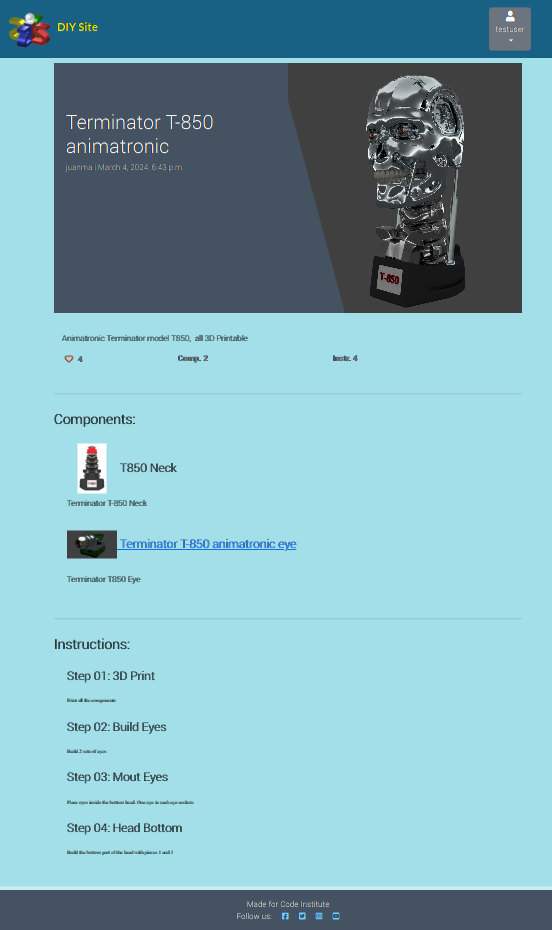
#### Email



#### DIY Project details

Once the user has selected a DIY project link, the details web page is displayed giving access to the details, the DIY components and the Instructions, so that the user can follow them to reproduce the object described in the project..

The layout of this page is highly dependent on the design of the DIY project content creator, as the description fields and instructions are displayed as HTML documents that can be fully customized with colours and images, allowing for a very flexible result.The order in which the information is presented is one of the few limitations that have been set when displaying the DIY project.

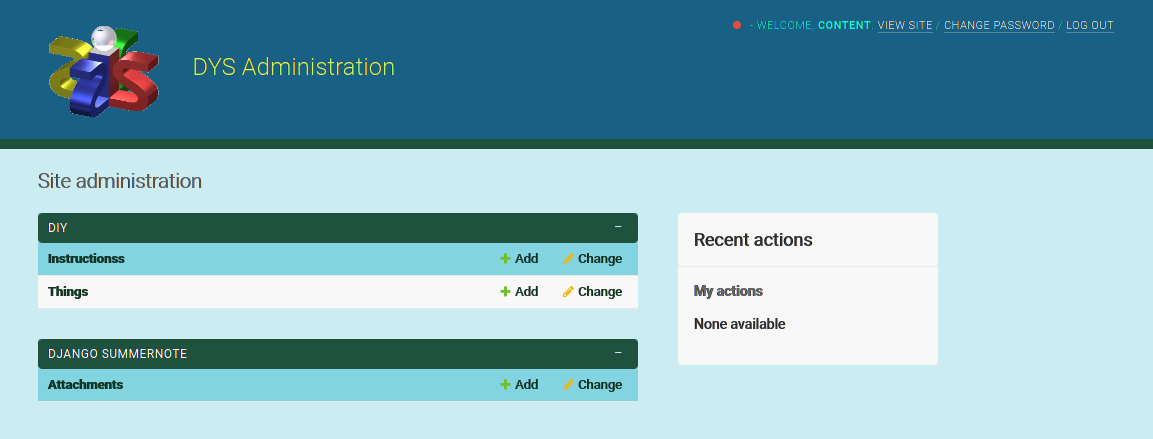


#### Administration pages

All the administrative tasks have been delegated to Django Allauth extensions, wich brings a lot of development releaf, but also comes with layout design limitations. Only the logo, website name and color schemas are customized using a modified copy of the original Django Allouth templates.

##### Content Creators homepage

This page gives the content creators access to all DIY project (Things), individual instructions documents or any document attachment stored by Django's Sumernote extension.

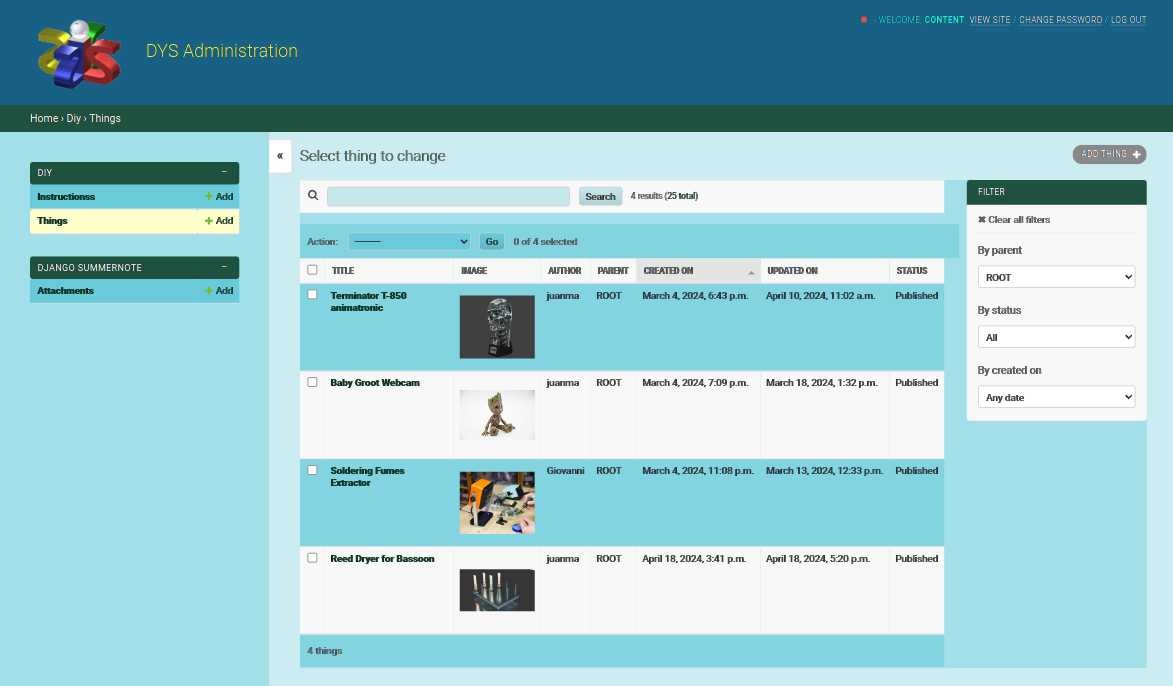


##### DIY-Things webpage

This page provides a list of the DIY projects (Things) created. The filter pannel will allow the creator to narrow down the list based on its parent project, the published status and the creation date. We can also search a particular text in the title or description text of the DIY projects if we want to locate a particular project.

Marking the desired Things we can use the Delete/Publish/Set as draft actions on any number of selected Things at once.

Clicking on any Thing's tytle from the list will present the Change Thing page for the selected Thing, where all the DIY Project data can be edited

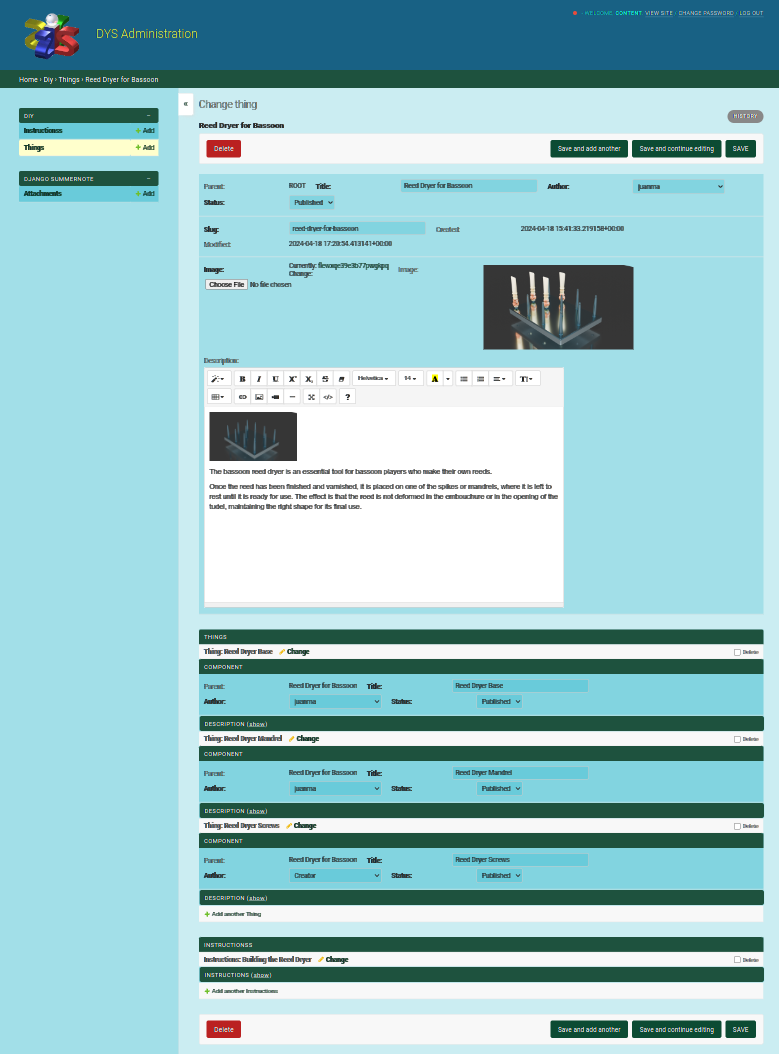


##### Change Thing webpages

This is probably the most complex administrative web page on this website. It combines the functions of adding/deleting/modifying DIY project details, components and instructions on a single page. It is designed so that the Creator can manage the DIY project information without having to navigate through many pages or make unnecessary clicks.

Once you have made changes to your DIY Project, there are three ways to save. Save all changes and return to the list, save all changes and continue editing in the current DIY Project, or save all changes and create a new blank DIY Project. Exiting this page or reloading without saving, will immediately delete all changes made to the DIY Project.

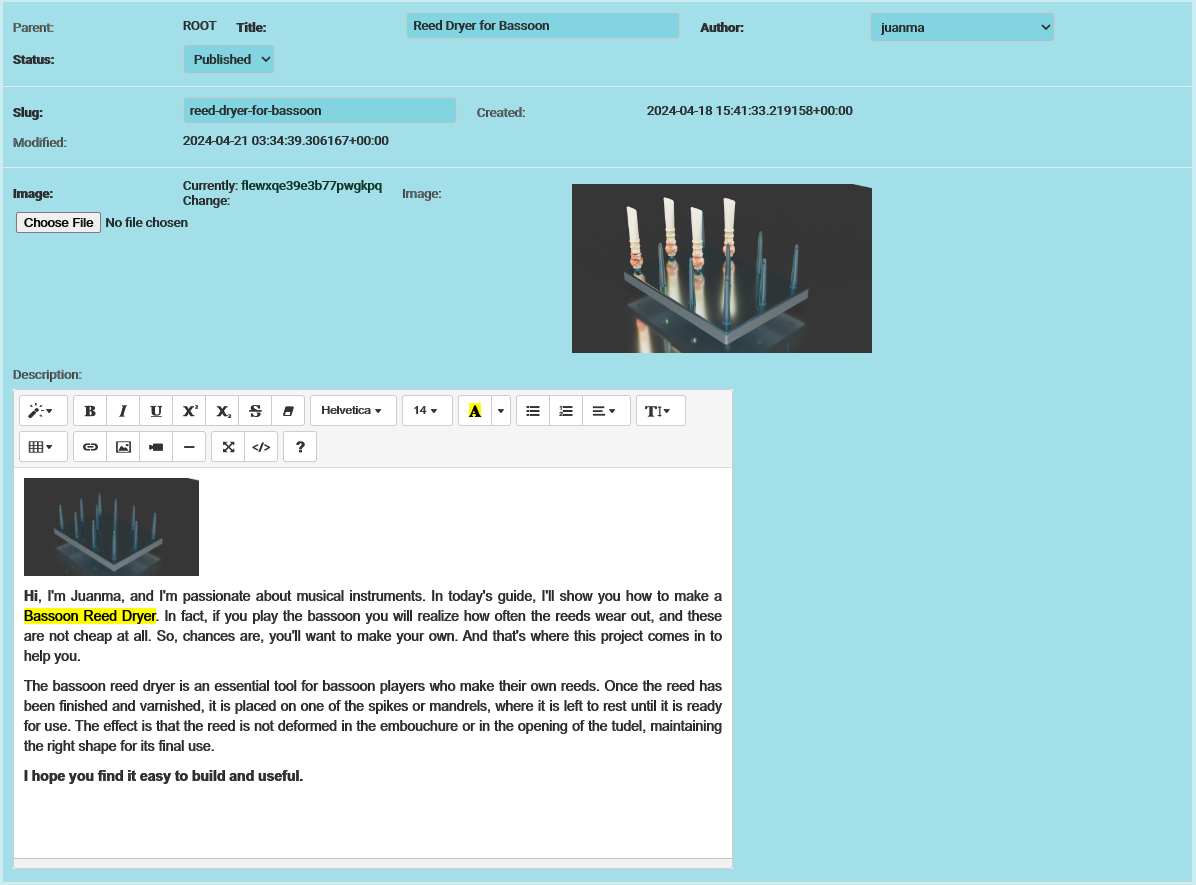
On this web page we can distinguish three areas: the details of the DIY Project at the top, followed by the list of components called "Things", and at the bottom, the list of instructions called "Instructions".



###### Change the DIY Project details

This upper area of the "Change Thing webpage" allows the content creator to change the Title, the Author, and the Status (Draft/Published). It allows also to assign a Featured Image that will be used in the list of DIY Projects at the home page.

The most important element that can be modified in this area areis the Description. This field has an embebded WSIWYG editor from the Django Sumernote extension, that allows to insert text, images, videos, links and any HTML/CSS needed to present the project. It is the responsability of the creator to write a description that will be displayed correctly to the final user .

DIY Project (Thing) details

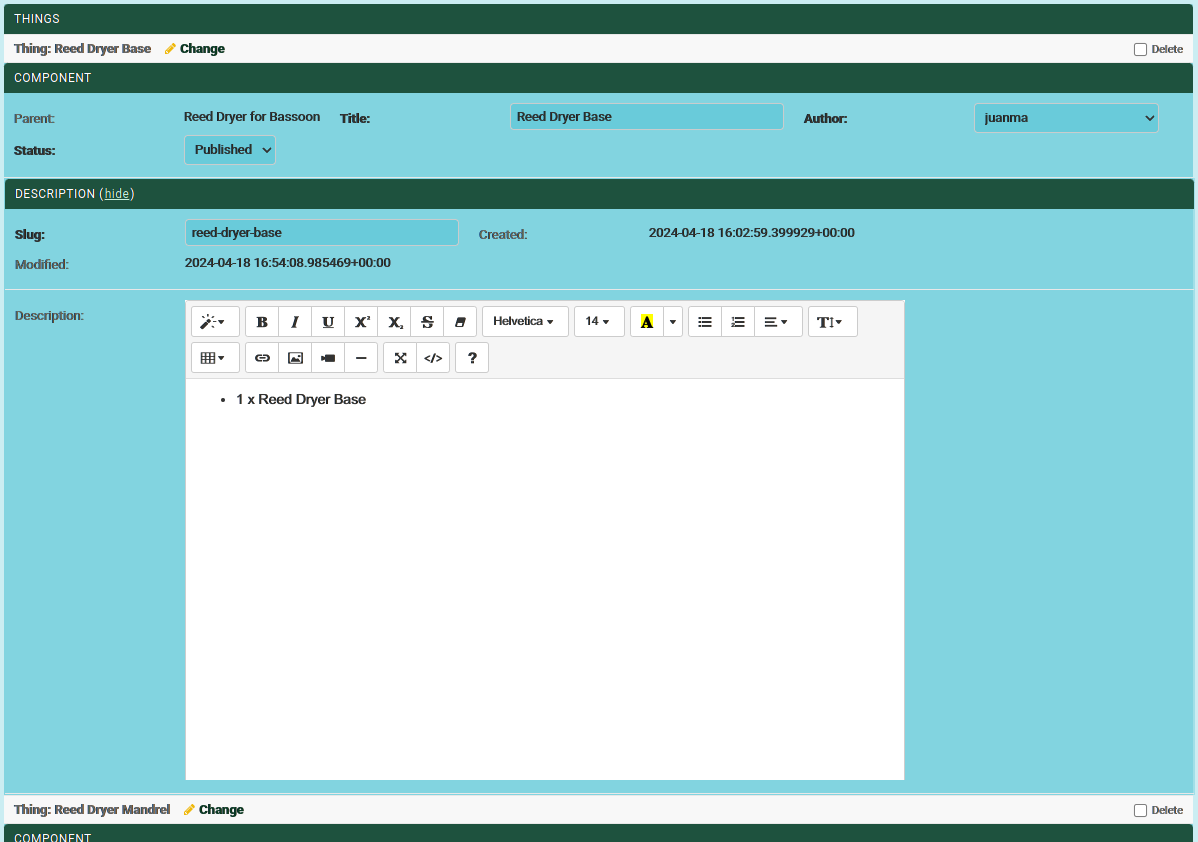
###### Change the DIY Project ***Components (Things)***

The DIY Components (Things) of a DIY Project (Thing) are in fact themselves Thing elements, so they can also be edited like a normal DIY Project. The only difference is that DIY Components have a "parent" object to which they belong.

New DIY Components can be added using the "Add another Thing" link at the bottom of the list.

Default DIY Componets list

On the Change Thing web page, each of its DIY Components are listed under "Things", and only their Title, Author, Status and Description are shown. To minimize screen space and rendering time, the Description is hidden by default, but can be displayed and edited using the `show` link next to the 'DESCRIPTION' label.

DIY Component `show` Description

The rest of the information about a DIY component is not displayed, but can be edited by clicking on the 'change' link next to the DIY component name, which will bring up a new, separate modal Change Thing page for this DIY component.

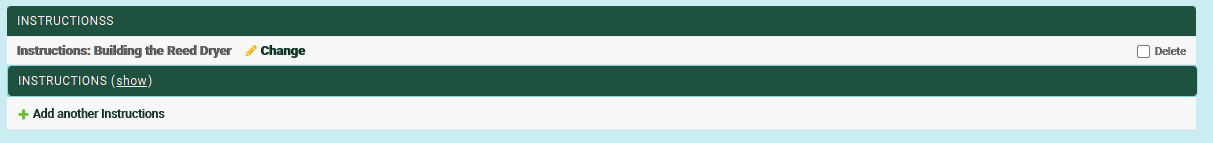
###### Change the DIY Project ***Instructions***

On the Change Thing website, each of your DIY Instructions is listed under "INSTRUCTIONSS", and only its Title is displayed. To minimise screen space and rendering time, the Description is hidden by default, but can be displayed and edited using the `show` link next to the 'INSTRUCTIONS' label.

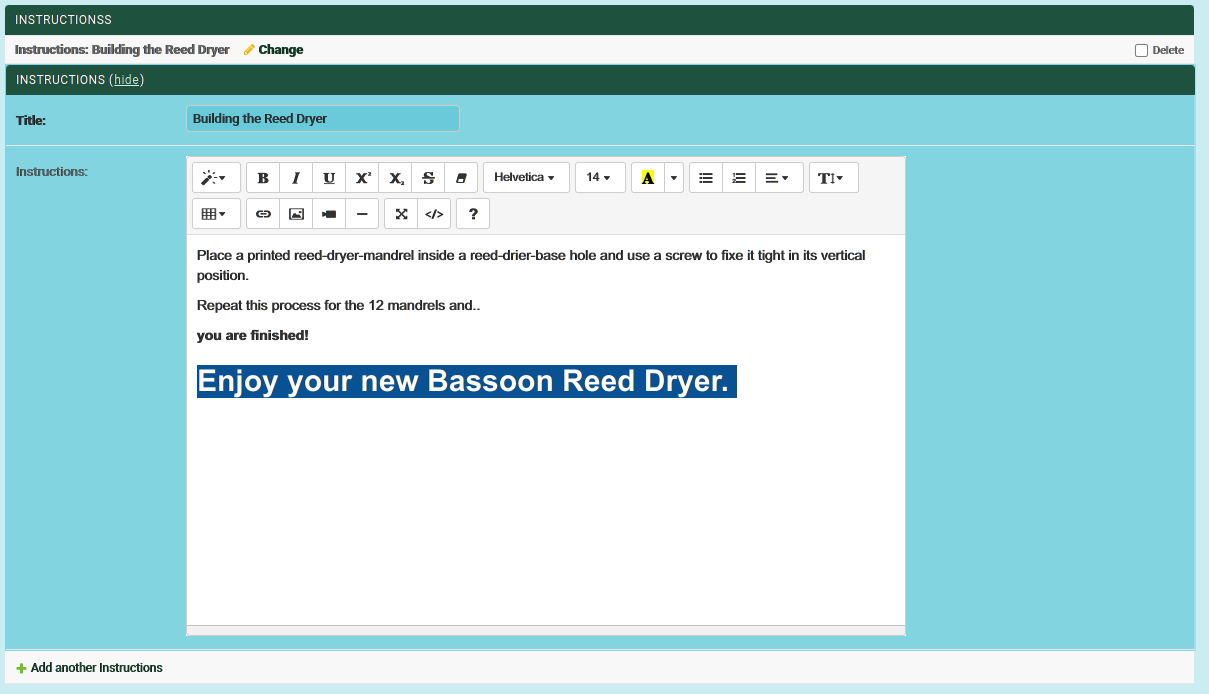
Note: The list name "INSTRUCTIONSS" is automatically created by Django admin based on the name assigned to the class model. More research on Django is needed to modify this name or further customize the layout of the list.

The list of Instructions is ordered by Title. This is useful if you need to maintain a particular order in the instructions, so assigning titles such as "Step 01", "Step 02", etc. could be used to maintain a strict order.

New Instructions elements can be added using the "Add another Instructions" link at the bottom of the list.

Default DIY Instructions list

Instructions field, like the Description fields, has an embebded WSIWYG editor from the Django Sumernote extension, that allows to insert text, images, videos, links and any HTML/CSS needed to present the instructions. It is the responsability of the creator to write a Instructions element that will be displayed correctly to the final user .

DIY Instructions element `show` Description

The Instructions element can be also edited by clicking on the 'change' link next to the DIY Instructions element title, which will bring up a new, separate modal Change Instructions page.

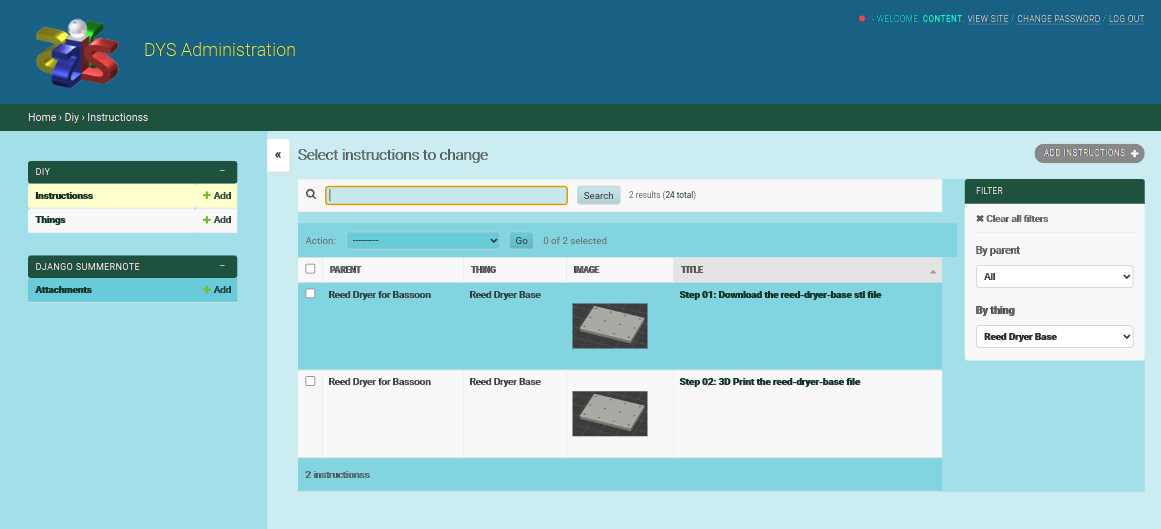
##### DIY-Instructions webpage

This page provides a list of the Instruction items created, and is a convenient way to access a particular Instruction item without having to open the DIY Project to which it belongs.

The filtering panel will allow the creator to narrow down the list according to their DIY Project (By Thing) or by the parent of the DIY Project. It is also possible to search on a specific text in the title or in the text of the DIY Project Instructions if we want to locate a specific Instructions element.

By marking the desired Instruction items, the Delete action can be used on any number of selected Instruction items at the same time.

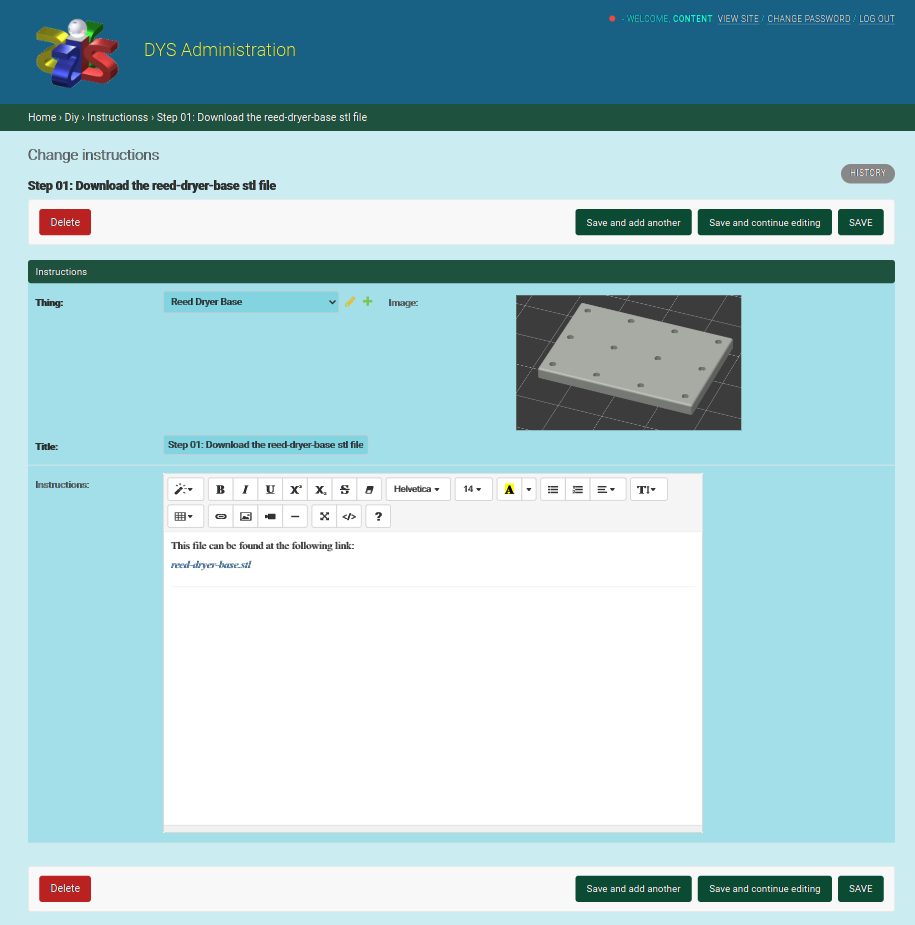
Clicking on the title of any Instructions item in the list opens its Change Instructions page, where the Instructions data can be edited.



##### Change **Instructions** webpage

In this page Instructions item data can be edited.

Once all the required changes to the Instructions item have been made, there are three ways to save the data. Save all changes and return to the list, save all changes and continue editing in the current Instructions item, or save all changes and create a new blank Instructions item. Exiting this page or reloading without saving, will immediately delete all changes made to the Instructions item.



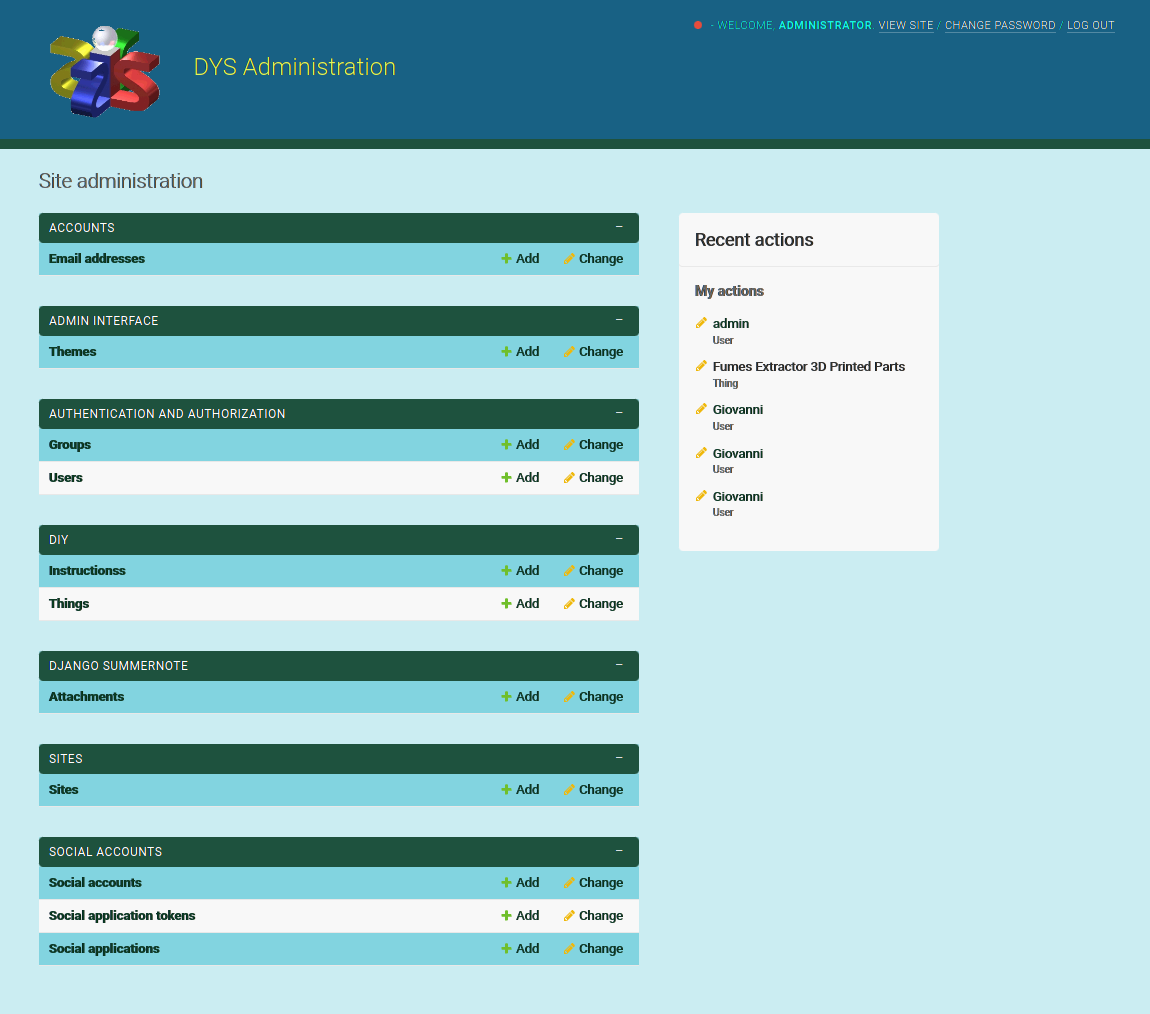
The Instructions field has an embebded WSIWYG editor from the Django Sumernote extension, that allows to insert text, images, videos, links and any HTML/CSS needed to present the instructions. It is the responsability of the creator to write a Instructions element that will be displayed correctly to the final user .

The Instructions elements are presented to the end user sorted by Title. This is useful if a particular order needs to be maintained in the instructions, so assigning titles such as "Step 01", "Step 02", etc. could be used to maintain a strict order. Special care should be taken when changing the Title, as it may affect the order in which the instructions are presented to the end-user.

##### Administrators webpage

This web page is only available when the user is an administrator, and provides full access to all website settings.

This documentation only shows the main admin page, as the DIY pages are the same as described for content creators, and the rest can be found in the Django documentation.



## Future Enhancements

In the current project it is assumed that content creation is done by administrators and staff, who have unrestricted access to all content in the system, and that users can only browse a master list of DIY projects. In a future implementation, the following capabilities would be desired:

* Users should be able to browse the list of their favorite DIY Projects
* The list of DIY projects should be prioritised based on user ratings.
* Users should be able to filter and search the DIY project database
* Projects should be clasified by DIY Project type
* Users should be able to comment on any DIY Project
* There should be a moderator function, automatic or manual, to validate and control user comments.
* Users should be able to loging using their social networks accounts
* The user should be able to exchange messages directly with other users.
* Users should be able to upload and modify their own DIY projects by becoming content creators.
* Users should be able to modify their own DIY projects
* A shopping bag should be implemented for user to buy published DIY-Projects
* A online payment system should be added to the website.
* Grant access to free content and restrict user access to pay per view content
* Users should be able to put a price on their DIY projects
* Users should be able to buy DIY Projects
* Users should be able to receive compensation for their DIY projects sold.
* Users should have the option to donate for any free shared content

## Testing

### Testing Strategy

Since the webpages for the unknown or regular users that are the webpages handled from scratch by the project which are minimalistic and simple, and the administrative pages are mostly handled by a well tested Django Allauth extensions, I decided that a manual testing strategy for the application can be easier to deploy and more effective in terms of time and human efforts.

As for the code validation, I use online applications to check that the source code for all the languages used are in line with the required standards.

### Validating the Source

#### Django Python code

In order to validate the python code as PEP8 compliant, I have run it througt "CI Python Linter" online tool and checked the results

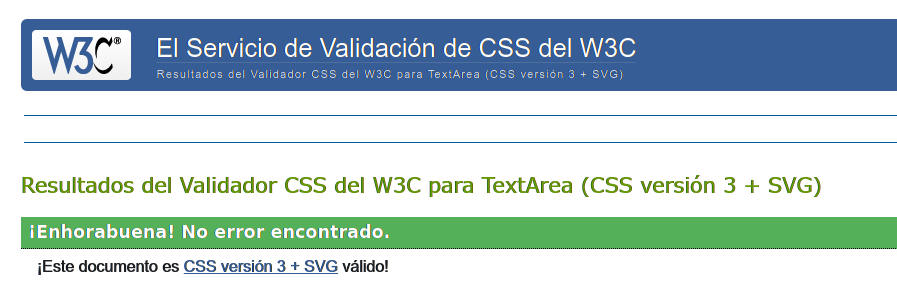
The only PEP8 error I systematically allow is the "E501 line too long". This is a personal decision and the reason I allow this error is because nowadays developers' monitors no longer have a limit of 80 characters per line and following this rule is extremely annoying for me.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| diy\_project | | | | |
| Done | File | Result | Pass | Additional info |
| x | asgi.py | No errors | x |  |
| x | settings.py | 9 errors - E501 Line too long | x | E501 ignored |
| x | urls.py | No errors | x |  |
| x | wsgi.py | No errors | x |  |
| diy | | | | |
| x | admin.py | 16 errors - E501 Line too long | x | E501 ignored |
| x | apps.py | No errors | x |  |
| x | models.py | 5 errors - E501 Line too long | x | E501 ignored |
| x | tests.py | No errors | x |  |
| x | urls.py | No errors | x |  |
| x | views.py | 2 errors - E501 Line too long | x | E501 ignored |

#### CSS code

In order to validate the CSS code, I tried to pass the url of the deployed website to the CSS validator from W3C, but there where many warnings related to the many third party extensions used by Bootstrap and others.

I decide to validate only the style.css file that is really the only one I have customized. The result is as follows:

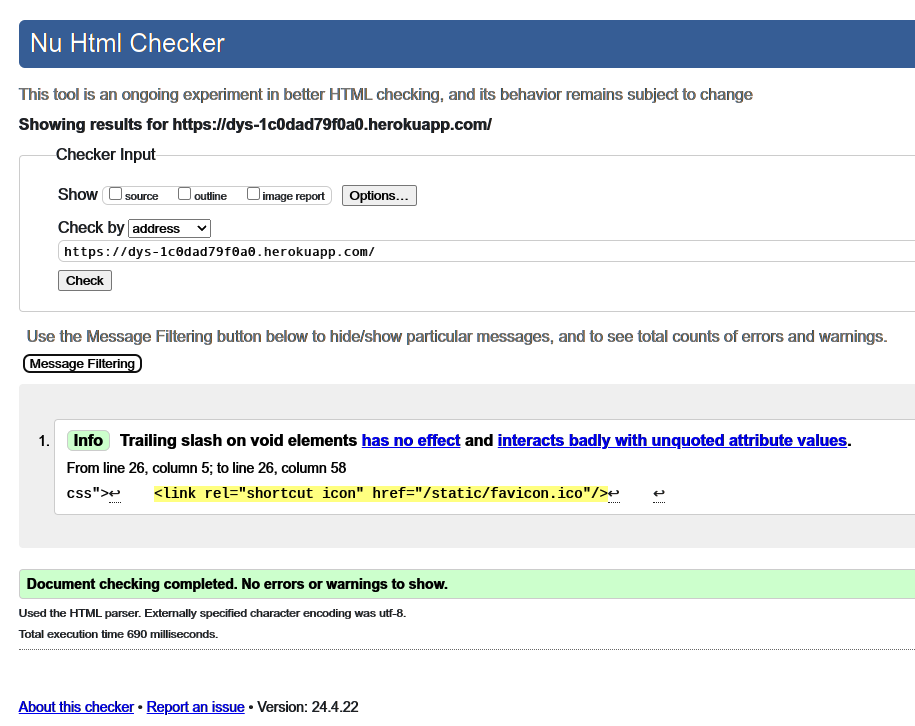
Result from CSS validator from W3C for style.css file

Unfortunately the result is in spanish, but translates to "Congratulations!, No errors found" and "This document is CSS version 3 + SVG Valid!",

The validator only reported 2 warnings regarding two features from a vendor extension, which its no problem.

#### HTML code

In order to validate HTML, I pass the deployed website urls that I want to test to the HTML validator from W3C.

Result of W3C HTML validator

### Testing the application

Most of the bugs have been detected and fixed in the development process. And Deployment in Heroku counts for me as a Smoke Test for the application. Then all the specific features can be tested.

In order to test most of the developed features, I created a list of test cases to run in the deployed website

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TC-01 - user website navigation** | | | | | | |
| **Done** | **Step** | **Description** | **Expectations** | **Result** | **Pass** | **Additional info** |
| x | 1 | Navigate to the home page | Home page is shown | OK | x | - |
| x | 2 | The banner carrousel | automatically move left one picture every 3 seconds | OK | x |  |
| x | 3 | In any page click on header brand logo | redirects to home page | OK | x |  |
| x | 4 | In any page click on header brand name | redirects to home page | OK | x |  |
| x | 5 | Anonymous user in any page click on user menu button | offers `Sign Up` and `Sign In` options | OK |  |  |
| x | 6 | Regular signed-in user in any page click on user menu button | offers `E-mail` and `Sign Out` options | OK |  |  |
| x | 7 | Administrator or creator signed-in user in any page click on user menu button | offers `Admin`, `E-mail`, and `Sign Out` options | OK | x |  |
| x | 8 | In any page click on `Sign Up` option menu | redirects to the sign-up page | OK | x |  |
| x | 9 | In any page Click on `Sign In` option menu | redirects to the sign-in page | OK | x |  |
| x | 10 | Anonymous user In Home page click on 'Join us..' link | redirects to the sign-in page | OK | x |  |
| X | 11 | Signed in user In Home page click on 'Check out ...' link | Redirects to the DIY project list below | OK | x |  |
| x | 12 | In Home page click on `<` on the banner carrousel | moves the carrousel one picture left | OK | x |  |
| x | 13 | In Home page click on `>` on the banner carrousel | Moves the carrousel one picture right | OK | x |  |
| x | 14 | In Home page click on any DIY project picture | redirects to the selected DIY Project details page | OK | x |  |
| x | 15 | In Home page click on any DIY project Title | redirects to the selected DIY Project details page | OK | x |  |
| x | 16 | In DIY Project details page click on heart icon (likes) | The heart icon toggles the fill colour  The number of likes change accordingly | OK | x |  |
| x | 17 | In any page click on any social media icon at the footer | Opens the new webpage to the corresponding social network | OK | x | Action works correctly but does not point to any particular social account. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TC-02 – Unregistered user sign-up procedure** | | | | | | |
| **Done** | **Step** | **Description** | **Expectations** | **Result** | **Pass** | **Additional info** |
| x | 1 | In sign-up page enter an invalid email, a valid username and valid passwords, and push sign-up button (e.g. “theemail” as email) | Message indicating invalid email | OK | x |  |
| x | 2 | In sign-up page enter a valid email, an invalid username and valid passwords, and push sign-up button (e.g. “test!!user” as user name) | Message indicating invalid user name | OK | x |  |
| x | 3 | In sign-up page enter a valid email, a valid username and two different passwords, and push sign-up button | Message indicating passwords must be the same | OK | x |  |
| x | 4 | In sign-up page enter a blank email, a valid username and valid passwords, and push sign-up button | Message indicating to fill email | OK | x |  |
| x | 5 | In sign-up page enter a valid email, a blank username and valid passwords, and push sign-up button | Message indicating to fill username | OK | x |  |
| x | 6 | In sign-up page enter a valid email, a valid username and empty passwords, and push sign-up button | Message indicating to fill passwords | OK | x |  |
| x | 7 | In sign-up page enter a valid email, a valid username and valid passwords, and push sign-up button | Redirected to confirm-email page.  An e-mail requesting to confirm the account is sent to the email with the “confirmation link” | OK | x |  |
| x | 8 | In sign-up page enter an existing valid email, a different valid username and valid passwords, and push sign-up button | Redirected to confirm-email page.  An e-mail warning of a duplicate e-mail attempt is sent to the email | OK | x |  |
| x | 9 | Navigate to a valid ‘confirmation link’ | Presents the email confirmation page with a confirm button | OK | x |  |
| x | 10 | Navigate to an invalid ‘confirmation link’ | Presents the email confirmation page with the “invalid confirmation link” message  Offers a “request a password reset” link | OK | x |  |
| x | 11 | In an email confirmation page press the confirm button | Redirected to sign-in page | OK | x |  |
| x | 12 | In sign-up page enter a valid email, an existing valid username and valid passwords, and push sign-up button | Message indicating that the username already exists | OK | x |  |
| x | 13 | In the sign-in page click the ‘sign up’ link | redirects to the sign-up page | OK | x |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TC-03 – registered user sign-in procedure** | | | | | | |
| **Done** | **Step** | **Description** | **Expectations** | **Result** | **Pass** | **Additional info** |
| x | 1 | In sign-in pag enter an invalid email, a valid passwords, and push sign-in button (e.g. “username” as email) | Message indicating invalid email | OK | x |  |
| x | 2 | In sign-in page enter a blank email, a valid password, and push sign-in button | Message indicating to fill email | OK | x |  |
| x | 3 | In sign-in page enter a valid registered email, an incorrect password, and push sign-in button | Message indicating wrong email or password | OK | x |  |
| x | 4 | In sign-in page enter a valid non registered email, and password, and push sign-in button | Message indicating wrong email or password | OK | x |  |
| x | 5 | In sign-in page enter a valid registered email and correct password, and push sign-in button | Redirects to home page  User menu button shows the username  User menu options are ‘e-mail’ and ‘Sign Out’ | OK | x |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TC-04 – registered user reset password procedure** | | | | | | |
| **Done** | **Step** | **Description** | **Expectations** | **Result** | **Pass** | **Additional info** |
| x | 1 | In sign-in page click the ‘Forgot your Password?’ link | Redirects to password-reset page | OK | x |  |
| x | 2 | In password-reset page enter a blank email and press ‘reset’ button | Message indicating to fill email | OK | x |  |
| x | 3 | In password-reset page enter an invalid email and press ‘reset’ button | Message indicating invalid email | OK | x |  |
| x | 4 | In password-reset page enter a valid unregistered email and press ‘reset’ button | Redirects to reset-done page  presents a message saying that the reset email is sent  An email is sent to the email address with a link to the sign-up page | OK | x |  |
| x | 5 | In sign-in page enter a valid registered email and correct password, and push sign-in button | Redirects to reset-done page  presents a message saying that the reset email is sent  An email is sent to the email address with a link to the change-password page | OK | x |  |
| x | 6 | In password-change page enter blank passwords and press change password button | Message indicating to fill passwords | OK | x |  |
| x | 7 | In password-change page enter different passwords and press change password button | Message indicating to fill same passwords | OK | x |  |
| x | 8 | In password-change page enter correct passwords and press change password button | Redirects to change-password-done page.  The new password is active | OK | x |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TC-05 – signed in user change e-mail procedure** | | | | | | |
| **Done** | **Step** | **Description** | **Expectations** | **Result** | **Pass** | **Additional info** |
| x | 1 | In any page Click on `E-mail` option menu | Redirects to email-addresses page | OK | x |  |
| x | 2 | In email-addresses page enter a blank email and press ‘Add Email’ button | Message indicating to fill email | OK | x |  |
| x | 3 | In email-addresses page enter an invalid email and press ‘Add Email’ button | Message indicating invalid email | OK | x |  |
| x | 4 | In email-addresses page enter a valid already registered email and press ‘Add Email’ button | Message indicating email address already registered | OK | x |  |
| x | 5 | In email-addresses page enter a valid non registered email and press ‘Add Email’ button | A message is displayed requesting to confirm the email address.  An e-mail requesting to confirm the account is sent to the email address with the “confirmation link”  The email-address page is reloaded  The new email address is added to the users email addresses list and it is marked as Unverified | OK | x |  |
| x | 6 | In email-addresses page select an unverified email address and press ‘Make Primary’ button | Message indicating that the email address has not been verified | OK | x |  |
| x | 7 | In email-addresses page select a verified email address and press ‘Make Primary’ button | A message is displayed confirming that the email address is primary now.  An e-mail requesting to confirm the account is sent to the email address with the “confirmation link”  The email-address page is reloaded  The selected email address is marked as Primary  The new email address is active for next sign-in | OK | x |  |

#### The development environment

For the development of this application the following list of frameworks, services and resources where used

* Microsoft Visual Studio Code v1.87 for linux was used as de IDE for the development.
* Git was used for version control and file transfer between the IDE and the repository in GitHub.
* GitHub is the online service that stores the development project
* Moqups.com is the online service I used to create the UI sketches for the website
* GIMP 2.10 used for all the image manipulation.
* PGAdmin IV was used to verify and generate the ERD representation for the SQL database
* Django v3.2.24 was used for the runtime frameworks
* Bootstrap v5.3.3 is used for the front end development as a CDN service
* ElephantSQL is the online service that hosts the SQL database for development and the deployed version
* Cloudinary is the online service that hosts the media files for the deployed version
* Fontawesome is the service that provides with fonts and icons as a CDN service
* Heroku is the online service that hosts the web site for the deployed version.

#### Resources

* Ghithub online documentation
* The Django and its extensions (Allauth, Sumernote, etc) online documentation
* The HTML and CSS documentation
* Bootstrap 5 online documentation
* The Code Institute reference material.

Files and Images

* diy-projects-01.jpg from constructionmarketingassociation.org
* diy-projects-02.jpg from homeaddict.io
* diy-projects-03.webp from greenbuildinginsider.com
* diy-projects-04.jpg from www.simplifiedbuilding.com
* diy-projects-05.jpg youtube.com/c/NoSkillsRequired
* diy-projects-06.jpg youtube.com/TrendVerse Talks
* DIY Project "Reed Dryer for Basson"

Author: Juan Manuel de las Heras

Images: Juan Manuel de las Heras

* DIY Project "Soldering Fumes Extractor"

Author: Giovanni Aggiustatutto

Images: Giovanni Aggiustatutto

* + DIY Project "Baby Groot Webcam"

Author: Juan Manuel de las Heras

Images: Juan Manuel de las Heras

* + DIY Project "Terminator T850 animatronics"

Author: Juan Manuel de las Heras

Images: Juan Manuel de las Heras

## Deployment

The site was deployed via Heroku. Access the site using the following link -[DIY Site](https://dys-1c0dad79f0a0.herokuapp.com/)

For the deployment in Heroku, we assume that the media files server and the database server is already defined and a base database is restored from a backup. Otherwise, the process of deployment must be done using a development environment first, and use the fixtures from Django to populate the database with the minimum information. In that scenario, first clonning the repository could be a better choice.

### How to deploy the project on Heroku

1. Sign up / Log in to [Heroku](https://www.heroku.com/)
2. From the main Heroku Dashboard page select 'New' and then 'Create New App'
3. Give the project a name and select the region, then select create app.

I used ‘dys’ as the name for my Heroku app

1. Once finished creating the app, navigate to the setting tab, and select the `Reveal Config Vars` in the config vars section. And create the following variables:
   * ALLOWED\_HOSTS = [the address for the deployed application host]

usually, after the deployment, this variable can be found in the application Settings tab, under Domains section

* + DEVELOPMENT = False

This is a very important variable. If it is not present, the application will asume False, but if it is specifically set to True, the deployed application will be in debug mode, and could be vulnerable for hackers.

* + SECRET\_KEY = [your secret key]

With Python, get\_random\_secret\_key() from django.core.management.utils can be used to create this variable

* + DATABASE\_URL =[your database service url]

This variable is provided by the database server, in my case its ElepantSQL. And it looks like :

“postgres://kjsdhf:sdfçsdafsadfsdfg987dgfs987fdg@dumbo.db.elefantsql.com/kjsdhf”

* + CLOUDINARY\_URL = [your media file service url]

This variable is provided by the media files server, in my case its Cloudinary. And it looks like :

“cloudinary://826348763248682436:5dc5sd5f5sdf5fd5dfs5@klhjsadkhdfsag”

* + EMAIL\_HOST = [your email service host]

In my case I use google host “smtp.gmail.com”

* + EMAIL\_HOST\_USER = [your website email account username]

in my case I use “pruebasdeprogramacion13@gmail.com”

* + EMAIL\_HOST\_PASSWORD = [your website email account password]

Be careful with his password, because some email service providers do not support user password, but a special application password.

This variables would have to be defined in the env.py file of the project if the application runs in the local computer of a development environment.

1. In Heroku, navigate to the “Deploy” tab and use the option “GitHub (connect to GitHub)”

Scroll down and click on “Connect to GitHub”

In the repository search option, type the name of this repository or your own repository name if you have cloned it, and then use the “Connect” option and wait for the new “Deploy Branch” option appear.

Use the “Deploy Branch” option and wait until the deployment is complete.

Once you receive the message “The app was successfully deployed” you can access the deployed applications

1. Once the application is deployed, do not forget to assign the ALLOWED\_HOSTS variable from the step 4 if it was not assigned earlier.

#### Create a clone of this repository

Creating a clone enables you to make a copy of the repository allowing you to run the application in your local development environment:

1. Navigate to [This repository](https://github.com/Juanma1313/dys)
2. click on the arrow on the green ‘code’ button at the top of the list of files
3. select the ‘HTTP’ option to clone using the web URL and copy the URL to your clipboard.
4. At you local computer, use the console and change your directory to the disk directory where you want the application to be cloned.
5. At the console prompt, type 'git clone ' and paste the https link from your clipboard, followed by enter
6. once the git command has finished, you should have a local copy of the repository that can be opened by a IDE of your choice and for example, push it to a new repository in GitHub to deploy it from your own repository

## Acknowledgements

* Giovanni Aggiustatutto author of the “Soldering fumes extractor” DIY Project included in the deployed website as demo.
* Ainhoia de las Heras Hombrados for her help in the authoring of the “Reed Dryer for Bassoon” DIY Project included in the deployed website as demo.
* CI Student Care members, for their patience and care in dealing with my case.
* StackOverflow with some probmottos specifically with Python