DJANGO

A “secure by default” web framework

Søren Aurehøj – KEA Hand-in Software security

NEWSLETTER SIGNUP APPLICATION

Django – batteries included

Executive summary

This assignment consists of two parts, an app made as a Django website according to the specifications given in the KEA hand-in assignment and the corresponding test scripts for creating test input and input validating.

Django is a mature Python web framework created in 2005. It emphasizes security, predictability and stability in the code and has a large and active ecosystem surrounding it. All Django projects have a similar basic structure, since Django enforces a consistent structure. When a new Django project is created with Django-admin, there is a utility called manage.py as default in the project folder, which is later used to generate the correct layout of the app and to migrate the database. This in particular makes Django a good choice securitywise, if the project consists of more than 2-3 developers collaborating, as the consistency in the framework helps in keeping a clean and structured codebase. This is also relevant with regards to onboarding new developers and when the codebase needs a review or a change in functionality later on.

Picking the app apart

This webapp is handling the login and signup part of a website - and is made up of 3 parts, each in separate folders:

1: Aflevering - the main Django app

2: Users – the signup part of the website

3: Templates – where the html for the site is located

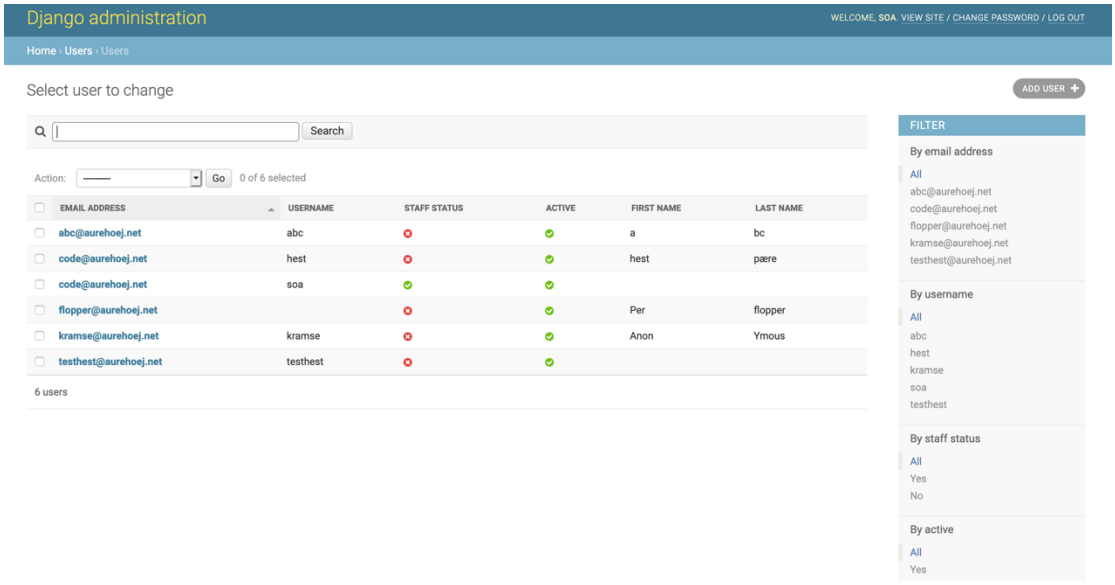
To store the data we use a SQLite3 database which is the default in Django, unless otherwise specified.

In the following I will describe the 3 parts in more detail.

AFLEVERING

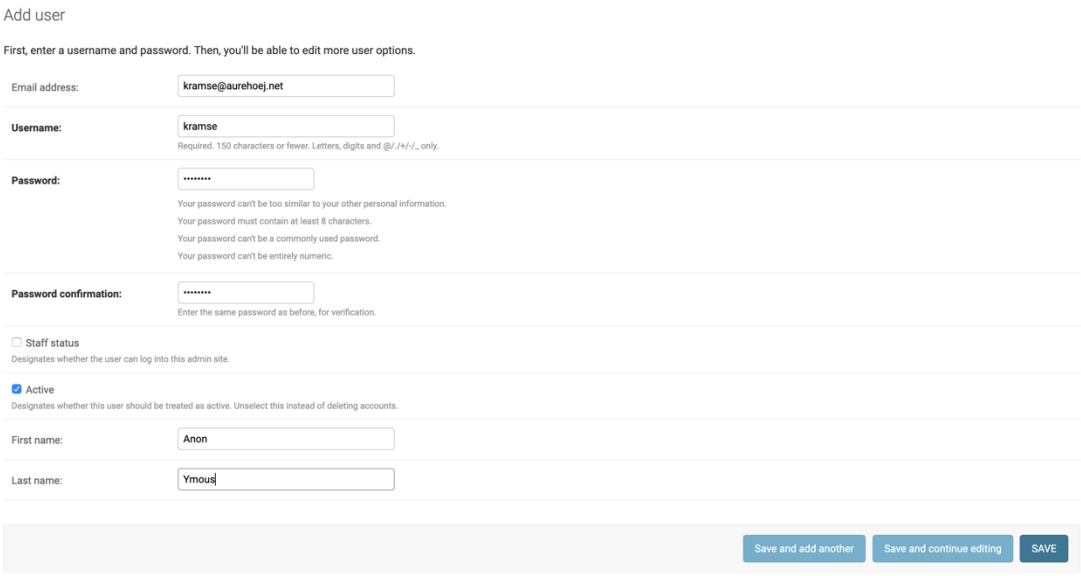
Aflevering is the main cogwheel, the entry point and where it all starts. This is where the primary URL's and the rest of the apps get defined and thereby brought into existence. This is also where the default authentication mechanism is defined in the settings.py file, in this case "users.CustomUser". This is also the place to add further apps besides our "users" app, when needed.

USERS



Users is the meat of this application and is responsible for the forms residing below /users/signup/ with regards to which fields are present in the forms and their validation. Django recommends not to use the default user authentication in the app, but make a dedicated, as modifications to the user scheme later on tend to be much more difficult to accomplish due to the need to keep the original Django user scheme intact. This app makes use of a custom class derived from the standard Django auth model. This custom class “CustomUser” is defined in forms.py in the users folder and afterwards sourced in admin.py, also in the users folder. Django has the ability by default, to make an administrative entry to the website, for a better view and easier access to the database if wanted. Django uses admin.py to give the correct view on the admin-site, which is essential since the default scheme is modified in the users app.

TEMPLATES



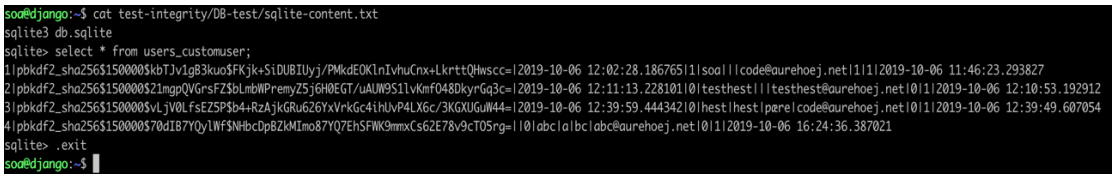
Templates is the folder where the basic html files are placed. It is a nested environment where base.html is inherited by the rest of the files, which makes a consistent layout of the webpages easier attainable, by only having to include things once. In the html files containing forms which is supposed to be “post”-ed, we find a part of the reason Djangos claim for “secure by default”. Django has a standard function to add a token which blocks attempts to cross site request forgeries and keep rudimentary session states. Although it makes scripted input testing more difficult, due to the need of getting the token before a “post” to insert data is made, it is very much a worthwhile feature security-wise.

Part 2 – Sanitize and testing

SANITIZING

In order to sanitize the input to the app, we primarily use the built-in defaults in the framework. This is a part of Django, where the "secure by default" philosophy really excels. Among other things it makes extensive checks on passwords, comparing it to the username and a list of more than 160000 known compromised passwords it installs along with the framework in "common-passwords.txt.gz" as default. Furthermore it has a whitelist of which characters is acceptable in the forms, in order to avoid common injection attacks.

TESTING



Testing of the app is done partly by scripted insertions via the webinterface and also by direct access to the database, with the help of shellscripts. The scripted insertions works by first making a request via cURL to get a csrf-token, and afterwards utilizing this token in a follow-up request with the sign-up data needed by the form. This way it is possible to make a standardized test with both good and bad data. To check the consistency and validity of the database there is a script to dump the database to a JSON file for further consumption by other scripts, a script to insert data in the database and a basic script to test if the database is able to open at all. By opening the database it becomes apparent that the password is well taken care of. The password is encrypted with PBKFD2 with 150000 iterations and an added salt. Furthermore the database contains “date of registration” and “last access” of the user, leaving ample room to mine after stale users among other things.

CONCLUSION

I personally find Django to be an extremely well thought out framework, build with security from the outset, and not being a “bolt-on” afterthought. Although in my opinion, Django does have a very steep learning curve, partly due to the constraints the framework imposes on the developer and partly due to my lack of knowledge of Python, I thoroughly enjoyed my time wrestling with Django and I’m sure I will be using it in the future.