Practical Course "Web Technologies"

Assignment 3

This exercise will give you practice using Django forms and Django models.

Exercise

The Form class is the heart of Django's form handling system. It allows to specify the fields in the form, their layout, display widgets, labels, initial values, valid values, and (once validated) the error messages associated with invalid fields. The class also provides methods for rendering itself in templates using predefined formats (tables, lists, etc.) or for getting the value of any element (enabling fine-grained manual rendering).

a. In your Django web-site's directory create a new Django-app 'LoginApp' and make it available by your web-site. Create a file `urls.py` to specify all URL-routings for the Login-app.

b. In your app's directory create and open a file `loginform.py`. To create a Django-`Form`, import the `forms` library from `django` and derive a class `LoginForm` from the `forms.Form` class.

c. In this class declare a class attribute `login_name` and assign to it an instance of type `CharField` with the following attributes:

   - required form-field
   - `'Enter your login name'` as label
   - maximal length of 10 characters.

   This is the official description of the class `CharField` can be found on
   `https://docs.djangoproject.com/en/4.0/ref/forms/fields/#charfield`

   All `forms`-objects accept a set of so-called core-field arguments when being created
   `https://docs.djangoproject.com/en/4.0/ref/forms/fields/#core-field-arguments`

d. In your app's directory create and open a file `loginform.py`. To create a Django-`Form`, import the `forms` library from `django` and derive a class `LoginForm` from the `forms.Form` class.

   Django provides numerous places where validation of data can be performed. The easiest way to validate a single field is to override the `forms.Form`-method `clean_<fieldname>()` for the field to be checked. Validate that the value of an HTML-input `login_name` does not start with a number. Access to the form's data is possible by accessing a dictionary `cleaned_data` with the field's name as key. In case of an error raise a `ValidationError`.

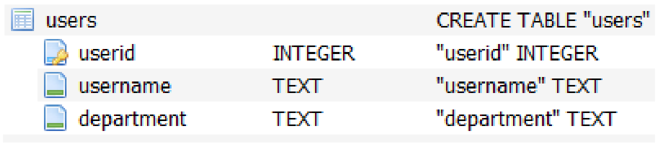
e. In a sub-directory 'templates' prepare a simple HTML-page 'index.html' with a Welcome-message. Beside this message the template must have a div-element which must be shown centered. The div-element consists of a Django template language block with name 'loginblock'. The block consists of a paragraph with text 'Please login' and another paragraph with a submit-button 'Let me login' of width 400px. Create another template file 'login.html' in which the loginblock-block is replaced by a submit-button 'Login' and a table which contains the form for logging in (generated by the view-function `show_login`).

f. In 'views.py' define a function `index` that returns a HttpResponse-object whose content is filled with the result of rendering the template 'index.html'.

   Define a function `login` rendering the default form when it is first called and then then either re-render it with error messages if the data is invalid, or process the data and redirect to a new page if the data is valid. In order to perform these different actions, the view has to be able to know whether it is being called for the first time to render the default form, or a subsequent time to validate data. Note that the decision what to render depends on the HTTP-method.

g. A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table. Each model is a Python class that subclasses `django.db.models.Model`. Each attribute of the model represents a database field.

In the file `models.py` creat a model `Users` for the table users with the following columns

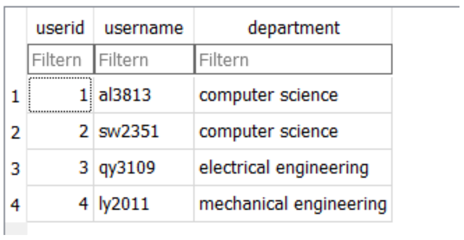


users		CREATE TABLE "users"
userid	INTEGER	"userid" INTEGER
username	TEXT	"username" TEXT
department	TEXT	"department" TEXT

and define a `property`-method `userinfo` providing `username` and `department` together as a string.

Note: `userid` is the table's primary key.

The additional file db.sqlite3 contains a sample-database:



	userid	username	department
	Filtern	Filtern	Filtern
1	1	al3813	computer science
2	2	sw2351	computer science
3	3	qy3109	electrical engineering
4	4	ly2011	mechanical engineering

h. Manager is the interface through which database query operations are provided to Django models. At least one Manager exists for every model in a Django application. By default, Django adds a Manager with the name objects to every Django model class. Calling the Manager's method `all()` (i.e. calling `Users.object.all()`) returns a QuerySet consisting of all model objects in the database. The Manager's method `get` retrieves a single model object. It requires an attribute's value as parameter (e.g. `userid=1234`)

   In the function `login` after validating the form's input retrieve the `Users`-object with the given username. Then, return a simple HTML-page based on a template 'home.html 'in which the loginblock-block is replaced by a text which contains the object's `userinfo`.

Submit the web-site's directory as zip-archive.