**General Instructions**:

1. **Submission date: 22.9.2016**.
2. You should submit a zip file of the Django project's and app's files and folders.

You should create the app folder right next to *manage.py* as we did in class.

If you do so, you can zip the folder that includes *manage.py*

(in class it was the outer mysite/ folder), this folder includes everything.

1. Write your name and ID in a comment at the top of the *settings.py* file.
2. You can submit in singles, pairs, or triplets. If you submit in pairs or triplets, since we use the Moodle system, then each student should submit the files by himself, and all students submitting together should include all their names and IDs in the comment at the top of the *settings.py* file.
3. The assignment will be tested with the default Django deployment: Django 1.8.x Python 3.4.x, sqlite DB and the built-in web server running at <http://127.0.0.1:8000/>. Make sure your project runs smoothly with this configuration (or higher versions).
4. **Write readable code**: use meaningful names for URLs, views, models, methods and variables.
5. **Don't overcomplicate things** **and don’t invent the wheel.**
   1. You should use the built-ins that Django offers.

Remember that one of Django's design goals is to minimize the amount of code needed for the web application.

* 1. Your templates don't have to be visually stylish;

They do need to be short, clear, and organized.

* 1. Working code that is overcomplicated will lose (some) points.

**Exercise Details**:

In this exercise you will write a messaging web application, a simplified version of WhatsApp, Facebook messenger, etc.

There is no need to support groups of users; all messages will have a single receiver.

We will also not support contacts; all registered users of the app can chat with each other.

Models:

You'll need to define a *Message* model.

Each message includes:

A single sender, a single receiver, the message text, and the publication date.

You **don't** need to define a *User* model, you can simply use Django's built-in *User* object

(**from** django.contrib.auth.models **import** User).

The sender and the receiver of a message should both be defined by this *User* object.

What is the relationship between *Message* and *User*? Think about it and define it.

In class we said that the relationship is, many-to-one.

Note that you'll need to define a *related\_name* argument for sender or receiver in the relation definition inside *Message*, because both of them are *User* objects

(see the documentation for details).

http://stackoverflow.com/questions/2642613/what-is-related-name-used-for-in-django

https://docs.djangoproject.com/en/1.10/topics/db/models/

Webpages & URLs:

**The app's main page** should be reached by the app's URL. For example, if your app's name is *whatsapp*, then it will be <http://127.0.0.1:8000/whatsapp/>.

The page should display **links** of all the **registered** users except for the **logged-in** user.

When the user clicks a user's link, he will be directed to the *messages page* that will display all **his** conversations with **that user**.

**The messages page** is reached by the apps' URL followed by /<number>.

For example, if your app's name is *whatsapp*, then it will be reached by <http://127.0.0.1:8000/whatsapp/1/>, <http://127.0.0.1:8000/whatsapp/2/>, etc. , where the number at the end of the URL is an argument **representing** a user.

The page should display **all** the messages exchanged between the logged-in user and the other user represented by the argument, **ordered by date and time**.

There is no need for fancy styling of the messages;

The page should merely display for each message its date and time, its sender, and its text.

The page should also include a small form that allows the user to send a new message, and a link that goes back to the *main page*.

Those should be accompanied by short instructions.

After sending a new message, the same page should be displayed again, this time also displaying the newly sent message.

Both the *main page* and the *messages page* should have their own views and templates.

**The action of sending a new message can have its own view**, although it does not need a template, since the *messages page* should be displayed again after sending the message.

You should also use a **base template** to be inherited by the other templates.

It should include a header text, a footer text, a base title to be extended in the derived templates and an (optional) background image.

The **base template** should also provide a ***Log Out* link** at the bottom of the page that should appear only if the user is **authenticated**.

**Dealing with Authentication**:

As you might have realized reading thus far, the messaging app should support user authentication. Django has a built-in authentication system that should be used in your project. You can read about it [here](https://docs.djangoproject.com/en/1.8/topics/auth/default/).

This document covers everything about Django's authentication and authorization, including permissions and groups.

It is recommended to read this document, but not everything is needed for this exercise.

You should read about *User objects*, *Creating users*, and *Authenticating users* at the beginning, and can skip the *permissions* and *groups* parts.

Then, you should read about Authentication *in web requests* and

About *Authentication Views*.

Here are **some important points for this exercise** about Django's authentication system (summarized from the documentation):

* Django's authentication system deals with everything in the background – checking the user's credentials, creating cookies that indicate the user has authenticated, and passing them along the session, so that the user remains authenticated all the way.
* Django has built-in authentication views. You can use all of them in your app's URLs by adding url('^', include('django.contrib.auth.urls')to your URLConf. **We will only need the *login* and *logout* views in this exercise**.
* Since the *main page* and the *messages page* should only be viewed by logged-in users, then if a non-logged-in user tries to reach them, he should be redirected to a *login page*. The easiest way to do so is by using the *@login\_required* decorator.
* If you use the default login view (the simplest option, no need to define the view and its URL), then *login\_required* will look for LOGIN\_URL in the *settings.py* file. The default (if the setting does not exist) is */accounts/login*. It is recommended to define this setting as */myappname/login*, so that if your app's name is whatsapp it will be reached by <http://127.0.0.1:8000/whatsapp/login/>. You can place this setting right after STATIC\_URL.
* If you use the default *login* view, Django will look for a template called *login.html* inside *templates/registration/*.(somewhere in python folder!!! ,Not in your project )

You can change this by providing a *template\_name* argument, or simply put your template in a *registration/* dir.

* The documentation provides a sample template for *login.html* which you can use (parts of it), or define your own.
* There is no need for a *logout page*. After clicking the *Log Out* link, the user should be redirected to the main page (which itself redirects to the *login page)*.
* You need to define a logout URL in your URLConf that calls the built-in *logout* view. This is because you need to provide the *next\_page* optional argument that will direct to the main page after logging the user out.

If you don't do so, Django will not have a *next\_page* to redirect to, and will look for the default *template\_name*, which is *registration/logged\_out.html*. It will find it in the *admin* app and will go there, and this is not what we want.

* You also need to place your logout URL before the line url('^',include('django.contrib.auth.urls'),**or** define a different URL than *logout/*, perhaps *mylogout/*, so Django won’t call the default view without the *next\_page* argument.

At the first use of the app, users need to register before they can login. Therefore, the *login page* should include something like: Don't have an account? [Register](http://127.0.0.1:8000/watsapp/register/) here. The link should direct to a *registration page*.

Django provides a [Forms API](https://docs.djangoproject.com/en/1.8/topics/forms/), which you are encouraged to learn, and provide some built-in forms for authentication, covered in the authentication document mentioned above.

In particular, you can use the [User Creation Form](https://docs.djangoproject.com/en/1.8/topics/auth/default/#django.contrib.auth.forms.UserCreationForm) for registration.

But this is not mandatory; you can create your own simple registration form in the *registration page*.

All it needs to do is the following:

* Read a username, a password and a password repeat from the user.
* Display the form again with a suitable error message if one of the fields was empty, if the passwords did not match, or if the username was already taken.
* There is no need to check for password length, strength, or the chars it contains.
* If all the fields are OK, create the user, and redirect him to the *main page* or the *login page*.