Sign up form:

What it accomplishes?

The sign up form used in this project is from the django.contrib.auth.forms library where the specific form used is called the UserCreationForm. This form allows us to create a new user in the application. We have accomplished this by altering the UserCreationForm by adding in extra fields as shown in the authentication/forms and we have renamed this to UserRegisterForm. Hence, the problems it solves are that it parses the user data, storing said data and takes care of hashing, and adding salt to the password as well.

How does this technology accomplish what it does?

The form itself is within authentication/templates/Authentication/Register.html. This form will be within the standard html form with the method set to POST. Now, when submit is clicked a post request will be sent and received by Register() in views. This request will contain the POST request information.

What the form does is basically parses and extracts all the field information. This information is within request.POST and will be in the form of JSON. So each field (username, password etc) will be extracted from this JSON string within request.POST.

After the information is extracted it will be saved. When form.save is called, we are calling UserCreationForm.save function here. Once this function is called two things happen:

1) save_password() is called which is linked here. Here django first calls make_password() listed here, where it basically hashes and salts the password and thus returns the hashed+salted password. This returned password is then set to be the users new secure password. To reinforce that Django does by default returns a salted hash of the password, please visit this
2) Once, the password has been hashed and a secure one is stored another save() is called which causes django to make an insert sql statement which pushes this data to the database. This is listed on django's official website here

When form.save is done, we use a form.cleaned_data.get, which is also used throughout the application. This is a simple function that simply gets a particular dictionary value from the JSON within request.POST.
On completion of the registration, the user is redirected based on the LOGIN_URL variable within course_project/settings.py. This dictates where the application will be sent.

Login Form:

What it accomplishes?

The login form solves the problem of authenticating a user everytime they want to login. Using this form a user can login, if and only if they have registered via the Register form listed above. It authenticates if a user has previously signed up, and if they have then it allows the user to proceed to the application.

How does this technology accomplish what it does?

The login form is initialized with class_project/settings.py. Here you can see the auth_views.LoginView, the code for this can be found here. Here we see the form being initialized in auth_views.LoginView is the AuthenticationForm, linked here.

The auth_views.LoginView is given a template_name which specifies the html file to display. This specifies that the request data from the template_name variable must be recieved. auth_views.LoginView takes the request data which contains the username and password and begins to authenticate the user.

Once the login page is called via auth_views.LoginView, one of two things happen as described here.

1) If a GET request is received then it displays the template along with the AuthenticationForm
2) If a POST request is received then it checks the database for the user information. If the credentials match then it looks in course_project/settings.py and looks for a LOGIN_REDIRECT_URL, which we have set to the home page. However, if the user is not authenticated then it just shows the same page again.

Django Forms(forms.EmailField):

What it Accomplishes?

A basic form class that handles email field inputs within our UserRegistrationForm and UserUpdateForm.

How does this technology accomplish what it does?

The official documentation of the EmailField is here. Please read this to understand how to use this.

The source code for the forms.EmailField is here. This method takes a CharField input, sets a widget using the EmailInput Widget, and validates it.
Finally, this method has an __init__ function which calls the super(), which renders it and marks whether the html is safe or not. It does this marking by calling another mark() function.

Model Querying:

What it accomplishes?

This accomplishes querying data for the right match. This accomplishes the task of getting all logged in and signed up users. This form of querying is present within blog/views.py, in the get_all_logged_in_users() and get_all_users() methods.

How does this technology accomplish what it does?

As described here a model that is created can be queried for specific data, which in turn creates SQL queries on the backend and returns the data that fits that specific query. This act of querying is the equivalent of writing SQL statements such as INSERT, DELETE etc.

To expand more on models and how they are created:

1) The Model will be instantiated as either a Query Class or a RawQuery Class.

2) Once the class is instantiated the query will be converted into an SQLPrepared statement using the __str__ function. This function internally calls sql_with_params

which basically returns an sql string with parameters. This creates a prepared statement-isque environment for security.

Within our get_all_logged_in_users() and get_all_users() methods, we see a filter() and all() function being called on the models. How this works internal is as follows:

1) Now, each Model created within authentication/models is a “manager”(documented here) of a QuerySet class.

2) a) Each time an object.all() is called, then the function all gets called.
b) This function internally calls _chain which basically just returns the entire query set class. For example, in get_all_users() we call a Users.object.all() which returns a QuerySet copy of all the elements within the QuerySet class Users

3) a) Each time a filter() is called, then this function is called.
b) This function takes the arguments, ANDs these arguments to the querySet instance and returns these results.

Hence, all the queries that one sees within the get_all_logged_in_users() and get_all_users() methods, are just the django version of simple SQL statements.

User Model:

What it Accomplishes?

The User Model provided by Django provides us with a class based usage of using all details related to a user instance using the application. This User Model is used within our Message and Profile Models. We create these models to account for different parts of the application, and we use the Django User model to pull user specific data within the Message and Profile models respectively.

How does this technology accomplish what it does?

The official django documentation provided here provides us with a high level overview of the Django model and also gives us information on exactly how to use it. Please refer to this documentation to understand how to use it.

The source code for the Django User Model here shows the class that is responsible for pulling all the information we use.
This User class in turn used the AbstractUser class. This class is where fields like first_name, last_name ,email etc live. This class also calls the UserManager, and places it into an objects variable for further use.

This class also uses the AnonymousUser class to use methods like isAuthenticated.

Finally, this model can be used like any other Model and can be queried as such.

Sessions model:

What it accomplishes?

This basically allows us to get a list of all user IDs based on the sessions.

How does this technology accomplish what it does?

The code for the session model used is here. The Sessions model is implemented by calling SessionStore linked here.

Django Stores session info by simply sending a session key in a cookie called SessionID to the client. This Session Key is associated with data on the server side, which is used by our code. Creating and deleting said cookies our done using Sessions in django and is listed here

SessionStore stores all sessions thus far and it also supports adding and deleting sessions. The SessionStore approach is also talked about in the official django docs.

This Session model will be like any other model (for example Profile model for registration), and can be queried as such. Hence, this will store all session details thus far and a model query is done on this model instance in order to gain only the unexpired sessions. These unexpired sessions are the users currently logged in currently.

Logout:

What it accomplishes?

This logs users out of the application, thus “unauthenticating them”

How does this technology accomplish what it does?

The Logout feature is pretty simple in that all that is done is add LogoutView class in class_project/urls.py. The class is provided a template_name that shows which html template to use. We have given the template authentication/Logout.html which will be displayed. The class being referenced is here. This class takes a request object, takes the session data and then wipes the session data from the list of Sessions.

By doing this, the users session is “closed” hence they would have to authenticate again in order to use the service.

login_required Decorator:

What it accomplishes?

When this decorator is placed above a method, then it only allows the method to be used if the user is logged in and authenticated. This allows for people to only use the various pages other than login/register only if they are an authenticated and registered user.

How does this technology accomplish what it does?

The login_required decorated here, simply calls the is_authenticated function. This function has two instances in two different classes, and based on which class gets called different results are returned. The two cases are:

1) If the user is an actual registered/authenticated user then this method here gets called, which always returns true. Thus function is from the BaseUser class and will only be invoked if the user was logged in the first place.

2) However, if the user was not authenticated in the first place, then this gets called and the answer is always false. This method is a part of the AnonymousUser class which will be invoked if the user is not logged in or authenticated.

Now, if the user was authenticated(is_authenticated returns true) then this decorator allows the method it is above to run smoothly. If the user is not authenticated (is_authenticated returns False) then this decorator does NOT allow the method to run.

What license(s) or terms of service apply to this technology?

The entire application runs on the django framework. The owner of the license of all django libraries is Django Software Foundation as mentioned here. They give the right to use django under many circumstances mentioned in the link provided.

It is mentioned that you can use the libraries within the following conditions of :

1) Service Identification
2) Django-related software project
3) Groups and Events
4) Merchandise
5) Products and services serving the community
6) Other Commercial activity
7) Domain Names
8) Uses outside of this license
9) Community standards
10) Interpretation