# django report II

**NOTE**
**This is django report II**
**If you want to see the licence or the overview of what django is please see django report**

**Models:** This built in django feature is a way to have your database tables represented with each attribute of model being a database field and each model itself being a database table. Model provides different types of fields which can be used to hold data.

For our project we used the ForeignKey (link 3), DateTimeField (link 4), and TextField (link 4).
The field class in django/db/models/fields/_init_.py define what are the possible field types as well as their basic functions.
ForeignKey is defined in  django/db/models/fields/related.py

- https://docs.djangoproject.com/en/3.0/topics/db/models/
- https://docs.djangoproject.com/en/3.0/ref/models/fields/#model-field-types
- https://github.com/django/django/blob/master/django/db/models/fields/related.py
- **https://github.com/django/django/blob/537d422942b53bc0a2b6a51968f379c0de07793c/django/db/models/fields/__init__.py#L85**

**Authentication with contrib.auth:**
**Model**
This build in django feature allows for the uses of models which as discussed above allow your database data to be represented. However this is a special model as it is the database information which is used for authentication such as a username and password. In our project we used it to access the username of a user as well as the permission that user has (what that user can access). Basically this handles the basics of user accounts.

- https://github.com/django/django/blob/537d422942b53bc0a2b6a51968f379c0de07793c/django/contrib/auth/models.py

**Get_user_model**
Used to get the user model of the project. In other words it finds the database table for the users' data. From there we can access things like the user's username. Link 1 is the call link 2 is the function.

- https://github.com/django/django/blob/537d422942b53bc0a2b6a51968f379c0de07793c/django/contrib/auth/__init__.py#L152
- https://github.com/django/django/blob/537d422942b53bc0a2b6a51968f379c0de07793c/django/apps/config.py#L167

**Reverse and reverse lazy:**
In order to  find the value of the absolute URL. Django has a method for their urls in urls/base.py. The reverse function is a method that takes the viewname and uses it to give the full url.

- https://github.com/django/django/blob/537d422942b53bc0a2b6a51968f379c0de07793c/django/urls/base.py

**Views**
After a user logs in, logs out, or is creating an account we'd have to have them fill out / send a form that can be used to authenticate the user. Django has a built in way of handling this by the uses of contrib/auth/views.py. To put it simply this will take the template of the login in, log out, etc and will take the form data to handle the login, log out, etc for you. It does this by defining a class for each action and the methods in those classes can be set within your code to handle the authentication process.

- https://github.com/django/django/blob/master/django/contrib/auth/views.py

**Django.contrib:**
**Messages**
When a message from a post is deleted we want to send a message to the user saying whether or not it was successful. Django has a built in way of doing this by using the success method in django/contrib/messages/api.py. This sends the success status by sending a message by using the add_message function from the same file. The add message function adds a message to the request.

- https://github.com/django/django/blob/master/django/contrib/messages/api.py

**Django.contrib.auth.mixin:**
**LoginRequiredMixin & PermissionRequiredMixin**
When a user arrives at the site we want it such that they can only access the site if they have been logged in and we only want them to see certain features if they have permissions. For example they can only delete a post if they are the ones who posted it. In order to do this django has some functionality with the classes LoginRequiredMixin & PermissionRequiredMixin. LoginRequiredMixin has a function called dispatch which denies access when the user is not logged in. This

can be used to send a user to the login page then they are denied. Then the PermissionRequiredMixin is very similar but instead of denying based on login status you can set the parameters which would allow access to the content. To use this you can override the has_permission function to customize how the permissions are checked. This makes it very versical in use.

- https://github.com/django/django/blob/master/django/contrib/auth/mixins.py

**Django.contrib.auth.forms:**
**UserCreationForm**
When sending data to a database you want the data in good form and to be set through a form. To help with this django has the class UserCreationForm. This class allows you to set the fields necessary to create an account. It then uses models and its filed properties which were discussed above to help store the data away in the database. (Model is like a table so the form data is being set up to be stored into a database table).

- https://github.com/django/django/blob/master/django/contrib/auth/forms.py

**Django.views.generic:**
**CreateView**
When creating a new post we want to create a new object/form. Through a chain of super classes after the object is saved a form is sent by calling render_to_response which creates a response_class. This will handle the http response for us.

- https://github.com/django/django/blob/master/django/views/generic/edit.py

**TemplateView**
This is another way to render a file. This method does as by calling render_to_response which creates a response_class. This will handle the http response for us.

- **https://github.com/django/django/blob/master/django/views/generic/base.py**

**UpdateView**
This is the same as createview above except this is used to update an already made object. Most functions remain the same. Please look above.

    - https://github.com/django/django/blob/master/django/views/generic/edit.py

**DeleteView**

This is used when we want to remove a post. This is done by deleting the objects created by, for example, createview. It does this by removing the object from where the createview object was stored, then supers of the method calls render_to_response which creates a response_class. This will handle the http response for us.

- https://github.com/django/django/blob/master/django/views/generic/edit.py

**ListView**
Similar to templateview, expect this is to handle multiple posts. It does this similarly to templateview expect it used multipleobjecttemplateresponsemixin instead of singleobjecttemplateresponsemixin. Rendering is still done by calling render_to_response which creates a response_class. This will handle the http response for us.

- https://github.com/django/django/blob/master/django/views/generic/list.py

**DetailView**
This will give us the details of one of our posts in the form of a queryset. The details can be reached by passing in the username of the post we're looking for. The detailview used the base detail view class to call render_to_response which creates a response_class. This will handle the http response for us.

- https://github.com/django/django/blob/master/django/views/generic/detail.py

**Template Engine**
Django comes with a template engine to be used in the template html files (NOTE for django these file must be stored in a template directory as storing it in static would lose the dynamic ability of templates). Template engines allow for dynamic html file updating. This is done by the use of {% %} to use things like for x in list or if x in list. Also {{ }} can be used to create variables. This allows for more customization of the templates. Django accomplishes this by first finding the template, reading the template, replacing the special elements in ram to get ready for rendering. For more info on how it works click on the link (https://github.com/django/django/blob/master/django/template/base.py) and a distribution is at the top. To function the template engine uses:
-class engine gets the template engine pieces together to start registering, loading, etc

- https://github.com/django/django/blob/master/django/template/engine.py

-class Library registers template tags and filters

- https://github.com/django/django/blob/master/django/template/loader.py

-class loader_tags loads the content in between the tags
- https://github.com/django/django/blob/master/django/template/loader_tags.py

- class response has custom render for template and sends the temple through httprequest
- https://github.com/django/django/blob/master/django/template/response.py