**Objective 1 - User Accounts with Secure Authentication**

* *Describe how you completed the objectives of the assignment, how your software is structured, and why you made the design decisions you made. A reader should have a good understanding of your software by reading this section without looking at a single line of your code*

**/betterMal/urls.py**



**What does this accomplish?**

When we use Django, every path will have to be configured into a URL. Because betterMAL is the main branch of our project - we are going to incorporate the paths from the lower branches of *forum*and *online\_users* We are going to be using admin as an example, however for our project, we are not going to be using admin. Admin is a path that was already inputted when we start a django project. This is the same as localhost/example. It lets us know what is on each path. In our project, at the path ‘ ‘ which is an empty route so localhost/ we are going to have our forum.urls which includes our signup page, logout page, profile page. This accomplishes what is rendered at each path of our application.

**How did we structure the paths?**

We didn’t want to have to implement many different paths so we decided that in forum/urls.py we would put the paths to allow users to signup, log out, profile, and online.

**What do these libraries do for us?**

**Path :**  This function takes in a route and then returns the corresponding view. A view is a callable that takes in a request and returns a function. In this case it is taking in the path and returning a HTML.

**Include :** Include is a function that takes a Python path and includes other modules that should also be rendered on the page. In our case, we are using the include(pattern\_list) to include URLS that are from forum/urls.py.

**Github for libraries used**

Admin was not used in our project

**Path:** : <https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/urls/conf.py#L57> - Line 57

**Include:** <https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/urls/conf.py> - Line 12

**/betterMAL/wsgi.py**

**Libraries used:**

**get\_wsgi\_application *:***

**Link :** [**https://github.com/django/django/blob/main/django/core/wsgi.py**](https://github.com/django/django/blob/main/django/core/wsgi.py)

* This library is used to return a WSGI callable. A WSGI **(Web Server Gateway Interface)** is a calling convention that allows web servers to forward requests to web applications made using the Python programming language. This is similar to our TCPhandler that we did in our Hws

***WSGIHandler()* :**

**Link to WSGI Handler() :**

[**https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/core/handlers/wsgi.py#L122**](https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/core/handlers/wsgi.py#L122)

**Link to WSGIRequest:**

[***https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/core/handlers/wsgi.py#L64***](https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/core/handlers/wsgi.py#L64)

**Additional Information Used :**

[***http://ivory.idyll.org/articles/wsgi-intro/what-is-wsgi.html***](http://ivory.idyll.org/articles/wsgi-intro/what-is-wsgi.html)

[***https://djangodeconstructed.com/2018/02/15/how-a-request-becomes-a-response-diving-deeper-into-wsgi/***](https://djangodeconstructed.com/2018/02/15/how-a-request-becomes-a-response-diving-deeper-into-wsgi/)

**How does it work?**

* The get\_wsgi\_application will handle the django setup for us and create a WSGI handler which will be stored in a variable call application. This application is used by the server each time a request arrives and then sends a response back to the server to the browser. Django uses WSGI Middleware which is the code that sits between when a HTTPRequest is created to when the HTTPResponse is translated to bytes by Django and sent over the network to a browser.

**Steps in how it works:**

1. A callable is something that can be called. Python will check to see if the object returns True or False, if it returns True then the object will be callable. This callable is named application and takes in two arguments that are sent by the server. One argument is a dictionary of environment variables called **environ** and another is another callable that is called **start\_response**. The environ will have data such as HTTP/request data, methods, query strings, content length, port numbers, headers, environment variables, etc .. that Django will use to complete the request and create a response while the start\_response will be used to pass the response back to the server once a response has been made. Start\_response takes in two arguments (status, response object) : status is a HTTP status string like 200 OK and the response headers is a list of HTTP response headers.
2. Once all the arguments have been sent by the server, the server WSGIHandler will use these arguments to pass a dictionary of data, environ, and start\_response to Django. When it is passed to Django, it will use the environ data to complete the request by parsing to find the path, content length, and also sets a cookie to be sent in the response headers.
3. The data will be sent to get\_response which will contain self and the request and will return the appropriate HTTPResponse object according to the HTTPRequest it received. This HTTPResponse object will be passed into start\_response along with a status and make the final HTTPResponse.

**/betterMAL/settings.py**

*Most of the code here is used for django*

**What is used here?** These are files that are generated when we start a Django project.

* The secret key that django uses to hash things is only used in production.
* Debug set to true allows us to put django in debug mode
* ALLOWED\_HOSTS are the hosts that we allow connections from
* INSTALLED\_APPS are the settings that lets Django know which parts you want to control
* MIDDLEWAREis also settings similar to INSTALLED\_APPS.
* AUTHENTICAITON\_BACKENDS is how Django does authentication
* DATABASES is how Django communicates with the databases
* AUTH\_PASSWORD\_VALIDATORSis for authentication of passwords
* STATICFILES\_DIRis where Django handles static files like pictures

1. **How does Django prevent HTML/JS Injection?**

* HTML and JS injection attacks are also called XSS attacks. XSS attacks are when a user can inject scripts into the browser which will cause either HTML or Javascript to be executed by their victim when they either enter a site or when they press on a link. Django is able to prevent HTML/JS injection by escaping characters that are close to HTML. Django’s template system has an auto-escaping mechanism for stopping XSS. It will automatically escape data that’s inserted into the template to make sure that they’re not being attacked.
* [**https://github.com/django/django/blob/main/django/utils/safestring.py**](https://github.com/django/django/blob/main/django/utils/safestring.py)

1. **How does Django prevent SQL Injection?**

* An SQL injection occurs when a user can put a SQL into an input. When this happens the SQL statement will run on your database and can modify a SQL query or interfere with the database. Django prevents SQL injections by having it’s queries constructed using query parameterization. They have the queries separated from the query’s SQL code so that the user’s input cannot be injected. After they are parameterized they are escaped as a string by the database driver before being added to the query. This is done within Django’s Object Relational Mapping layer.
* [**https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/db/models/sql/query.py**](https://github.com/django/django/blob/7582d913e7db7f32e4cdcfafc177aa77cbbf4332/django/db/models/sql/query.py)

1. **How does Django prevent securing user accounts and passwords?**

* Django protects user passwords by hashing the password before storing it into the database. Django will go into the function def salt and create a salt for hashing. Then in the function def encode (Line 271) it will take the salt and hash it using the pbkdf2 algorithm with a SHA256 hash and then hash it again with the base64 encoding.
* [**https://github.com/django/django/blob/main/django/contrib/auth/hashers.py**](https://github.com/django/django/blob/main/django/contrib/auth/hashers.py)

1. **How does Django protect private content - the server must check that the user is authenticated before sending them private content.**

* Django provides authentication through 3 libraries. First Django will confirm whether the user is who they are by checking if they may login, if they can login then the (1) will return status of None. Django then uses something called sessions which are used to keep track of the “state’ between the site and a browser. These sessions store information with unique data items that are specific to each user, this unique data item is called a “key” which is used by (2) library to be able to store and retrieve data without the risk of giving it to the wrong user.. It is done through the djang.contrib.sessions.backends.signed\_cookies which will create a secret\_key for the user. The cookie is then stored in the site’s database and then will be used to confirm users when library middleware is called. The third library that is given will help authenticate a user through REMOTE\_USER**.** REMOTE\_USERis an environment variable that will be used to check whether the user is in the backend database or not. If it is not set then the user is unable to login.

1. [**https://github.com/django/django/blob/main/django/contrib/auth/forms.py**](https://github.com/django/django/blob/main/django/contrib/auth/forms.py)
2. [**https://github.com/django/django/blob/main/django/contrib/sessions/middleware.py**](https://github.com/django/django/blob/main/django/contrib/sessions/middleware.py)
3. [**https://github.com/django/django/blob/main/django/contrib/auth/middleware.py**](https://github.com/django/django/blob/main/django/contrib/auth/middleware.py)

(Line 15 and Lines 28)