```
<!DOCTYPE html>
<html>
    <body>
        <form method="get">
            {{form}}
             <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

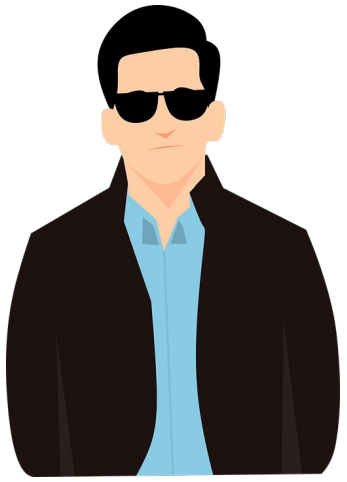
```
<!DOCTYPE html>
<html>
    <body>
        <form method="post">
            {{form}}
             <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

GET and POST

- GET should be used only for requests that do not affect the state of the system.
- Any request that could be used to change the state of the system should use POST.
- GET would also be unsuitable for a password form, because the password would appear in the URL, and thus, also in browser history and server logs, all in plain text. Neither would it be suitable for large quantities of data, or for binary data, such as an image. A Web application that uses GET requests for admin forms is a security risk: it can be easy for an attacker to mimic a form's request to gain access to sensitive parts of the system.
- POST, coupled with other protections like Django's CSRF protection offers more control over access.
- GET, by contrast, bundles the submitted data into a string, and uses this to compose a URL. The URL contains the address where the data must be sent, as well as the data keys and values.
- Django's login form is returned using the POST method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response.
- GET is suitable for things like a web search form, because the URLs that represent a GET request can easily be bookmarked, shared, or resubmitted.

What is Cross Site Request Forgery (CSRF/ XSRF)

A Cross-site request forgery hole is when a malicious site can cause a visitor's browser to make a request to your server that causes a change on the server. The server thinks that because the request comes with the user's cookies, the user wanted to submit that form.



What is Cross Site Request Forgery (CSRF)

Depending on which forms on your site are vulnerable, an attacker might be able to do the following to your victims:

- Log the victim out of your site.
- Change the victim's site preferences on your site.
- Post a comment on your site using the victim's login.
- Transfer funds to another user's account.

Attacks can also be based on the victim's IP address rather than cookies:

- Post an anonymous comment that is shown as coming from the victim's IP address.
- Modify settings on a device such as a wireless router or cable modem.
- Modify an intranet wiki page.
- Perform a distributed password-guessing attack without a botnet.

Cross Site Request Forgery (CSRF) Token

Django provides CSRF Protection with csrf_token which we need to add inside form tag. This token will add a hidden input field with random value in form tag.

templates/enroll / userregistration.html

```
<!DOCTYPE html>
<html>
    <body>
        <form method="post">  {% csrf_token %}
            {{form}}
             <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

Checking which form data has changed

has_changed() - This method is used on your Form when you need to check if the form data has been changed from the initial data.

has_changed() will be True if the data from request.POST differs from what was provided in initial or False otherwise. The result is computed by calling Field.has_changed() for each field in the form.

Example:- Form.has_changed()

changed_data - The changed_data attribute returns a list of the names of the fields whose values in the form's bound data (usually request.POST) differ from what was provided in initial. It returns an empty list if no data differs.

Example:- Form.changed_data