## Invariants

1. A group of people are working together to create website code. Hence there is a development project and files are sticky to a shared unix group.
2. The uwsgi server has a pid and gid. It calls the application function of django, so django will have that same pid and gid.
3. Django must be authenticated to communicate with one or more databases.
4. Nginx communicates with uwsgi through a unix socket. Thus nginx and uwsgi must have read write permission on the socket. Nginx runs as pid and gid www-data.

## Issues

The development directory need not have the same permissions as the release directory. However in the current workflow, they are one and the same. Development occurs against a test site, and when it looks good, then we change a symbolic pointer and what was the development site becomes the released site.

Django administration certainly requires giving django write permissions on the directory it is working out of. However I don’t know if the django application callable needs write permissions. Which is the same as saying I don’t know if the interpreter needs write permissions on files in the project when it is serving websites.

Sqlite is in a file, so the interpreter probably needs write permissions for that. If we use Unix credentials to authenticate to a local postgresql database, then the pid of django must be the same as the role for the database. The interpreter will need to be able to write the socket.

There is discussion on the internet about controlling permissions of django aps. That implies that django has more encompassing permissions.

## Proposal 1

Perhaps in the future someone with a more encompassing understanding of the issues here will find a better solution than this, but here is my proposal.

The files in the development directory have the same permissions as for the release directory, and they belong to the group named after the project. For sake of discussion suppose that project is ‘customer\_gateway’.

The uwsgi process is then run with the pid customer\_gateway. It also belongs to the customer\_gateway group. In a sense, uwsgi is the customer\_gateway. Then when it runs the python interpreter that will be running as customer gateway and thus django will have access to the project directories and files.

The unix socket between nginx and uwsgi will be owned by the customer gateway but will be in group www-data. .. I’ve run into a problem with this approach. uwsgi deletes the socket if it exists when it starts, and bails with an error if it can’t delete it. It creates the socket, and then when terminating it deletes the socket. Thus uwsgi sets the socket ownership and permissions. Consequently, I see no way to make this hypothesized mixed ownership socket and then have uwsgi use it. In addition, because uwsgi must set permissions so that nginx can use the socket, and because uwsgi runs the python interpreter, the whole chain ends up with those permissions. The whole chain turns into www-data.

Here is a solution. I will create a mediary group. The mediary only exists so that nginx can speak with uwsgi, so there will be no other files in the system that might be compromised. Both nginx and wsgi will belong to the mediary group. I’ll go ahead and make the mediary a user without login. Then it will have a home directory that it owns, and all files related to mediation between nginx and uwsgi will be put there. Currently that will just be the socket I guess.