The client makes a request to Kubernetes (which spins up servers when needed with the help of NGINX as the load balancer for scalability) and gets directed to an instance of Django where the middleware runs the necessary checks (including authentication and authorization when needed). If the checks fail the correct status code (400, 401, 403) is sent back through the pipeline to the client. If the checks pass, the request moves forward to be processed.

When the request is for a security test, a job is pushed to the task queue; a message is sent back to the user to inform them that the job has been put in the queue. By sending a list of targets with the request, various tests can be done with the same request.

The processing server is constantly pulling from the queue for jobs. When there is a job to be processed, it processes it and updates the SQL data base with the results. By using Celery, tasks can be chained and share information between them if necessary.

When the request is one that does not require asynchronous tasks, Django processes it and sends the necessary response back to the client (data, etc).

Whenever the SQL database is updated with frequently accessed data, another service is triggered to replicate that data on Redis. Whenever frequently used data is requested by the client (like profile information), Django first checks Redis, if the data is not there or is expired, it then checks the SQL data base.

Two types of users are supported by using organization ID and user ID, where the users with the same organization ID belong to an organization. Some parameter is assigned to the administrator of the organization to give them more permissions than other users in the organization.