Explanation of how the system handles increased users or data.

Currently Django's native handling of multiple requests is sufficient (as seen in the performance report). Even with 100 concurrent users, the average response time is < 100 ms which we have deemed sufficient. Moreover, in the future we plan to deploy multiple VM's on Google Cloud Platform along with a load balancer to help distribute a large influx of users or data. We don't have plans to implement dynamic scaling through something like Kubernetes as we won't have that many users but in the future moving towards that instead of a hard set number of VM's is possible.

Load balancing or caching strategies if applicable

For load balancing, in the future we plan to deploy the site using GCP and to route traffic through a load balancer first. We'll have multiple instances of the backend running on VM's and let the VM decide which of the backends the request should be handled by. This will help distribute requests in the case of many being received at once.

In terms of caching, we don't currently have a caching strategy implemented. The data for receipts are unique to users and are not fetched often so a cache is not required.

Deployment strategy and uptime considerations.

Our deployment strategy involves using GCP to host the backend. We plan to configure the load balancer to handle HTTPS traffic to ensure secure connections. To maintain uptime and reliability, a /health endpoint will be added so that the backend instances can be checked for uptime moreover, with multiple instances of the backend we'll prevent a single point of failure.