**KEY FEATURES**

* Scalable: The Flask backend is designed as a stateless server, so the application can be scaled up as usage grows. The Elasticsearch data-store can also be scaled up by adding more nodes. Both of the above can also have multiple replicas for easy redundancy and high availability.
* Cloud-based application: All the layers can be containerised and deployed on any platform. For example, we can use Docker to containerise the application and use Kubernetes to orchestrate the containers on a platform like OpenShift.
* REST-based backend: This allows for separation of the front-end application development from the rest of the application. The endpoints can be used by any future developers to create applications for any platforms, without concern for the logic in the backend and datastore.
* Easily expandable: For any future functionality or events to be added in between when a donor and a done are connected and donations are received can be handled without redesign of the data-store. Only an appropriate endpoint needs to be created in the Flask backend, and frontend can also be designed to allow for this expansion with minimal extra effort.  
  The backend code is also built with simplicity in mind, each endpoint kept simple and separate to make future work easier.

**Requirement-wise Features**

* Registration process for donors and donees (NGO’s), and vetting for the latter: Apart from the standard sign-up pages, the NGO account first needs to be approved by an admin. We require the NGO to provide details like PAN number, that allows the admin to verify that the NGO is legitimate.
* Ability to have quality checks on donations: We require the donor to upload images of the items and mention the quality of it, so the NGO can ascertain the quality before accepting a donation. A two-way rating system also allows NGOs and donors to rate each other.
* Track usage on donated items: NGOs are prompted to update the donor on how their item is being used. They can upload images and send text updates.
* Display of donation requirements (populated by donee requests): Donors can view open requirements in a feed which is customised for that donor ( based on their previous donations and location )
* Display of give-away items (populated by donor goods): NGO can view all available items in a feed which is customised for that donor ( based on their previous requirements and location )
* A 2-way communication channel between donor and donee (with privacy considerations): Once a donor accepts a request for their items or an NGO accepts an offer by a donor, a chat option is enabled to allow them to communicate.

**APPLICATION USAGE**

Account Creations

The application has three types of users: NGOs (Donees), Donors, and Admins.

The admins can compare the NGO details submitted against any supporting documents submitted or by looking it up against the Income Tax website before verifying them.

Donations

At the heart of it, the application aims to connect potential donors who have items to donate with NGOs whose requirements can be fulfilled by those items. From the back-end, then, we are dealing with two main data items: requirements and items. Further, as donors and NGOs connect with each other and the donations take place, we have another data item: update, which represents all the events that take place during their interactions.

NGOs create requirements in the application, specifying what they need and how much of it. Donors create items in the application, specifying the details of the items they are willing to donate. The details they fill in includes a declaration of the item quality as well as images, to allow NGOs to verify if its usable.

When a donor logs in, he views a feed of the requirements of NGOs on the platform. This feed is customized to the donor based on the location and previous activity. An NGO also views a similarly customized feed of items available for donation. Our use of Elasticsearch allows for such matching without complex code, because the data-store has such capabilities out-of-the-box.

The interactions between NGO and donor can be initiated by either party: an NGO can view items posted by a donor and request for it to be donated or a donor can see a requirement of an NGO and offer to donate to it. For large quantities for items, the application admins can define a limit as to how much can be requested by one NGO, to allow everyone a fair chance at benefitting from the platform. This limit can be specified on a category-basis.

We recognized that after the initial initiation (donor offering or NGO requesting), the events that take place are the same in either case: the items are to be donated. So rather than keep two entirely separate flows, we aimed to simplify the application by unifying both flows. We do this by creating a matching requirement when an NGO requests an item, and a matching item when a donor offers to donate towards a requirement. Once the other party accepts the request/offer, the rest of the flow is the same. This simplification of the logic eliminates duplication of code and better expandability.

Once an NGO accepts a donor’s offer or a donor accepts an NGO’s request, a two-way messaging feature is activated. This allows them to communicate with each other. The donor’s identity is not disclosed here.

When an NGO receives the donated items, he marks them as received in the application. At this point, our alerting mechanism comes into play, prompting the NGO to update donor about how the donation is being used ( the exact duration between receipt and this alert is customisable ). The NGO can make use of the 2-way communication channel to update the donor about usage.

The NGO and the donor are also prompted to rate each other. The application maintains the average ratings for both parties.