Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

A Web Portal for Linking Charities to Donors Worldwide

Author:
Dongxiao Huang

Supervisor:
Anandha Gopalan

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science of Imperial College London

September 2018

Abstract

Recent years have seen a fall in global generosity. Charities find it difficult to fundraise, particularly those which are small-scale or geographically limited. To break the boundary of geography, technology plays an increasingly significant role in philanthropy. However, there are currently almost no online portals that serve international donations for charities on the Internet.

To solve this problem, this project aimed to build up a transparent and trusted community for charities and users worldwide, so that donors can give monetary donations or offer volunteer work easily and with confidence. In addition, the feedback feature allows users to make direct comments and ratings toward charities through this portal. In order to make donors give with confidence, charities registered in England and Wales are able to get verified through this portal thanks to the data provided by *the Charity Commission*[1].

This report illustrates the architecture, implementation, and evaluation of the web application. Angular and Bootstrap4 were utilized to create the client-side, which communicates with the Express server by RESTful APIs. The Database used in this project was MongoDB. After development, this web application was deployed on Azure.

It is now accessible for users worldwide, to promote the act of giving across borders. Feedback from several users makes a great contribution in developing this project.

[1]A United Kingdom government organization that registers and regulates charities in England and Wales. Source: https://www.gov.uk/government/organisations/charity-commission. Accessed: 2018-09-02

Acknowledgments

I really appreciate the following people who have supported me in this project.

- My supervisor, Anandha Gopalan. He is such an excellent supervisor who has helped me a lot in the technical project-related problems and inspired me hugely through our talking. He is considerate and really cares about his students; after each and every one of our weekly meetings, I always feel encouraged.
- My great friend, Lillian Liu, who kindly and unselfishly helped to check the language mistakes of this report, which moved me a lot as always.
- My parents and sister, for their support and care.
- My friend, Yufeng Zhang, for his insightful suggestions in this project.
- My trustworthy friend, Lin Li. Through conversations with him, I can always be inspired.
- People who took the survey I published, for their kind and valuable feedback.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

According to the CAF World Giving Index 2017 [11], philanthropy trends have shown a fall since last year and less generous behaviors were displayed throughout the world except for the continent of Africa. The country with the highest share of donors was Myanmar in 2017, whilst all western wealthy countries in the top 20 had fewer scores in the index compared with that of 2016. This report not only challenges the conventional stereotype of the link between abundance and philanthropy, but also reveals the fact that actions should be taken to cultivate the habits of giving.

The index also shows that the score of volunteering time is worst for almost all countries in the top 20 compared with two other behaviors of giving: helping a stranger and donating money. As a part of civil society, people should be encouraged to serve the community and participant in voluntary activities. A variety of factors may result in this fact, such as the pressures of modern life and limited free time of urban citizens, but what we would never like to see is the limitation of spread for voluntary work information. To take advantage of today's technology, it is of great convenience for the non-profits to launch an activity and spread it through social media or their websites.

To small-scale charities with low budgets or those from emerging economies which lack IT specialists to construct websites, it is tough for them to promote their values and principles as well as raise money from society, let alone money from abroad. This is why underdeveloped countries see a decrease in the number of regional charities.

In brief, individuals are less generous in giving monetary or voluntary contributions and poor charities have difficulties in promoting themselves to the public. Difficult as it may seem, it would be beneficial to build a portal that can link charities universally. For charitable institutions, they can register their charities on the portal and launch activities or raise money. For donors, they can make donations through the portal and give feedback. In addition, all donated money can be tracked. Donors are

not only able to make monetary or material contributions, but they can also deliver voluntary services.

1.2 Objectives

This project aims to build an online portal that connects charities with the world's users and should allow the user to achieve the following goals:

- Charities, NGOs or Non-Profit Organisations can register themselves on this website. The database [1] provided by the Charity Commission in the UK has information about UK charities which can be used to ensure a charities' genuineness.
- The user should be able to search for charities (either by name, categories or location). It is not necessary for users to have a profile on the system.
- The potential users surfing the website can make a donation without registration.
- The user should be able to rank charities so that other users can see that the charity does good work and also possibly be able to track what happened to the donations.
- The user should be able to advertise the fact that they have donated through this portal - such as through Social Media/email.

1.3 Contributions

Considering all the specifications listed above, this project builds a platform for users in order to help associate donors with charities all over the world without the limitation of geography. As a small step, I hope it can make a contribution to promoting the act of giving by both donation and delivery of voluntary services.

In addition to the contributions above, this platform is also equipped with some inventive features. The most significant is the voluntary section which enables users to register to activities through this portal. Amongst all the related websites researched, not a single one fully implements this functionality.

Furthermore, users can also find charitable organizations on the map and check their profiles. To enable donors to give feedback, the comments section is created. Once users find loved charities, they can add them to favorites for a quick find. It seems that similar web pages focus more on the charity profile display and donation feature, but no user comments were allowed.

[1] Data available on site: http://data.charitycommission.gov.uk/. Accessed: 2018-09-02

This project is a forward step in philanthropy. It provides easy registration for charities all over the world. Similar products are mainly focused on charities within an area of a country. International donations are difficult through these portals.

Chapter 2

Background

2.1 Existing Relevant Websites

Websites offering similar functionalities are few and far between, but there are still some that serve partial demands and all of them have advantages and disadvantages of their own. In this section, each web page will be analyzed and all of them provide great experience for this project.

2.1.1 Charity Navigator

Charity navigator[1] is a United States based organization aimed at finding a charity you can trust. It is the largest and most-utilized charity evaluator in America. It serves charities only in America.

Advantages	Disadvantages
It enables users to search charities by charity name or other criteria.	Its user interface is too simple and not user-friendly.
It enables users to donate directly to charities.	No volunteer activities service are provided.
It allows users to register, to track liked charities and to rank charities.	It only contains charities within American and cannot enable charities or non-profits to register on this website.
It allows people to quickly donate without registration.	Feedback from users is not available.

Table 2.1: Analyses of Charity Navigator Website

[1]Source: https://www.charitynavigator.org. Accessed: 2018-09-02

2.1.2 Charities Aid Foundation

Charities Aid Foundation[2] is an organization based in the United Kingdom and aims for better giving, and for over 90 years it has been helping donors, companies and charities to make a bigger impact.
Through its website, analyses are given below:

Advantages	It is a global organization based in U.K.
	It allows donation to UK charities but not all.
	It allows users or charities to register.
	It provides a good search tool.
Disadvantages	It does not contain any charities outside of the United Kingdom.
	Donation is not convenient because personal information must be provided in this website if not registered.
	It cannot get feedback from donors.
	No charity volunteer activity information is provided.

Table 2.2: Analyses of Charities Aid Foundation Website

2.1.3 Global Giving

Global giving [3] is the largest global crowdfunding community which connects nonprofits, donors and companies in nearly all countries. It helps nonprofits get access to what they need and makes the world a better place.

Advantages	It lists philanthropic projects from all over the world and all of them can be searched by categories or countries.
	This web site has user-friendly UI design.
	Monetary donations can be transfered directly to projects.
	It enables charities to register and post projects.
	Users can add projects to favorites.
	Projects are introduced with descriptions, images and videos which are rich in forms.
Disadvantages	No user feedback of projects is available.
	No voluntary section is provided, and only monetary donation is available.

Table 2.3: Analyses of Global Giving Website

[2]Source: https://www.cafonline.org. Accessed: 2018-09-02
[3]Source: www.globalgiving.org. Accessed: 2018-09-02

2.1.4 Charity Needs Foundation

Charity Needs Foundation[4] is a website set up to promote public awareness of voluntary charities worldwide. Non-profits all over the world can register on this site to profile their organizations. This website has some similar ideas to this project, with some strengths and weaknesses.

Advantages	It contains multinational organizations.
	It enables organizations to register.
	It is able to provide a voluntary section.
Disadvantages	Website is still under construction and the UI is not responsible.
	No charities' profiles are provided on this website. All the information about a charity will be redirected to the charity's website, which will cause a problem if the charity is small and does not have its own website.
	Database of charities is still small, and the number of charities listed is 273 (on 19th, August 2018). No pagination is used in this list, so it may cause delay when loading the website.

Table 2.4: Analyses of Charity Needs Foundation Website

2.1.5 The Charity Commission

The Charity Commission [5] for England and Wales is a charities regulation platform in the UK. As an official website for charities management in UK, it provides public charities data via browsing the website or downloading the entire database. It is not user-friendly to find a charity on this site. Besides, when data is downloaded, conversion is needed using Python Script provided which is quite an obstacle for users with no computing experience. Analyses of this website are in Table 2.5.

[4]Source: http://www.charityneeds.com. Accessed: 2018-09-02

[5]Source: https://www.gov.uk/government/organisations/charity-commission. Accessed: 2018-09-02

Advantages	The information about charities in England and Wales is complete and comprehensive.
	It is an official charity management platform, which is trustworthy.
Disadvantages	The search functionality is not user-friendly. In the normal search, it can only search by registration number. In the advanced search, the search is complex and hard to use.
	Users can only check the charity details on this website. No other operations are supported, such as donation behaviors.

Table 2.5: Analyses of the Charity Commission Website

2.1.6 Just Giving

Just giving[6] is a fundraising platform, which enables charities or individuals to receive donations from people all over the world.

Advantages	Great user interface.
	It enables charities and users to raise money from people all over the world. Making Donations is quite convenient by means of a card or PayPal.
	The profile of the project contains different sources of media such as images, text and videos.
	Fundraisers can keep the donors updated on the project profile page.
	Comments and share on social media are available for projects.
	Recommendations are great for users.
Disadvantages	Only monetary donations are supported on this website - it would be better to add voluntary work features.

Table 2.6: Analyses of Just Giving Website

2.2 Summary

In summary, web applications available on the Internet all possess positives and negatives. Almost all of them can provide good charity search and donation functionalities and some enable both users and charities to register. However, nearly none

[6]Source: https://www.justgiving.com. Accessed:2018-09-02

of them enable global charities to register except for *Global Giving* and *Just giving*. Not a single one supports users to do volunteer services and users cannot leave comments for the charity except for *Just giving*. In fact, all websites are excellent in certain aspects but not well-round, so our project will overcome their weaknesses and acquire their strong points.

In addition to registration, donation, search system and feedback function, geographical searching will be implemented, which enables users to search on the map. The voluntary section helps charities to find volunteers in an easier way, which is an original section for this kind of portal. The dataset containing information about charities in England and Wales from *the Charity Commission* is utilized in this project to verify charity users.

Chapter 3

System Design

This chapter begins with web application design alternatives from a software perspective and reasons for the choices that are used in this project. Following this, details for system design will be illustrated. Explanation starts with the whole system architecture of the project, presenting the relationships between client-side, server-side and database, and then presents the association of collections inside the database. Finally, all the REST APIs[1] created for client-server communications will be listed and explained.

For a better understand the system design strategies, user journey will be listed in section 3.2.

3.1 Design Choices

As a web application, it is composed of three main parts, namely front end, back end and database. The front-end, or the client-side, is responsible for rendering the user interface to communicate data with the server, using the language of HTML, CSS and JavaScript typically. The back-end, or the server-side, serves to make a system work properly. It operates data retrieved from client-side behind the scenes and provides interfaces of a dynamic site. A dynamic site [7] is different from a static site which is constant and unchangeable in content. It can dynamically generate the changing content which can update in real-time [16].To achieve this, all data should be stored in a database. The server-side will provide interfaces for client-side to communicate with a database.

After implementation of the system, it should be deployed into a cloud server such that the website can be accessed by users on the Internet.

Each design choice will be detailed in this section.

[1] an application program interface (API) that use HTTP requests to GET, PUT, POST and DELETE data.

3.1.1 Front-end

To build up the front-end, language of HTML, CSS and JavaScript are used. Those three core languages are the cornerstone technologies of the World Wide Web. But in order to save time and be compatible with different web browsers, We can make use of frameworks for client-side [10].

There are two types of front-end frameworks that are easy to use, namely CSS front-end frameworks and JavaScript front-end framework.

UI design choice The user interface design strategy is the single page application (SPA). Traditionally, the user requests the server to fetch a new HTML via full server roundtrip, while an SPA rerenders its content without reloading the entire page. An SPA primarily fits its contents in one page and redraw parts of the page in response to navigation operations.

In this SPA, the most important functionalities are searching for charities and making donations. So I put the search bar on the header of the main page, so that the user can search easily. There are five main navigation tabs in this web. They are the home page, discovery page, volunteer page, about us page and contact us page. Users can easily navigate between them.

In the *home* page, a carousel (slideshow) of cover images will first appear which are designed to promote the values and work of the webpage such as donation and voluntary work. Down below it, eight newest charities' profiles with basic information are displayed. This is mainly made for promoting the new charities created on this website so that they can easily be found by the potential donors.

In the *discovery* page, all the charities registered on this web page can be discovered. Users can find out about charities here. For more specific searches, users can make use of the search bar.

The *volunteer* page is the portal to find all the available voluntary services posted by charities. In the *about us* page, users can find more information about the website. To contact, user can visit *contact us* page

In this project, a lot of icon fonts are used. Icon fonts are sets of symbols that are available to be styled with CSS like regular fonts and often used as a replacement for graphs. Lots of icon sets are on the Internet - Font Awesome[2] is used in this project for its performance and high-quality iconographies.

I have also referred to Bootsnipp[3], an element gallery for web designers, for some fancy UI design styles and structures in this website.

[2]Source: https://fontawesome.com. Accessed: 2018-09-02
[3]Source: https://bootsnipp.com/. Accessed: 2018-09-02

Pictures used for cover images are retrieved from Pixabay[4], a free images sharing website. The promotional images used of charities created for testing are all from the real charities' websites.

More specific UI design strategies of different pages are detailed in Chapter 4 in the corresponding sections.

CSS Front-end Frameworks CSS front-end framework is a collection of ready-to-use HTML, CSS and JavaScript templates for UI components, which make it easier for developers to add styles for each component.

Nowadays the access of web-pages is no longer limited to personal computers and it can be from different screen sizes. Thus responsible web design is of great significance. A responsive web is a web that can adapt its contents and styles to different devices and windows or screen sizes. The CSS front-end framework makes cross-browser compatibility easier and improves productivity in developing web applications. In addition, it provides much support with tutorials on the Internet.

Deluges of CSS front-end frameworks are available nowadays, and some of the most popular are Bootstrap4[5], Foundation[6] and Semantic UI[7]. In this project, Bootstrap4 is used because it is undisputedly the most popular in the front-end world. It provides well-styled components and layouts, which speed up workflow and increase productivity.

Most of the Bootstrap4 components are compatible with Angular, but some fancy ones requiring jQuery[8], a JavaScript Library, are not. Instead, there is a Bootstrap widget designed for the Angular ecosystem, *ng-bootstrap* [9] which largely improves the productivity.

JavaScript Frameworks A JavaScript Framework is collections of JavaScript code that form a framework, which is distinct from JavaScript libraries in control flow. It does not provide functions to be called in the program, whilst it defines the web application design pattern. Applications written using a JavaScript framework will be called and used in the internal way of the framework.

It is good practice to use JavaScript frameworks because it defines a single page web application and uses the structure of *Model-View-Controller* (MVC) */ Model-View-*

[4]Source: https://pixabay.com/. Accessed: 2018-09-02
[5]Source: http://getbootstrap.com. Accessed: 2018-09-02
[6]Source: https://foundation.zurb.com. Accessed: 2018-09-02
[7]Source: https://semantic-ui.com. Accessed: 2018-09-02
[8]Source: https://jquery.com. Accessed: 2018-09-02
[9]Source: https://ng-bootstrap.github.io/#/home. Accessed: 2018-09-02

ViewModel (MVVM) / *Model-View-Whatever* which enables Data binding and routing. In addition, it supports scalable, reusable, maintainable JavaScript code and test-driven development.

JavaScript Frameworks is used as a tool to speed up the implementation and improve productivity and efficiency. Frameworks such as *React*[10], *Angular*[11] and *Vue*[12] are several top solutions.

This project chooses Angular as JavaScript framework, as it is built with all the tools a modern JavaScript developer needs and it uses a component-based architecture, which consists of Modulars, Component-based with Templates and Services [9].

The Architecture of *Angular* framework is shown in Figure 3.1. Its structure is hierarchical and modularized. Each *Angular* application has its own root module and is composed of different modules, self-created or third-party. Each module is made up of components and services, and a component can insert other components and inject services. A component contains two types of file, namely template and typescript files, as Figure 3.2 shows.

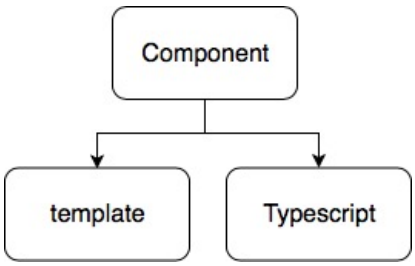


Figure 3.1: Angular Modular Architecture

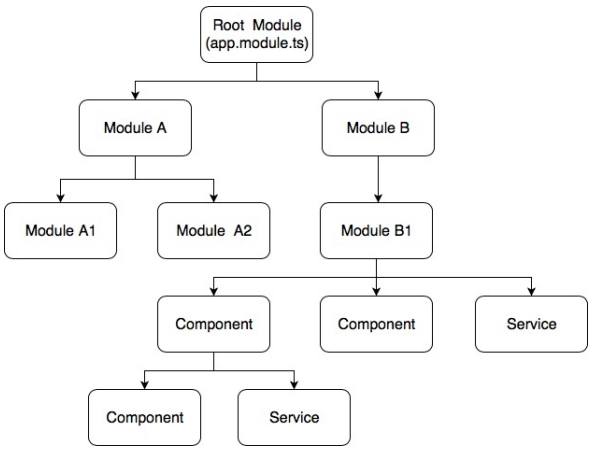


Figure 3.2: Angular Component Architecture

The angular framework uses the design pattern of a variant of MVVM. Typical pattern for MVVM is in Figure 3.3.

Typescript class A model in *Angular* is *Typescript* class or interface, which is utilized to define the structure of an object. Classes or interfaces defined can be used in any component's TypeScript file once being imported.

[10]Source: https://reactjs.org. Accessed: 2018-09-02
[11]Source: https://angular.io. Accessed: 2018-09-02
[12]Source: https://vuejs.org. Accessed: 2018-09-02

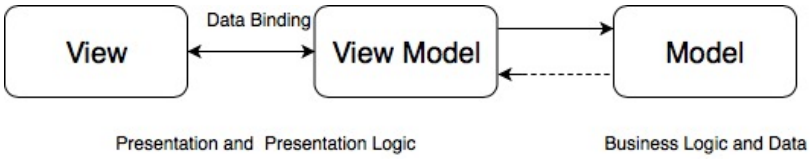


Figure 3.3: Model-View-ViewModel

The view is the presentation layer of the application, and it is defined as a *template* in *Angular*, which is written in HTML, CSS and some *Angular* specific elements like other components.

Template ViewModel in Angular is the *template* part of a *component*, and it is comprised of behaviors (methods) and properties (attributes) of the application domain. ViewModel and View are connected by data binding [13].
The specific data binding methods in Angular are displayed in Figure 3.4, and View

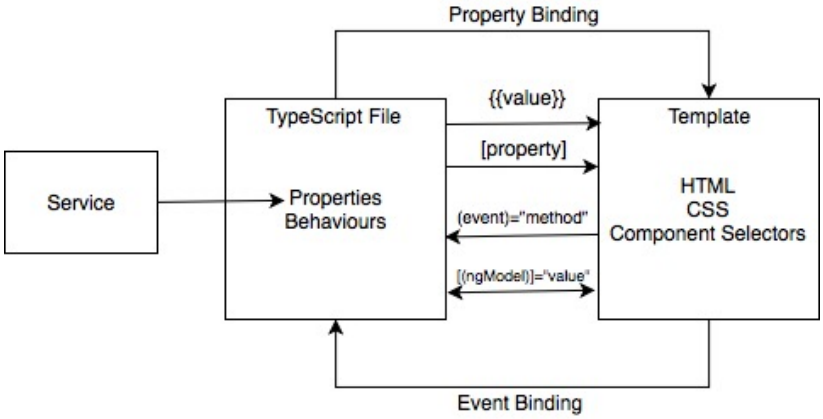


Figure 3.4: Data Binding in Angular

(*template*) and ViewModel (*TypeScript* file) contain three types of data binding.

- Property binding: Using the properties defined in *TypeScript* file into a template file, and the properties should be embraced by curly brackets.
- Event binding: Users interact with DOM elements and trigger the event defined in the *TypeScript* file. The event should be embraced by round brackets$()$, like $(click)$.
- Mutual binding: It enables a property mutually controlled in both *TypeScript* file and *template*, and it has its special syntax $[(ngModel)]$. It is especially useful when binding a form value, which is initialized in *TypeScript* file and then changed by a user.

Angular Service Another important element in both a *module* and a *component* is Angular *services*. As shown in Figure 3.4, services can manipulate properties in a

TypeScript file. This serves to factor out application logic and keep the component to be lean. Client-server communication operations, such as fetching data and uploading data, are usually delegated to *services* so that such services can be used in different *components* once being injected.

3.1.2 Back-end

On back-end or server-side, there are various technologies and approaches like *PHP*[13], *Java*[14], *Python* and *Node.js*[15].

As there is a trend now to use JavaScript in both front-end and back-end, I decided to use Node.JS to implement.

Node.js is asynchronous event-driven JavaScript runtime environment that allows you to write server-side applications in JavaScript. It excels at a scalable server as nearly no function in *Node.js* directly performs I/O, so it prevents blocks of the process.

Using *Node.js* for a server will be suitable for this project because as the number of users grows, scalability is a big factor in production, *Node.js* performs well in this part. Besides, *Node.js* as server language will improve learning efficiency because, in both front-end and server-end, the same language is used [14].

A back-end framework is a software framework that helps developers to build up a server easily by simplifying the operations between server and database, routing URLs to suitable handlers, setting responses and so on. Back-end frameworks are not necessarily needed, but it saves a lot of time so this project leverages it [3].

Amongst all the back-end framework using *Node.js*, *Express* is undoubtedly the leader[16]. It provides a robust set of features and has much third-party middleware to extend functions.

3.1.3 Database

A database is an organized and systematic collection of data. It is used for storage and data operations such as query, insert, update and delete. Those manipulations make data is easily managed [4].

Structured Query Language (SQL) is actually a standardized query language for Relational Databases. So Relational Databases are also called SQL Database. A Relational Database is good in many aspects but strict table schemata is a disadvantage

[13]Source: http://php.net. Accessed: 2018-09-02
[14]Source: https://www.java.com/en. Accessed: 2018-09-02
[15]Source: https://nodejs.org/en/. Accessed: 2018-09-02
[16]Source: https://stateofjs.com/2017/back-end/results. Accessed: 2018-09-02

and the system starts becoming slow in terms of response time when large data is stored.

Recent years have seen a growing demand for NoSQL databases, which are non-relational databases. To deal with the rigid schemas of SQL databases, NoSQL databases provide a mechanism that stores and operates data modeled in means, which increase flexibility and operational speed [17].

As NoSQL databases scale out better than relational databases and are designed with web applications in mind [5], this project chooses to use NoSQL to store data. That has advantages including scalability , availability, consistency and partition tolerance. In addition, it is easy to deploy and no object-relation mapping is required.

There are four main categories of NoSQL databases, namely:

- Document databases (e.g., *MongoDB*[17])
- Key-value databases (e.g., *Redis*[18])
- Column-family databases (e.g., *Cassandra*[19])
- Graph databases (e.g., *Neo4J*[20])

Specifically, I chose to use *MongoDB*, because it stores the documents in BSON (Binary JSON) format which is easily used for client-server communication.

3.1.4 Client-Server Communication

Except for static web pages, all dynamic websites are not stand-alone. They need to communicate with the server and retrieve dynamic or personalized data for web users, shown in Figure 3.5.

The communication in this project is built by HTTP with REST APIs. There are a set of common methods for HTTP, among which GET/POST/PUT/DELETE ways are most used. REST is commonly associated with HTTP, and it is an architectural style of designing URIs. In REST APIs, it explicitly uses HTTP methods and transfers data in formats like JSON or XML.

To communicate the client with the server, at first the server provides REST APIs that are available to use. Then the client requests the APIs using corresponding HTTP methods.

[17]Source: https://www.mongodb.com. Accessed: 2018-09-02
[18]Source: https://redis.io. Accessed: 2018-09-02
[19]Source: http://cassandra.apache.org. Accessed: 2018-09-02
[20]Source: https://neo4j.com. Accessed: 2018-09-02

Specifically, in the web application using the Angular framework, it supports communication with servers using the HTTP protocol via *HTTP Client*[21]. Delegating data access to a supporting service class of component should make use of *HTTP Client* service which supports functions as listed in Table 3.1.

	http.get()	http.post()	http.put()	http.delete()
http: *HttpClient*	Request API using GET method	Request API using POST method	Request API using PUT method	Request API using DELETE method

Table 3.1: Main Supported Methods by Http Client

3.1.5 Server-Database communication

The server provides RESTful APIs and performs operations on data. Every HTTP request is interpreted into a query, update, create or delete database operation.

To perform the mapping, *MongoDB* and *Mongoose* packages are used. *Mongoose*[22] is a MongoDB ODM(Object Document Mapping) which provides a solution to modeling document data. *Mongoose* adds structure to *MongoDB* documents via *Schema* and then Schema object is used to create a *Model* function. Model functions provide CRUD operations on corresponding MongoDB collections.

3.2 User Journey

After deciding the tools to implement the portal, another necessity is the functionalities that the platform serves according to user journeys. There are three types of users. Web visitors who are not registered in this portal is un-authorized users, another two are donor users and charity users. Different users can have different user journeys.

Un-authorized Users The journey of un-authorized users in this portal are listed below:

- Search charities by name/location/category
- Find a charity nearby/on map
- Find all and search voluntary activities
- Make donations and share on social media
- See all non-profits' profiles and share on social media

[21] Source: https://angular.io/guide/http. Accessed: 2018-09-02
[22] Source: https://mongoosejs.com. Accessed: 2018-09-02

Authorized Donor Users Apart from all the experience of unauthorized users, donor users can also experience the other features in this portal:

- Authentication via an account or Facebook
- Signs up for and manages voluntary work
- Rates/comments/likes charities
- Manages its profile
- Checks donation histories

Authorized Charity Users Charity users like donors can have extra experience in addition to those of unauthorized users'.

- Creates a charity account and manages its profile
- Charity verification
- Voluntary work management such as creation/modification and check.
- Check donation histories
- Check voluntary work participants

3.3 System Architecture

Three layers structure of web development are adopted in this project.

- Presentation logic: The UI which displays data to the user and accepts input from the user. In a web application, this is the part which receives the HTTP request and returns the HTML response.
- Business logic: It handles data validation, business rules, and task-specific behavior.
- Data Access logic: It communicates with the database by constructing SQL queries and executing them via the relevant APIs.

In this project, the methodologies and techniques used for each layer are shown in Table 3.2. The web is developed from three parts, and for each part using different frameworks, but overall, all of them are based on JavaScript and can be considered as full-stack JavaScript development.

For the presentation layer and data access layer, *Bootstrap4*, as well as *Material* Module provided by *Angular*, are used to design the user interface and *Angular* framework is applied as the JavaScript framework.

UI Framework: Bootstrap4	JavaScript Framework: Angular	Express	MongoDB
JS, CSS, HTML		BaaS NodeJS Modules NodeJS	NoSQL
Presentation Layer		**Business Logic Layer**	**Data Access Layer**

Table 3.2: Full Stack Web Development Strategy in Technical Implement

For Business Logic Layer, *Express* Generator scaffolded out an *Express* server and *MongoDB* database was used in this project.

Apart from the technical strategy used in each layer, specific implementations for the purpose of this project are displayed in Figure 3.5. It can be roughly divided into 5 sections in the client-side, authentication section, voluntary section, search section, feedback section and donation section. Each section carried out different functionalities, and those sections will be detailed in Chapter 4. In the server-side, 13 routers are established, because 2 of them are auxiliary, so only 11 of them are presented in Figure 3.5. Each router has several endpoints to help interact with data resources. All the routers utilized are listed in the following section REST APIs. As for the database, *MongoDB* is used to store data collections and the images uploaded are stored in the Azure Storage Account. 11 collections are created in the *MongoDB* database and 3 types of images, charity promotional images, user profile images and cover pictures are saved in Storage Account. A more specific model of the database is described in the following section Database ER Diagram.

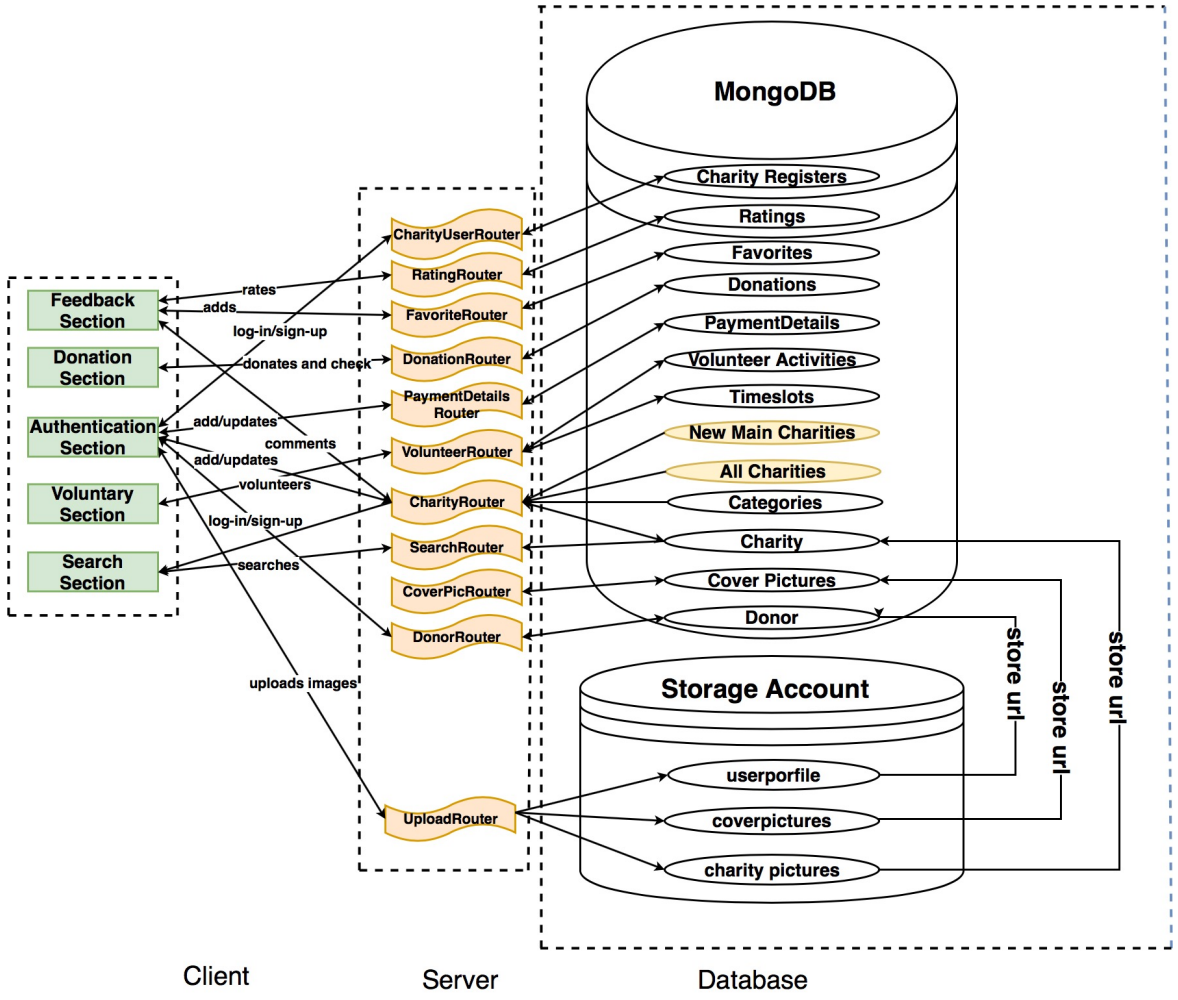


Figure 3.5: System Architecture. Three layered structure which is composed of presentation logic layer, business logic layer and data access logic layer.

3.4 Architecture of Front-end in Angular

The client-side was built using Angular framework which is made up of different components. As shown in Figure 3.6, the root module is *app.module.ts* and it has one component,*AppComponent*, which consists of three parts, namely *HeaderComponnet*, *FooterComponent* and *AppRoutingModule*.

As a SPA, all pages use the same footer (implemented as *FooterComponent*) and header (implemented as *HeaderComponnet*), only the contents of pages are different. The content is controlled using different routers in the client-side. All the routers of the client-side are listed in the Appendix E. There are 20 routers in total for the front-end, and each router is bound with a component which often made up with several sub-components. Only 13 main components, which are responsible for dynamic pages, are depicted in this figure. All of them are dependencies of the *AppRoutingModule*. In Chapter 4, the details of the components will be explained in the corresponding sections.

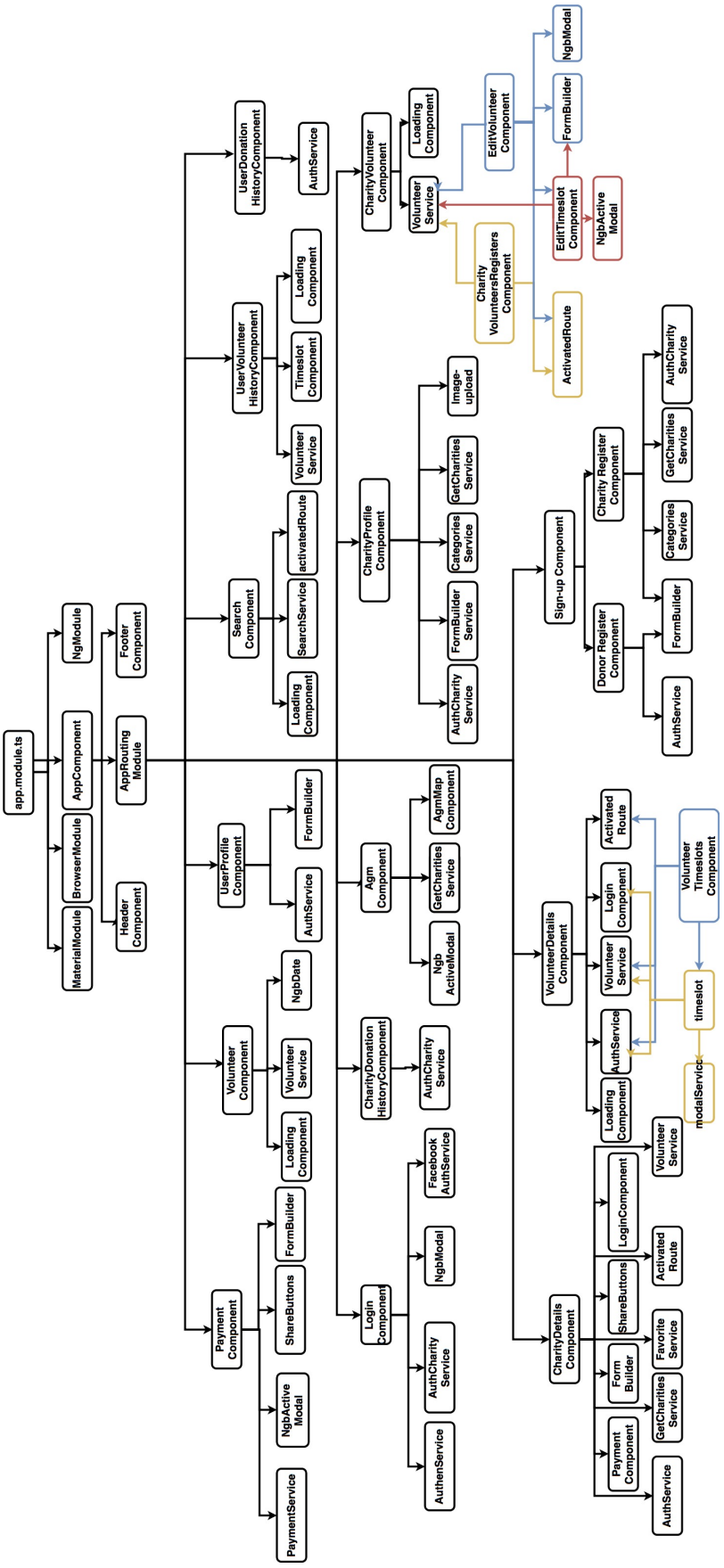


Figure 3.6: Angular Application Architecture

3.5 RESTful APIs

In this section, all routers created in the server will be briefly explained and two or more key endpoints are listed for each router. All the endpoints and their explanations are listed in Appendix C. For some important endpoints, specific definitions will be given in related part in Chapter 4 and Appendix D.

Categories Router Each charity belongs to one or more categories. Categories are managed through this router. Operations on the categories collection used in this project are all handled by APIs listed in Table 3.3.

Resource	GET read	POST create	PUT update	DELETE
/categories	Return all the categories for charities	Create new category for charities	Method not allowed (405)	Remove all the categories for charities
/categories/:categoryId	Return a category with id:categoryId	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table 3.3: Supported APIs for Categories Router

Charity Router Charity Router is one of the most significant routers in this project. It is used for charity search, charity discovery, charity feedback and so on. In the home page, the recommended charities shown are also retrieved from this router. When a user would like to make comments, an API in this router is used. Besides, geographical data for each charity which is important information used for geographical search is also operated by this router. Table 3.4 list part of endpoints in this router.

Resource	GET read	POST create	PUT update	DELETE
/charities	Return JSON format data, containing information like current page, total pages, and a list of charities in current page	Create new charity	Method not allowed (405)	Remove all charities

/charities /:charityId /-geocoding	Return the geocode of the charity with id: charityId	Create the geocode of the charity with id: charityId	Update the geocode of the charity with id: charityId	Method not allowed (405)
/charities /:charityId/ -comments	Return the comments of the charity with id: charityId	Post comment to the charity with id: charityId	Method not allowed (405)	Delete all the comments of the charity with id: charityId

Table 3.4: Partial Supported APIs for Charity Router

User Router User router serves as authentication for donor users. It enables donors to sign up, log in and log out of this site. In addition, it also offers APIs to check JSON web token which is assigned to authenticate a user and is stored in local storage in the user's browser. This site will check the JSON web token to keep user login status alive. In order to support Facebook third-party authentication, there is an API to uphold it. During the registration period, username's existence will be checked through this router. As for profile and password management, it is supported by the /users/profile and /users/newpassword endpoints separately. In Table 3.5, part of the available APIs in this router are listed.

Resource	GET read	POST create	PUT update	DELETE
/users/signup	Method not allowed (405)	Create a new donor account	Method not allowed (405)	Method not allowed (405)
/users/login	Method not allowed (405)	Log in with an account	Method not allowed (405)	Method not allowed (405)
/users /check-JWTToken	Check JSON Web Token is valid or not	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/users /facebook /token	Sign in using Facebook Authentication	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table 3.5: Partial Supported APIs for Donor Router

Charity Users Router Charity Users Router bears almost the same features as User Router, except that it does not support Facebook authentication and all the charity users' data is stored in a different collection from donor users' data. Similarly, partial

APIs in this router are in Table 3.6.

Resource	GET read	POST create	PUT update	DELETE
/charityusers /signup	Method not allowed (405)	Create a new charity account	Method not allowed (405)	Method not allowed (405)
/charityusers /login	Method not allowed (405)	Log in with an charity account	Method not allowed (405)	Method not allowed (405)
/charityusers /check-JWTToken	Check JSON Web Token is valid or not	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table 3.6: Partial Supported APIs for Charity Users Router

Cover Picture Router On the home page, there are cover images in the jumbotron sector. Each cover image is stored in MongoDB in the format of JSON, consisting of URL, title and description. To change or update these cover pictures in the database, operations can be conducted through APIs listed in Table 3.7.

Resource	GET read	POST create	PUT update	DELETE
/coverpics	Return all cover pictures information	Create a new cover picture information	Method not allowed (405)	Delete all cover pictures information
/coverpics/:coverId	Return the cover pictures information (id=coverId)	Method not allowed (405)	Update the cover pictures information (id=coverId)	Delete the cover pictures information (id=coverId)

Table 3.7: Supported APIs for Cover Picture Router

Donation Router Donation router makes use of Stripe API[23] for donations. Once donors make donations and charities have received monetary contributions, transaction history will be saved. To query such documents, APIs in Table 3.8 are available to enable registered donors to find out all donations they made and charity users to check donation histories that they received.

[23] A payment gateway for Internet commerce:https://stripe.com/docs/api

Resource	GET read	POST create	PUT update	DELETE
/donation	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of donation history for authenticated charity user in current page	Make a payment through Stripe API	Method not allowed (405)	Method not allowed (405)
/donation/user	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of donation history for authenticated donor user in current page	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table 3.8: Supported APIs for Donation Router

Rating Router Making a rating for charities is one way for charities to receive feedback from users. Each registered user can rate all charities, but only once for each charity. An average rate will be calculated for each charity. Table 3.9 lists all endpoints in this router.

Resource	GET read	POST create	PUT update	DELETE
/rating /:charityId	Return the rating for charity (id = charityId) made by an authenticated donor if any.	Make a rating for charity with id: charityId from an authenticated donor user	Method not allowed (405)	Method not allowed (405)
/rating/ :charityId /averageRating	Return the average rating for the charity (id = charityId)	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table 3.9: Supported APIs for Rating Router

Favorite Router This router is used for donors to add charities to their favorite lists, remove them and also check if a charity is in one's favorites list or not. Table 3.10 contains all endpoints under this router.

Resource	GET read	POST create	PUT update	DELETE
/favorites	Return all the favorite charities for a authenticated donor	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/favorites /:charityID	Return if the charity with id: charityID is one of favorites for the authenticated user or not	Add the charity with id: charityID to one of favorites for the authenticated user	Method not allowed (405)	Remove the charity with id: charityID from one of favorites for the authenticated user

Table 3.10: Supported APIs for Favorite Router

Upload Router This router is used mainly for image uploads for both charity users and donor users. See Table 3.11 for complete APIs.

3.6 Upload Router

Resource	GET read	POST create	PUT update	DELETE
/imageUpload	Method not allowed (405)	Post a profile image for a authenticated donor	Method not allowed (405)	Method not allowed (405)
/imageUpload /charitiesPics/	Method not allowed (405)	Post at most 3 promotional images and return images URLs	Method not allowed (405)	Method not allowed (405)
/imageUpload /charitiesPics /:chairtyId	Method not allowed (405)	Post at most 3 promotional images when charity registration	Method not allowed (405)	Method not allowed (405)

Table 3.11: Supported APIs for Upload Router

Payment Details Router This is an auxiliary router to search, create, update and delete payment details for each charity. References of payment details will be stored

in corresponding charity collection. Table 3.12 displays its two endpoints.

Resource	GET read	POST create	PUT update	DELETE
/paymentdetails	Method not allowed (405)	Create a new payment detail information	Method not allowed (405)	Method not allowed (405)
/paymentdetails /:cardId	Return the payment detail with id:cardId	Update the payment detail with id:cardId	Method not allowed (405)	Method not allowed (405)

Table 3.12: Supported APIs for Payment Details Router

Search Router Search Router only provides an API, and searching charities by location, name or categories are all through it. Results return is not simply for entire charities that hit the search key, instead, it returns paginated JSON data, which consists of searched results and pagination details. More details are presented in Table 3.13.

Resource	GET read	POST create	PUT update	DELETE
/search	Return JSON format data, containing information like current page, total pages, number of items in each page, and a list of charities searched by keywords in current page	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table 3.13: Supported APIs for Search Router

Volunteer Router Volunteer router is one of the most sophisticated routers in this project. It handles all jobs relevant to voluntary activities, such as activities sign-up, query, creation and management for both participants and organizers. Only three endpoints are presented in Table 3.14. Check Table C.12 in Appendix C to gain more specific information on APIs in this router.

Resource	GET read	POST create	PUT update	DELETE
/volunteer	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of voluntary activities that is available from today or on certain day in current page	Create a new voluntary activity	Method not allowed (405)	Method not allowed (405)
/volunteer/:volunteerId	Return the voluntary activity identified by volunteerId	Method not allowed (405)	Update the voluntary activity identified by volunteerId	Delete the voluntary activity identified by volunteerId
/volunteer/:volunteerId/timeslot/:timeslotId/register	Check if the authenticated donor user has registered to the timeslot identified by timeslotId of voluntary activity identified by volunteerId or not	Register to the time slot identified by timeslotId of voluntary activity identified by volunteerId	Method not allowed (405)	Unregister to the tim eslot identified by timeslotId of voluntary activity identified by volunteerId

Table 3.14: Partial Supported APIs for Volunteer Router

3.7 Database ER Diagram

An entityrelationship diagram (ER diagram) depicts the logical design of databases through the associations between different entities. An entity represents a set of objects (in *MongoDB* is called collections) which conceptually are of the same type. According to the relationships of entities, *MongoDB* schema can be mapped.

An ER diagram mainly contains entities and relationships. In addition, attributes of each entity are also presented, in which different symbols are used for different types such as mandatory (no special sign), identifying (with underscore), optional (with question mark) and multi-valued (with start sign). Another important extension in ER diagram is cardinality constraints on relationships. Look-here cardinality constraints are used in the ER diagram, which states the number of occurrences of

the entity next to the constraint.

To implement the functionalities of this project, the database is used to store different data. Since there exist relationships between different collections in the website database, an ER diagram is chosen to model them. The diagram is shown in Figure 3.7, and is composed of 8 entities and 11 relationships between different entities. In relationships, 5 are with multiple-multiple associations, so they are also mapped into collections in *MongoDB*. Together with 8 entities, they make up the 13 collections shown in the Figure 3.5.

There are two main parts in this model, namely charity users part and donor user part. Each charity user owes one charity and for every charity, it has its own attributes as well as four weak entities, which are bound to the charity entity. Among them, volunteer activity entity is another essential part - it is created by a charity, and has different time slots for each activity. The Donor entity which stores the documents of donor users has different relationships between Charity entity and can sign up different time slots for voluntary services.

Details of specific entity and relation will be explained in its implementation part in Chapter 4.

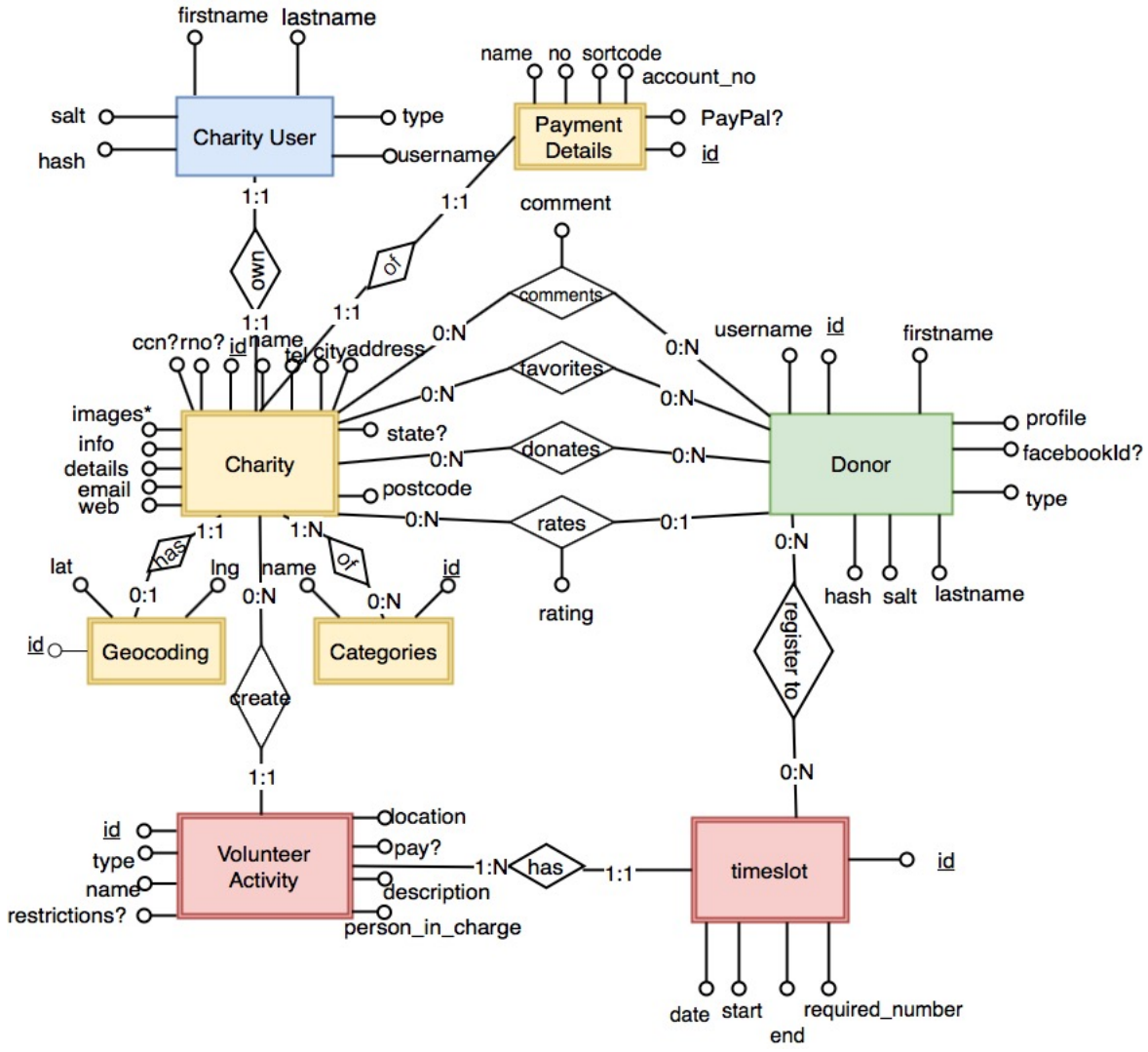


Figure 3.7: Entity Relationship Modeling for Linking Charities Database.

Chapter 4

Implementation

In this chapter, implementations will be explained by functionalities. There are five main sections: authentication, search, donation, voluntary work and feedback. All these sections are explained in this chapter. Each part will begin with a description of the work it serves, then with implementations from client-side, server-side and database perspectives. In the client-side, both the UI design and the Angular implementation will be covered. As there are nearly 50 APIs (See Appendix C and D) for all APIs provided by the server, it is not feasible to cover all in detail in this report, so only significant ones are illustrated in this chapter.

Functionalities such as profile and activities management are not detailed because the implementations are similar to others' and comparably easier, but the architecture of implementation for those features and *Angular* components are provided in Chapter 3. Deployment is also covered in this chapter.

4.1 Authentication Section

As a portal of philanthropy, user registration is essential for charities' owners and donors to manage their accounts and information. In this part, implementation for sign-up and log-on will be covered. A brief explanation of this section's functionalities is in the Table 4.1.

	Donors	**Charity Users**
Sign up	Sign up via an account	Sign up via an account
	Sign up via Facebook	Charity Verification & Charity preview
	Check user exists or not	Check user exists or not
Log in	Log in via an account	Log in via an account
	Log in via Facebook	

Table 4.1: Authentication Capabilities

The client-side communicates with the server through different server routers which perform operations on various *MongoDB* collections. The design architecture of this section is depicted in Figure 4.1.

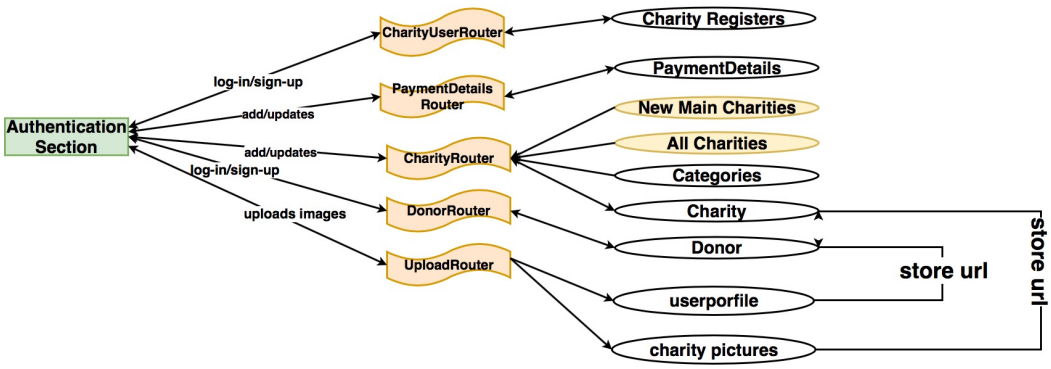


Figure 4.1: Authentication Section Implementation Architecture

4.1.1 User Registration

User Registration part enables both donors and charities' administrators to sign up. For donor users (Figure 4.2), it has additional third-party authentication support. To implement such functions, the two types of user models (Donor and Charity Register) are required in the database, and suitable authentication strategies are needed. RESTful APIs serves to communicate with the database from the client.

Client-side Implementation In order to make the user interface concise and simple, the access to user registration is on the same page and separated by different tabs. Users can choose to register as a specific type of user depending on their needs quickly and easily.

In registration procedures, different information is required according to user types. If all form data is collected on one page, users may be fed up with the long proce-

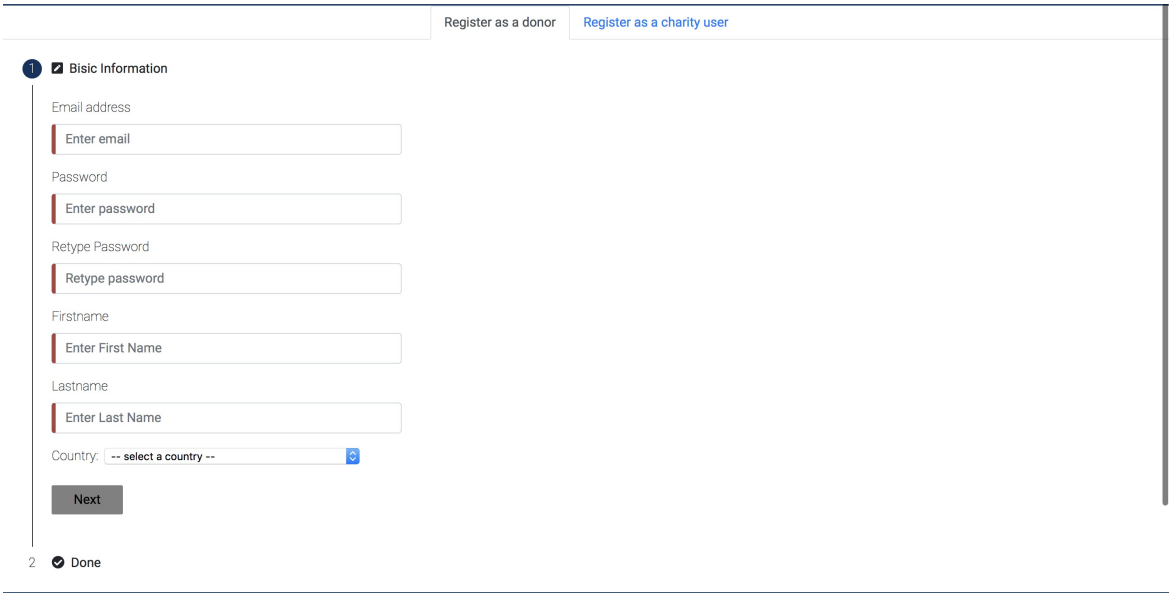


Figure 4.2: Donor User Sign-up Page User Interface

dure, so I resort to using *stepper*[1] to separate different kinds of data into separate steps, and once a user fills in one form he or she is able to continue to the next stage.

On the donor user sign-up page, only once the donor finishes the basic information can they click the next button to make a submission. Username existence check will be carried out at this step as well.

On the charity user sign-up page as shown in Figure 4.3, there are five steps to complete the registration. Every next step is enabled only when the information required is filled out correctly on the current step. Otherwise error messages will appear. Charity details, user information, charity payment account and promotional images are collected on separate tabs. When all required information is completed, the charity can check preview page on the final step and decide to modify information or submit.

The verification of charity is supported on the first step. If a charity is registered in U.K. then it can fill in its registration number and click the search button. All the required information provided by the Charity Commission will be automatically filled in and cannot be modified.

For Angular Implementation, the architecture is depicted in the Figure 4.4. There is a sign-up component which made up of two other sub-components: Donor Register Component and Charity Register Component. Each sub-component is a tab content of the sign-up page.

[1]UI component provided by Angular. Source: https://material.angular.io/components/stepper/overview. Accessed: 2018-09-02

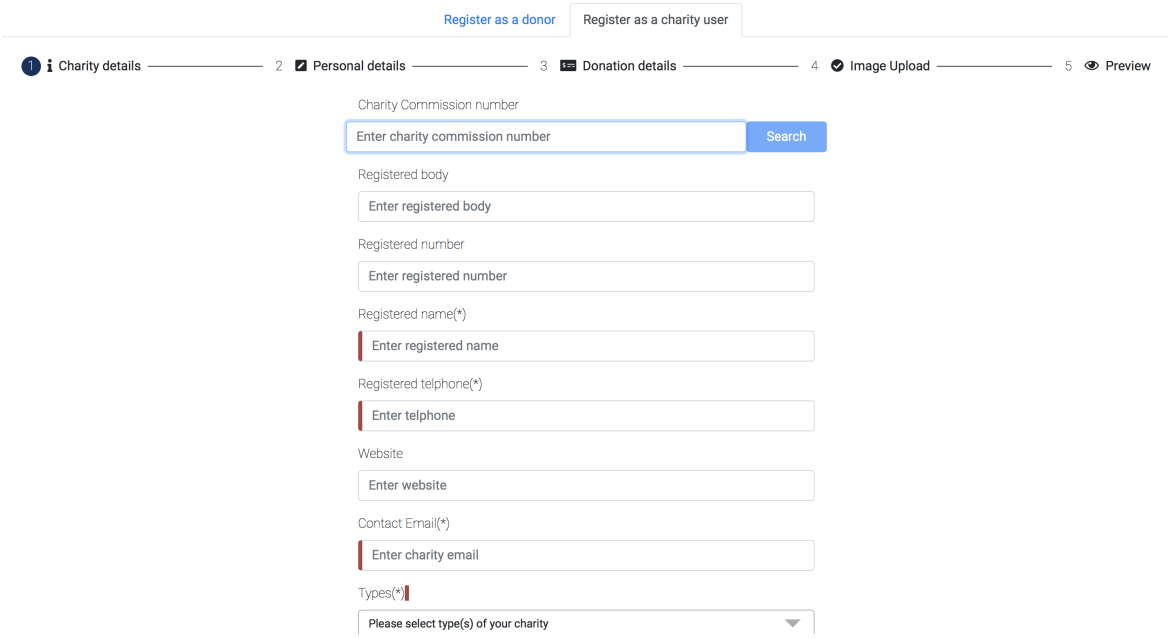


Figure 4.3: Charity User Sign-up Page User Interface

To build a form to collect user data, Angular FormBuilder [2] service is used. This is because Angular Form support for:

- Two-way data binding using [(ngModel)] directive on form fields then data from Typescript file will be bind with the template file.
- Change tracking that quickly determines the filed is modified or invalid and so on.
- Form validation and error handling which enables the system to give warnings or make some logical operations.

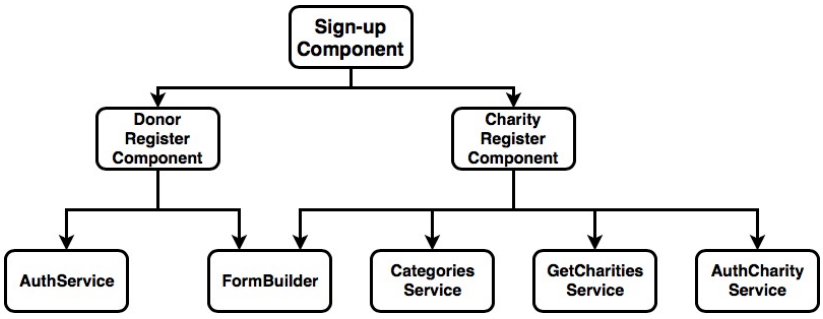


Figure 4.4: Sign Up Component Structure in Angular

In *Donor Register Component*, one personal information form of type *FormGroup*[3] is created along with its validators and error message. Angular form supports binding

[2]Source: https://angular.io/api/forms/FormBuilder. Accessed: 2018-09-02
[3]Source: https://angular.io/api/forms/FormGroup. Accessed: 2018-09-02

form values, validators, error and error messages together. In this component, validators like require, pattern, length, and password check are used.

Once the angular form is valid, which is a property of *FormGroup* class. The *next* button element in the template file binds its property $[disable]$ with the form's valid attribute via property binding. So if the form is valid then the button will be available, and be disabled otherwise.

After the form is filled out, the form data should be uploaded into the server to create a user in the database. As discussed in Chapter 3, in Angular, it is good practice to use Angular Service to communicate with the back-end. *Auth Service* is therefore created. It serves to upload the user form value in JSON format, which is available by *FormGroup* class's $value$ attribute.

In *Auth Service*, to support sign up, I created a signup function to post to sign-up API using *HttpClient* as mentioned in Section 3.1.4. After the asynchronous operation is finished, the client-side will subscribe to the returned *Observable* object and make responses.

To avoid the duplicated username, the system will also check the existence of username via *Auth Service*.

In *Charity User Register Component*, as shown in Figure 4.4, it is much more complex. Just like Donor User Component, it needs forms to collect data. But as the design of *Charity User* and *Charity* models, there are four forms created. A brief explanation is in Table 4.2.

charityDetailForm	Information used to create an document in *Charity* collection
addressForm	Information of charity address used as one field of charityDetailForm.
basicInfoForm	Information used to create an document in Charity User collection
paymentDetailsForm	Information used to create an document in Payment Details collection

Table 4.2: Forms And Brief Explanation for Charity User Sign-up

Because charity users require to choose the categories of charity, so *Categories Service* is used to retrieve the categories data from the server.

To enable charity users to upload promotional images, the *Image-upload Component* provided by the third-party module, *Image Upload Module*[4] is utilized. It provides a

[4]Source: https://www.npmjs.com/package/angular2-image-upload. Accessed: 2018-09-02

user-friendly way for image uploading. Once the images are transferred to the Azure Storage, URL address of images will be returned, and those addresses are thereafter stored in the charity document's images field. All this work is done by *AuthCharity Service* via an API provided by *UploadRouter* as illustrated in Figure 4.1.

For U.K. charities that have registered themselves in the Charity Commision, they can be verified using their register number. This is operated via *GetCharityService*. All the corresponding APIs used will be listed and explained in the section Server-side Implementation in Table 4.3 and Table 4.4.

To implement the feature of finding charities on the map, geocode details are required for charity documents. So after the charity is created in the database, in the client-side, it will use *GetCharitiesService* to query the *Google Map API*[5] and fetch the geocode information of the charity's address, and then attach needed information to the geocode field of the charity document.

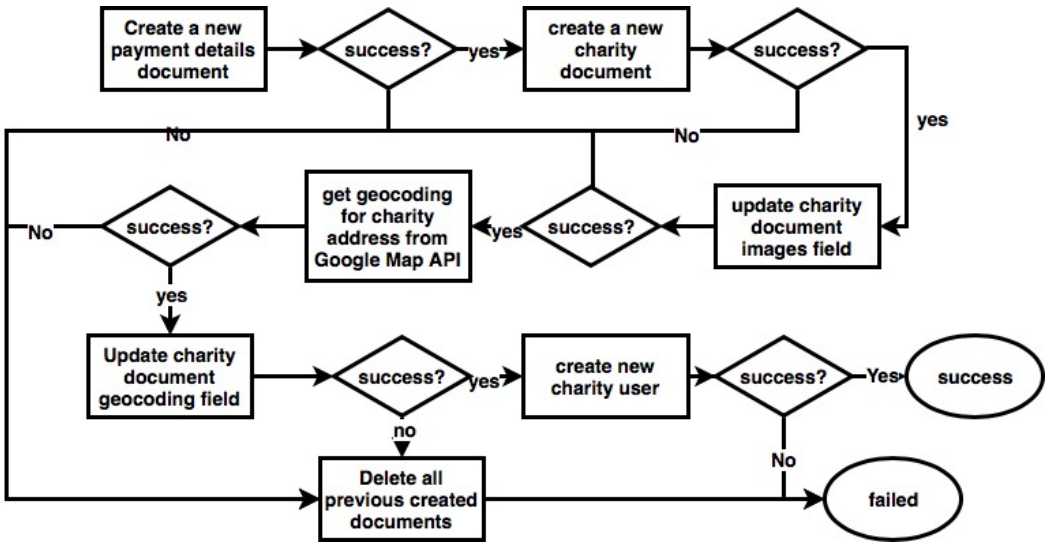


Figure 4.5: Charity User Registration Client-Server Communication Procedure in Angular

All these operations such as information posting, images uploading and API querying are asynchronous, but some requests depend on the finish of other requests. In this case, *mergeMap* pipe[6] is used for dealing with sequential inner Observables Events. Once the charity user submits its registration, the payment detail information will first be uploaded to the server and create a payment detail document in *MongoDB*, then its document *ObjectId* will be stored in card field of the *charityDetailsForm*. The *charityDetailsForm*'s value thereafter is posted to the server and creates a new charity document in Charity Collection. After the creation, images will be uploaded to

[5]Source: https://maps.googleapis.com/maps/api/geocode/. Accessed: 2018-09-02

[6]Source: https://www.learnrxjs.io/operators/transformation/mergemap.html. Accessed: 2018-09-02

Azure and then their images' URIs are saved in the images field of the charity document. Next, *getCharityService* queries Google Map API and return the geocoding information, and change the geocoding field of the created charity. After all this, the *basicInfoForm*'s value together with the charity document's *ObjectId* is uploaded to create a new charity user. If one phase fails, then all previous documents created in the database will be deleted. The procedure is demonstrated in Figure 4.5.

Server-side Implementation In *Donor Register Component*, only *AuthService* is used and two APIs are requested. As shown in Table 4.3.

AuthService	POST /users/signup
	POST /users/checkId

Table 4.3: Services and APIs used in Donor Register Component

The definitions of these two APIs will be illustrated in Appendix D so only some implementation key points are discussed.

In this project, *Passport*[7], an authentication middleware for Node.js is applied. A comprehensive set of strategies supporting authentication are available such as local strategy, *OpenID* [8],and *OAuth* single sign-on [9]. To create an account on the website is the local strategy. Passport-Local [10] is then utilized to create a local strategy for donor user authentication. After configuration, we can just use the local strategy to authenticate users.

To coordinate with authentication, passport-local-mongoose[11] plugin for Mongoose is applied which provides *User* Model register function. So *User* Model first registers a new user via username and password and then updates the user's other information. After saving the changes, Passport uses local strategy to verify if the user is registered successfully.

Username existence check is implemented in the server by querying username via Mongoose *User* Model to check if there exists a user with the username same as the requested username.

In *Charity Register Component*, three Angular services are applied and ten APIs are requested as shown in Table 4.4 and the metadata of partial APIs used in this section

[7]Source: http://www.passportjs.org. Accessed: 2018-09-02
[8]Source: http://openid.net/connect/. Accessed: 2018-09-02
[9]Source: https://oauth.net/2/. Accessed: 2018-09-02
[10]Source: https://www.npmjs.com/package/passport-local. Accessed: 2018-09-02
[11]Source: https://github.com/saintedlama/passport-local-mongoose. Accessed: 2018-09-02

are explained in Appendix D.

CategoriesService	GET /categories
GetCharityService	GET /maps.googleapis.com/maps/api/geocode/json ?address=address
	PUT /charities/:charityId/geocode
	GET /charities/ccn/:regno
	DELETE /charities/:charityId
AuthCharityService	POST /paymentdetails
	POST /charities
	POST /charityusers/signup
	POST /imageUpload/charitiesPics/:charityId
	POST /charityusers/checkId

Table 4.4: Services and APIs used in Charity Registrater Component

To upload images to the server or local storage in Node.js, the package *multer*[12] is utilized which is a Node.js middleware for handling multipart/form-data. In the server side, we need to configure the storage destination to Azure Storage Account and the file filter for *multer* package. After this, the API is available. If the upload is successful, file (if upload single image) or files (if upload multiple images) field will be added to Express Request Object, and return corresponding results to the client according to that.

Database Implementation In this section, there are external data and internal data. External data such as residential countries data [13] and U.K. charities data are used during registration. Internal data such as charity user profile and donor user profile are created in this system.

Information like the residential country of donor user is collected because it would be meaningful to analyze donors behaviors of different countries in the future.

To verify charity users, data containing charities information is needed. In this project, the charities data obtained from the Charity Commission are made use of to do verification for registered charities in England and Wales. Similar databases are assumed existent for other countries or regions. The charity data [14] used in this project is the latest August 2018 version. There are 15 files [15] and only two files are

[12]Source: https://www.npmjs.com/package/multer. Accessed: 2018-09-02

[13]Source: https://github.com/mledoze/countries/blob/master/countries.json. Accessed: 2018-09-02

[14]Source: http://data.charitycommission.gov.uk/default.aspx. Accessed: 2018-09-02

[15]For details via source: https://data.ncvo.org.uk/a/almanac16/how-to-create-a-database-for-charity-commission-data/. Accessed: 2018-09-02

used for verification in this project because all the information required to generate a charity document in this project is in these two files. One is *extract_charity* which contains the main information of charities, and another one is *extract_main_charity* which contains more detailed information [16] such as charity website, email address and company number. The data downloaded is not available immediately. First, it needs to be transformed from the ZIP into CSV files use the python scripts [17]. Then import the needed CSV files via *MongoDB* data migration tool, *mongoimport* [18].

Field	Type	Required?	Description
firstname	String	true	Donor user's first name
lastname	String	true	Donor user's last name
country	String	true	Donor user's country of residence
profile	String	false	Donor user's profile image URL.
type	String	false	Donor user's type. Donor as default.
facebookId	String	false	Donor user's Facebook id if the user is registered via Facebook.
blobName	String	false	Donor user's profile image name in Azure Storage Account.

Table 4.5: User Schema Definitions

The donor *User* model is presented in Figure 4.6. Basic information of user is stored together with the account information. The modeling is mapped into mongoose Schema and the detailed definitions of field are described in Table 4.5.

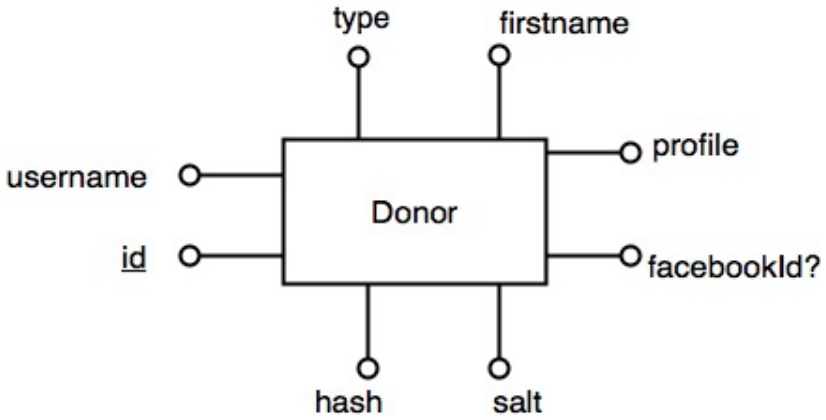


Figure 4.6: Donor User Profile Modeling in Database

[16]Table definitions are available via source: http://data.charitycommission.gov.uk/data-definition.aspx. Accessed: 2018-09-02

[17]Source: https://github.com/ncvo/charity-commission-extract/. Accessed: 2018-09-02

[18]Source: https://docs.mongodb.com/manual/reference/program/mongoimport/. Accessed: 2018-09-02

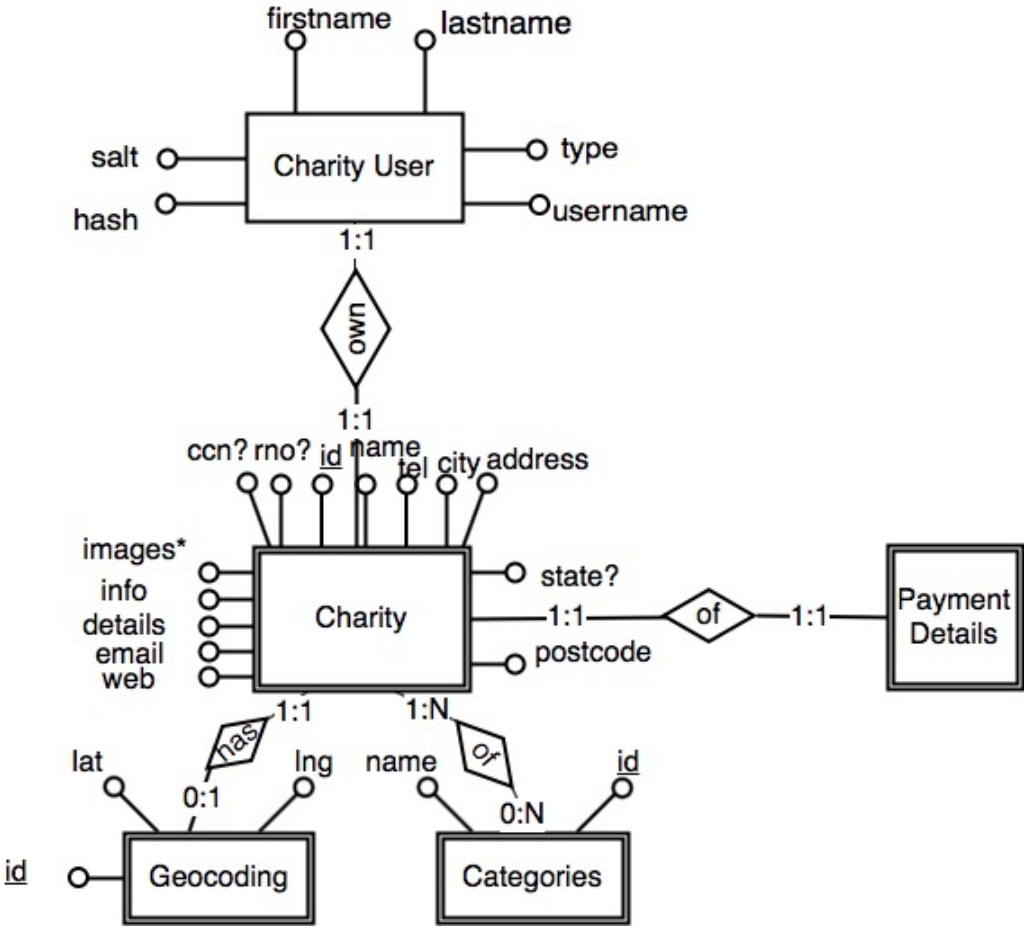


Figure 4.7: Charity User Profile Modeling in Database

The *Charity User* model is shown in Figure 4.7 and it is much more complicated. The *Charity User* object possesses a charity entity and one charity entity has another three weak sub-entities together with its own basic attributes. A weak entity can be implemented as an independent Schema or sub-document Schema. An independent mongoose Schema stores documents in its own collection while the sub-document Schema can only exist as a field of a super document. To decide how to interpret the weak entity mainly depends on the relationships that the entity has. If an entity does not have multiple-to-multiple relationships with other entities, it is mainly implemented as a sub-document because it only exists as part of the parent entity, just like *Geocoding* entity. While if the sub-document needs to be queried frequently such as *PaymentDetails* entity, it should be interpreted as independent schema. For the relationships, normally a multiple-to-multiple relationship will be implemented as an independent schema, but the Comments Relationship, shown in Figure 3.7, is defined as a sub-document of charity because comment documents can only be accessed via the public charity profile page, so it can be a sub-document of charity. A sub-document, not a collection, in *MongoDB* will be created based on the schema. It is a part of the parent document. While as an independent schema, a new collection will be created according to it and it is associated with its parent document via

ObjectId. Mongoose provides *populate* operation that enables the parent document to generate its sub-documents via *ObjectId*. Every document of *MongoDB* will be automatically assign an *ObjectId*. The interpretation from the charity user ER modeling to mongoose Schemata is listed on Table 4.6.

Field/ Collection/ Schema	**Type**	**Required?**	**Description**
Charity Register			**Charity Register Schema**
firstname	String	true	Charity user's first name
lastname	String	true	Charity user's last name
charity	ObjectId	true	The charity that managed by this charity user. Object Id in reference to Charity Schema
type	String	false	Charity user's type. Charity as default.
Charity			**Charity Schema**
ccn	Number	false	Charity Commission number registered in the Charity Commission.
rbody	String	false	Register Body in the Charity Commission.
rno	Number	false	Register Number in the Charity Commission.
name	String	true	Charity name
tel	String	true	Charity telephone number
web	String	false	Charity website
email	String	true	Charity contact email
categories	[ObjectId]	true	A list of charity categories that the charity belongs to. ObjectId in reference to Category Schema.
images	[String]	true	A list of promotional images URLs
info	String	true	Brief information of charity
details	String	true	Detailed information of charity
postcode	String	true	Post code of charity address
country	String	true	Address that the charity belongs to
state	String	true	State that the charity belongs to
city	String	true	City that the charity bases in
address	address Schema	true	The charity address
comments	[comment Schema]	false	The comments that charity received

card	ObjectId	true	The donation detail information of the charity. ObjectId in reference to PaymentDtail Schema.
geocoding	geocoding Schema	true	The geocoding information of charity.ObjectId in reference to Geocoding Schema.
comment Schema			**auxiliary Schema for Charity Schema**
author	ObjectId	true	The author of comment. ObjectId in reference to User Schema.
comment	String	true	The content of comment
address Schema			**auxiliary Schema for Charity Schema**
line1	String	true	The first line of charity address.
line2	String	false	The second line of charity address.
geocoding Schema			**auxiliary Schema for Charity Schema**
lat	Number	true	The latitude of charity address.
lng	Number	false	The longitude of charity address.
Categories			**Category Schema**
name	String	true	Category name
Payment Details			**Payment Details Schema**
name	String	true	Account holder name
number	Number	true	Card number
sortcode	Number	true	Account Sort code
account_no	Number	true	Account Number
paypal	Number	true	Palpal number

Table 4.6: Related Schema Definitions for Charity Registration

4.1.2 User Login

Client-side Implementation Login component is designed as a modal (Figure 4.8 and 4.9) because it will be launched by other components frequently. When the authentication is required, this website will launch the modal.

For Angular implementation, *NgbModal Service* is used to wrap the component as a modal, and *AuthService* is used to authenticate donor users by the server and *AuthCharityService* is for charity users similarly. To access a Facebook frame in the

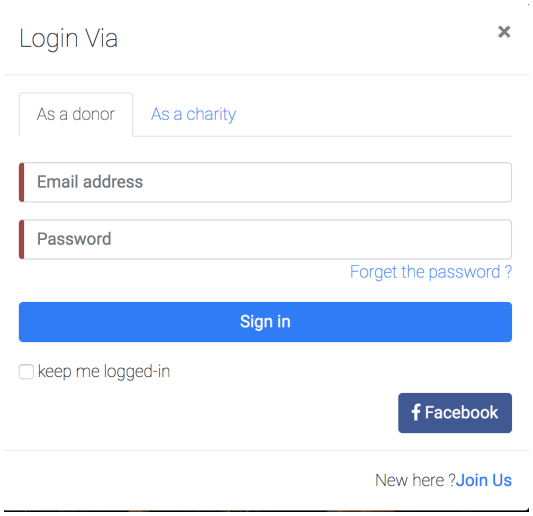


Figure 4.8: Donor User Login User Interface

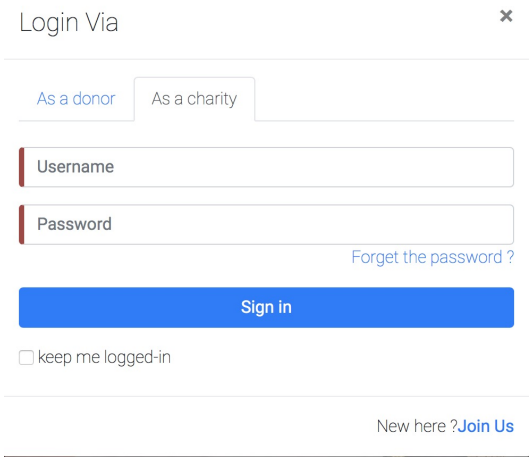


Figure 4.9: Charity User Login User Interface

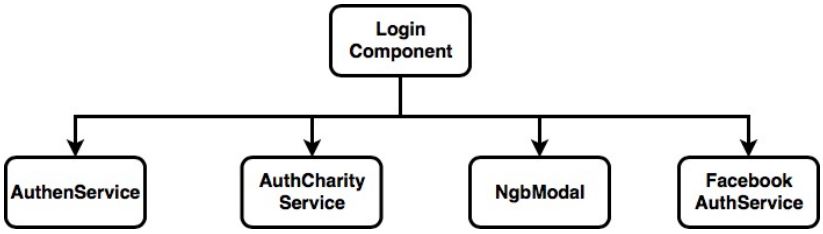


Figure 4.10: Login Component Structure in Angular

Angular application, *FacebookAuthService* provided by angular5-social-login[19] is utilized. OAuth2 authentication is operated through Facebook frame and user token is returned. The APIs and services used are listed in Table 4.7

AuthService	POST /users/login
	GET /users/facebook/token?access_token={token}
AuthCharityService	POST /charityusers/login

Table 4.7: Services and APIs used in Login Register Component

Once a user logs in with an account on this website, the server will return JSON Web Token (JWT), and the JWT is then stored in the local storage of browser. To keep user login status alive, when the web application is initialized, the credential token will be loaded and the client will check the JWT with the server. If the JWT is invalid, then destroy the credential. Otherwise, every communication between the client and server will be intercepted and set a field of *Authorization* with the credential, so that it is recognized as an authenticated request.

[19]Source: https://github.com/sabyasachibiswal/angular5-social-login. Accessed: 2018-09-02

Server-side Implementation To enable Facebook login, a *FacebookTokenStrategy* for *passport* package is created using the *passport-facebook-token* [20] module. In this strategy, first check if the *User* collection contains a user with this facebook id, if so return the user, else create a new user document and set facebook id field, then return newly created user.

For account login, it is authenticated by local strategy although charity users and donor users adopt different local strategy objects. After the user is authenticated, the server will create a signed token and send it back to the client. This is called the Token-based Authentication strategy. Cookies and Session Authentication strategy is not used for this project because if so, the server requires to store session information for users, which is not in compliance with the principal of stateless server. In addition, Mobile application platforms have trouble handling cookies/sessions. Using Token-based Authentication, no user status is stored so it has an advantage in scalability and efficiency.

4.2 Search Section

Searching is designed of two types; one is multi-keys search whilst another is geographical search. The functionalities are shown in Table 4.8.

Multiple-key Search	By name	By charity categories	By charity location
Geographical Search	Find charities on map		

Table 4.8: Search Capabilities

As illustrated in Figure 4.11, search section in the front-end query search keys from the *Charity* collection in *MongoDB* database is via the *Search Router*. For the geographical search, as all the *Charity* document has a field of *geocoding*, so the client-side requests the charities from the server and use the returned data to render the map.

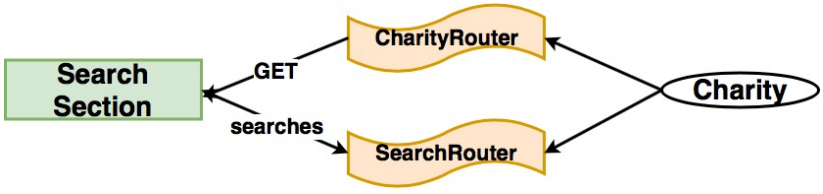


Figure 4.11: Search Section Implementation Architecture

[20] Source: https://github.com/drudge/passport-facebook-token. Accessed: 2018-09-02

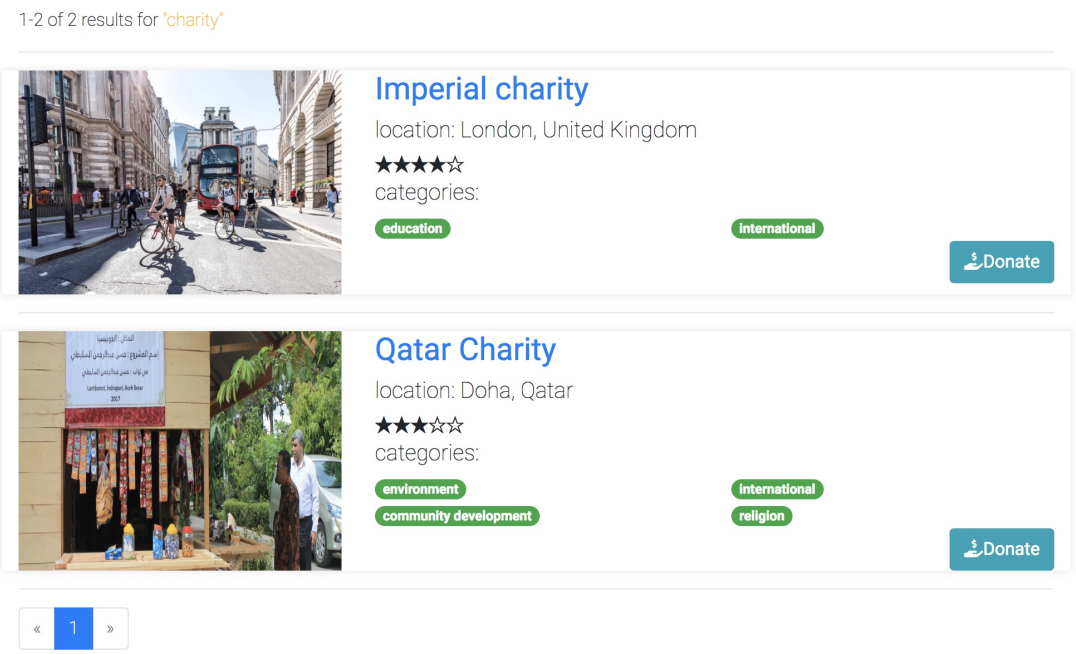


Figure 4.12: Search Page User Interface

4.2.1 Multiple-key Search

Client-side Implementation As an essential feature, the basic design principle for searching is convenience and accessibility for users so I designed a search bar which placed on the header of the page as well as the middle of the main page. After the search keys are inputted, the page will navigate to the search page as shown in the Figure 4.12 which displays search results. While data is loading, the loader will show in response. In the search result page, users can see search information such as search key, current page, index of charities and at most ten charities results in one page. Pagination is used for results.

The search results page is implemented as *Search Component*, and it is made up of three other dependencies (Figure 4.13). The search result page uses the same general Angular router but for each specific query parameters, it uses a specific activated

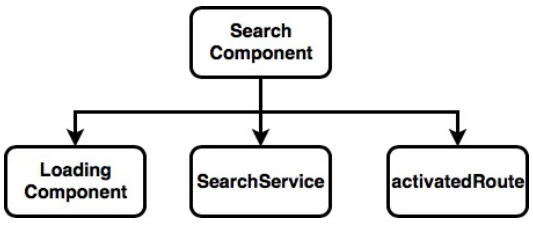


Figure 4.13: Search Component Structure in Angular

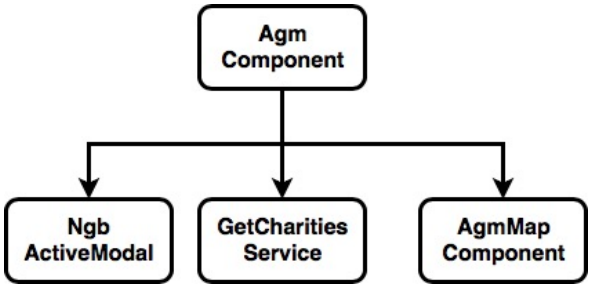


Figure 4.14: Geographical Search Component Structure in Angular

router, i.e. URL. This is useful to fetch the results of your search key after revisiting. The Angular *activatedRouter Service* serves this functionality. After the search key is hit, the *search Service* will request the search API with the query parameters and fetch data for client-side. When the data is loading, the search page will render the loader using *Loading Component*. The API used by the service is in Table 4.9.

SearchService	GET /search?q=query&page=page
GetCharitiesService	GET /charities/allcharities

Table 4.9: Services and APIs used in Search Component and Agm Component

Server-side Implementation Search charities via name/location/category is implemented via API

HTTP Request: GET /search?q={query}&page={page}

and return paginated list of charities whose name , location or categories are partially matched the search key. URL parameters definitions are in Table 4.10.

Parameter	Type	Description
query	String	Search keyword, such as charity name, charity category, charity location. Partial match is available.
page	Number	Page number of search results. Each page contains at most ten results. 1 as default.

Table 4.10: URL Parameters for Multiple-key Search API

The output variables and explanations are illustrated in Table 4.11.

Implementation Strategies To implement the search API, first figure out the match expression. After that, count the total number of results and query the results on the request page. These two functions can be done in parallel using Async package [21], which provides support for asynchronous JavaScript. The algorithm used is in Algorithm 1. In this project, for large data retrieval, pagination techniques are utilized which includes features like voluntary work search results, comments results, donation histories and so on. Pagination is quite significant for scalability because if all the data is returned to the client, it will cause a significant delay in the client application after a large-scale data loading which heavily harms the user experience. Segmenting the information is also done because it reduces the server load and reduces the response time of the database.

[21] Source: https://github.com/caolan/async. Accessed: 2018-09-02

Variable	Description
success	Boolean value which indicates if the request is successful.
message	Search result message.
search_result	A List of charities that match the search key in current page. Return in JSON array format. Each charity is of Charity Schema Type.
search_key	The search key that user requested.
totalNumber	Total number of charities that match the search key.
page	The return page number.
numberPerPage	The number of charities that a page can contain at most.

Table 4.11: Output Variables And Explanation for Multiple-key Search API

Algorithm 1 Search API Implementation Algorithm

Input: query, page
Output: JSON output as mentioned in Table 4.11.

$queryExpression = []$
PUSH {name LIKE query} INTO $queryExpression$
PUSH {city LIKE query} INTO $queryExpression$
PUSH {country LIKE query} INTO $queryExpression$
PUSH {state LIKE query} INTO $queryExpression$
if query IS one Category **then**
    PUSH {categories = category ObjectId} INTO $queryExpression$
for all functions **do in parallel**
function FUN1
    COUNT the number of charities that MATCH $queryExpression$
    return results[0]
function FUN2(queryExpression, perPage, page)
    FIND the charities that MATCH $queryExpression$
    LIMIT number
    SKIP $perPage * page$
    POPULATE categories
    return results[1]
end for
after functions **finish**
$totalNumber \leftarrow results[0]$
$charities \leftarrow results[1]$
return JSON

4.2.2 Geographical Search

Client-side Implementation Map searching is implemented as *AgmComponent* (Figure 4.14) and it is launched as a modal because the Google Map is in the shape of

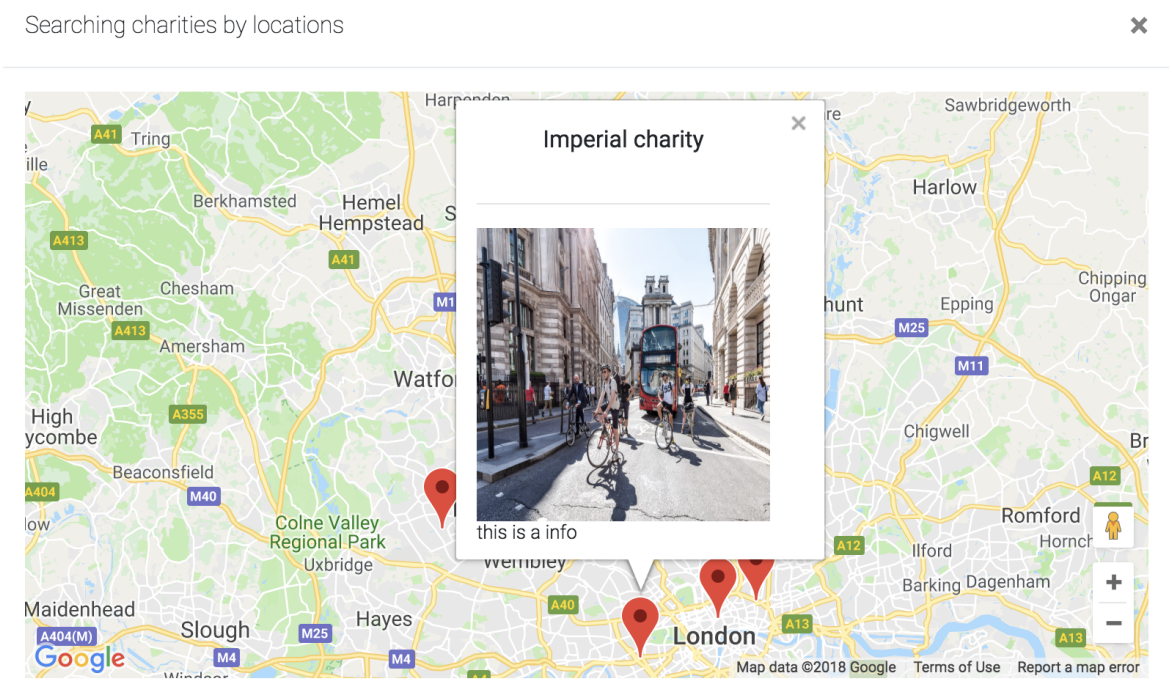


Figure 4.15: Geographical Search Page User Interface

a rectangle which is not suitable for a single page, and it does not fit with the contents of any other page. For all charities marked on the map, it is not enough for users to see the markers of existent charities, the charity brief information should be displayed. For the intuition of the users, when the marker on the map is clicked, an information window will appear (Figure 4.15) containing the charity name, charity image, and a brief introduction.

To implement the map search result page as a modal, *Ngb ActiveModal Service* is used. Another important Component is *AgmMap*, which is provided by a third-party Angular Module, *AGM* i.e. Angular Google Maps[22]. To add markers on the map, a list of geocoding information is required. This is the reason why *Charity* Schema contains a field named geocoding. *GetCharitiesService* (see Table 4.9) is applied to retrieve charities data from the server and the data is handled to draw the map.

4.3 Donation Section

Donations are the main financial source for charities; donors can donate directly to charitable organizations without registration. In addition, both donors and charities can check the monetary histories on this website. More detailed functionalities in this section are listed in Table 4.12.

[22]Source: https://angular-maps.com. Accessed: 2018-09-02

	Donation	Donation History
Donor User	Make a payment via Stripe API	Check all donation history
	Leave a message for this payment	Total money donated
	Share the successful donation information on social media	Transaction history pagination
Charity User	Receive a payment via Stripe API	Same As donor user
Un-Registered User	Same as donor user	/

Table 4.12: Payment Capabilities

As depicted in Figure 4.16, *Donation Section* in the client-side communicates with the server via *Donation Router* to perform database operations on the *Donations* collection.

Figure 4.16: Donation Section Implementation Architecture

4.3.1 Client-side Implementation

Donation Component is implemented as a modal so that is can be launched in any other components. Donors are able to choose or input the amount of money they wish to give and leave a message as seen in Figure 4.17. Then it processes to Stripe[23] payment gateway (Figure 4.18. After successful payment, donors can share the donation information on social media (Figure 4.18).

The implementation architecture in Angular are presented in Figure 4.20.

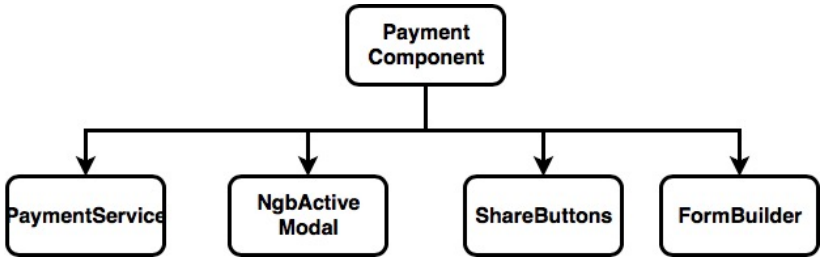


Figure 4.20: Payment Component Structure in Angular

[23]Source: https://stripe.com/gb. Accessed: 2018-09-02

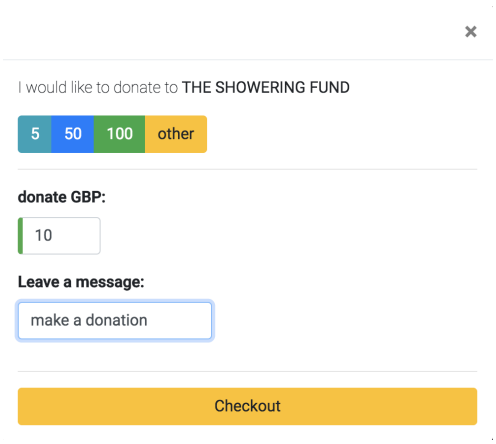


Figure 4.17: Donation Modal User Interface

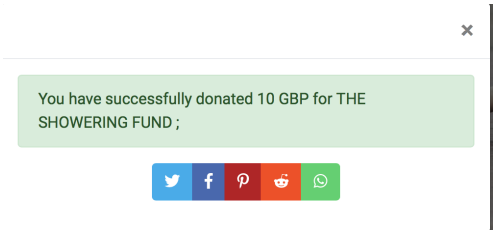


Figure 4.18: Post-payment User Interface

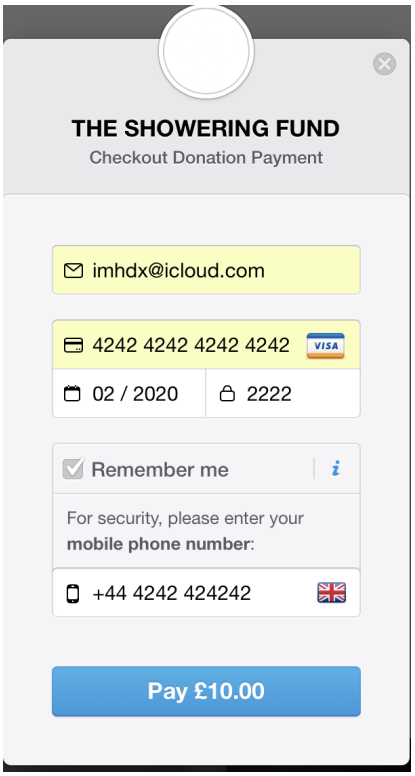


Figure 4.19: Payment User Interface

To implement the *Payment Component* as a modal, *NgbActiveModal* service is used. The payment details and messages are collect via *FormBuilder* service. The client-side utilizes *PaymentService* process the payment via Stripe gateway and store the donation histories into the *Donation* collection. Donor users and charity users can make use of *Payment Service* as well to check the donation histories.

The *Payment Service* and the APIs it used are presented in Table 4.13. Only one API in this table will be detailed in next section, and the details of all APIs used are in the Appendix C.

PaymentService	POST /donation	Make donation
	GET /donation/user?page={page}	User donation histories
	GET /donation?page={page}	Charity donation histories

Table 4.13: Services and APIs Used in Donation Section with A Brief Explanation

Implementation Strategy In order to use Stripe Checkout functionality in Angular, *Stripe* checkout script is needed and it will export the global variable *StripeCheckout*, which is used to create a handler by passing a configuration object. In the configu-

ration object, we can assign the payee's *Stripe* key, so that the donation goes directly into its Stripe account. The configuration object is adjusted for each distinct *Charity* object according to the charity's Stripe key [6].

4.3.2 Server-side Implementation

Three APIs are used in this section. For the donation history retrieval, the implementation strategies used are similar as described for the search API (Algorithm 1), so only the first donation API, which serves to make donations through Stripe API and create new donation document in *MongoDB*, will be illustrated here.

HTTP Request POST /donation
Input & Output Variables The input, output variables and explanations are illustrated in Table 4.14.

Input	*stripeToken*: Stripe transaction information created in client-side; *stripeKey*: Payee's Stripe key; *currentCharge*: number of money charged; *charityId*; message	
Output	*charge*	Stripe transaction information.
	success	Boolean value indicates if the donation is successful or not.
	results	Donation document created. Type: Donation Schema.

Table 4.14: Input, Output Variables And Explanation for Make Donation

Implementation Strategy The implementation strategy is in Algorithm 2.

Algorithm 2 Make Donation API Implementation Algorithm

Input: stripeToken, stripeKey, currentCharge, charityId, message
Output: JSON output as mentioned in Table 4.14

create stripe object using stripeKey ▷ payee's Stripe account
$source \leftarrow stripeToken$
create Stripe transaction using *source*
make transaction using *currentCharge*
make transaction and return *charge*
if Authenticate User **then**
    Create Donation document with User ID and *message* and *charge*
else
    Create Donation document using *message* and *charge* without User ID
end if
return JSON

4.3.3 Database Implementation

In database architecture (Figure 3.7), the *donates* is a multiple-to-multiple relationship between charities and donors. This relationship is designed as an independent schema as illustrated in Table 4.15.

Field/ Collection	Type	Required?	Description
user	ObjectId	false	User that made the donation. ObjectId in reference to User Schema.
charity	ObjectId	true	Charity that be donated to. ObjectId in reference to Charity Schema.
amount	Number	true	The amount of money for the donation
message	String	false	Message leaved by the donor.

Table 4.15: Donation Schema Definitions

4.4 Voluntary Work Section

Voluntary work section is one of the main sections in this portal. It allows users to search activities and find activities' details. If users are interested in an activity, they can sign up to any available time slot. In addition, voluntary work creation and management for both donor users and charity users are also supported. The details of the capabilities are presented in Table 4.16.

Voluntary Activity Details	Activity information
	View activity available time slots for authenticated donor users
	Sign up a time slot of the activity for registered donor users
Voluntary Activity Search	All voluntary activities available from now on
	All voluntary activities available on a certain date
	All voluntary activities available from a charity
Voluntary Activity Management	Charity user checks all created activities
	Charity user creates a new activity
	Charity charity user update/delete an activity
	Charity user checks activities' registers
	Donor user checks registered activities' time slots

Table 4.16: Voluntary Activities Capabilities

As depicted in Figure 4.21, the *Voluntary Section* communicates with the server to

manipulate the documents in *VolunteerActivities* or *Timeslots* collections via *VolunteerRouter* .

Figure 4.21: Voluntary Work Section Implementation Architecture

Only the voluntary activities search feature will be covered in this section and for the other functionalities, the design architecture and APIs were discussed in Chapter 3.

4.4.1 Client-side Implementation

The voluntary work page(Figure 4.22) is presented in a table, which presents the basic information of the activities such as activity name, a brief introduction, location, and its host charity. The brief introduction of each charity is shown in a responsive style, but if users hover the mouse over it, the full-text description will appear as a tooltip. The details of an activity are displayed in a neat table format as shown in Figure 4.23. It is likely that the available time slots of the activity are plentiful, so if the time slots are on the same page as the activity page, then the page will be long. For some users that are just browsing, they are not willing to see such information. Consequently, the available time slots for the activity are shown on another page (Figure 4.24) and can only be accessed by authenticated users.

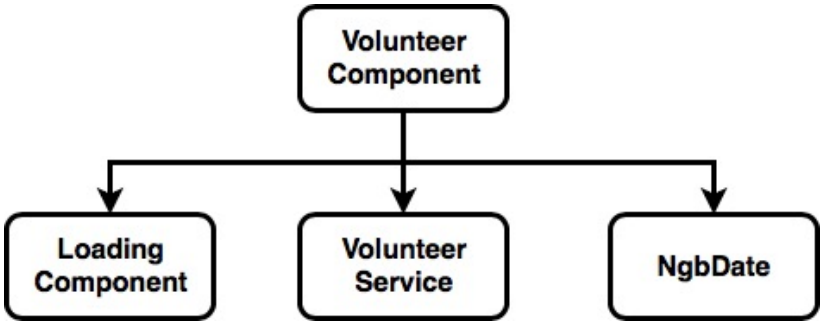


Figure 4.25: Voluntary Activities Search Component Structure in Angular

The voluntary search page is implemented by *VolunteerComponent*. As depicted in Figure 4.25, it is dependent on the other three dependencies. It uses *Volunteer Service* to fetch available voluntary work data and when data is loading, the *Loading Component* serves as a loader effect. The *NgbDate* is a component used as a date picker for date selection. In the *Voluntary Details Component* (Figure 4.26), *Volunteer Service* is injected for data retrieval. The *Activated Route Service* is used for navigating to a

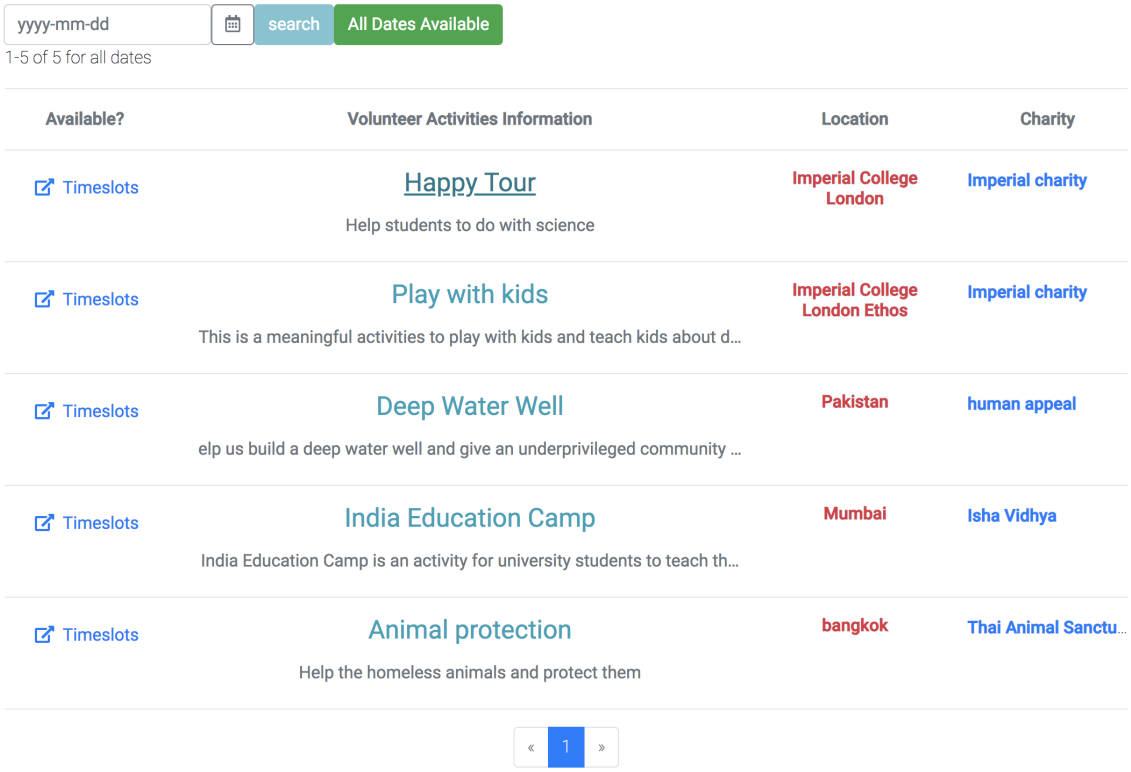


Figure 4.22: Voluntary Search Page User Interface

specific activity page. Because only authenticated users can check the available time slots, the *AuthService* is applied for login checking. For unauthorized users, *Login Component* will pop up for the login operation.

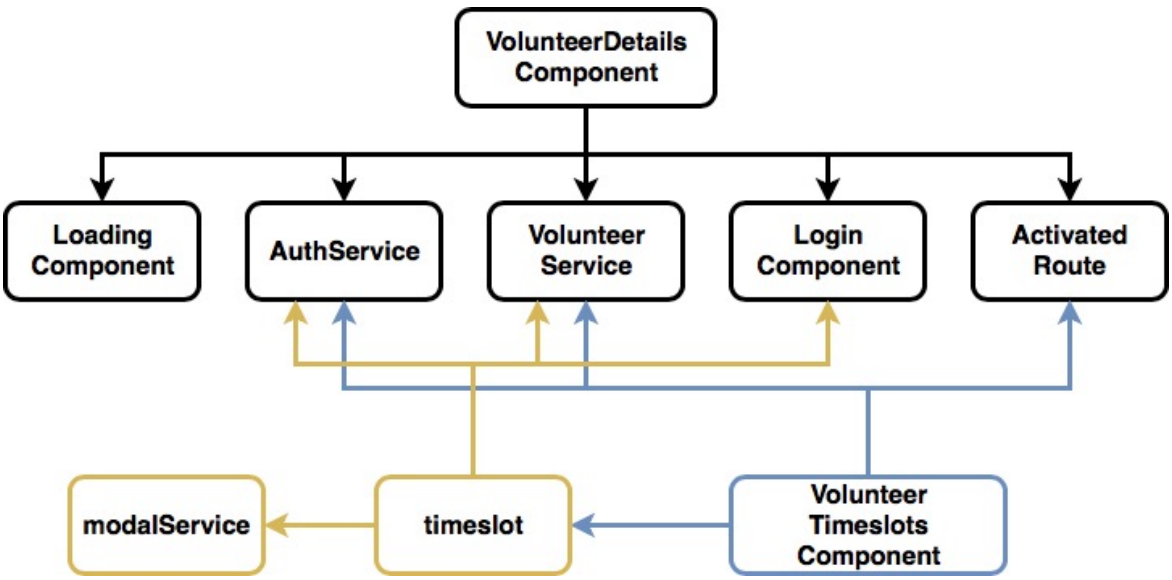


Figure 4.26: Voluntary Activities Details Components Structure in Angular

In the *Volunteer Timeslots Component*, it is made up of a list of time slots which is

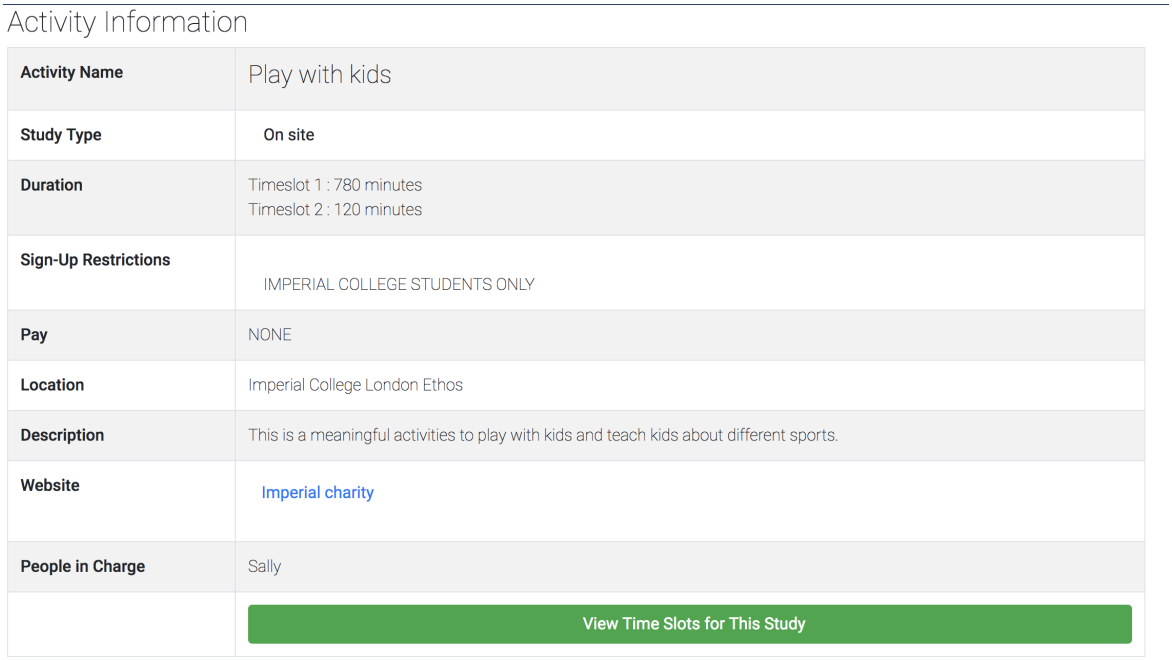


Figure 4.23: Voluntary Activity Detail Page User Interface

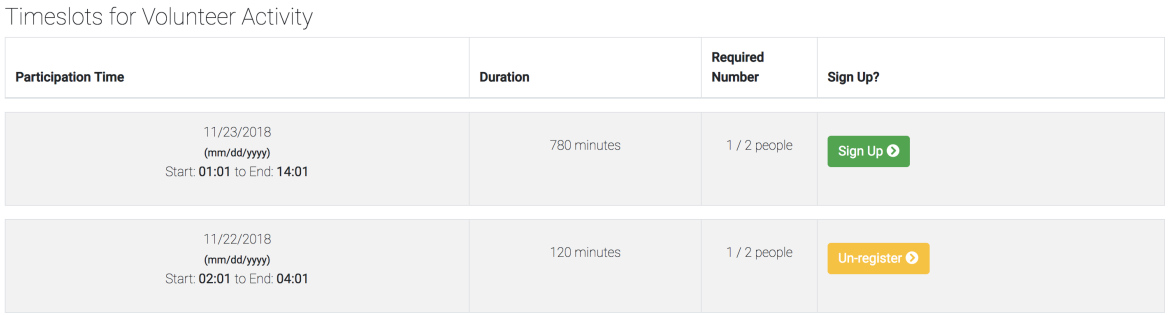


Figure 4.24: Voluntary Activity Registration Page User Interface

implemented as *Time Slot Component*, which also involves an authentication check. The registration status check and signup or dismiss operation is served by *Volunteer Service*.

All the APIs listed in the *Volunteer Router* (Refer to Appendix C)from back-end server are used by the *VolunteerService* in client-side.

4.4.2 Server-side Implementation

The voluntary activity search API also returns paginated activity results. The implementation for this is similar to the search API, with the difference of the query expression. By default, the query expression is the activity that has at least one time slot that is later than the search day, and it will return the available activities. But if one date is selected, then the query expression will be activities that have the time

slot available on that date.

As for API serves for signing up to time slots, authentication status is first checked. For an authenticated user, spare spaces of the time slot are then examined. Only when the free space is available, the user's *ObjectId* is pushed to *Time Slot* document's register field.

4.4.3 Database Implementation

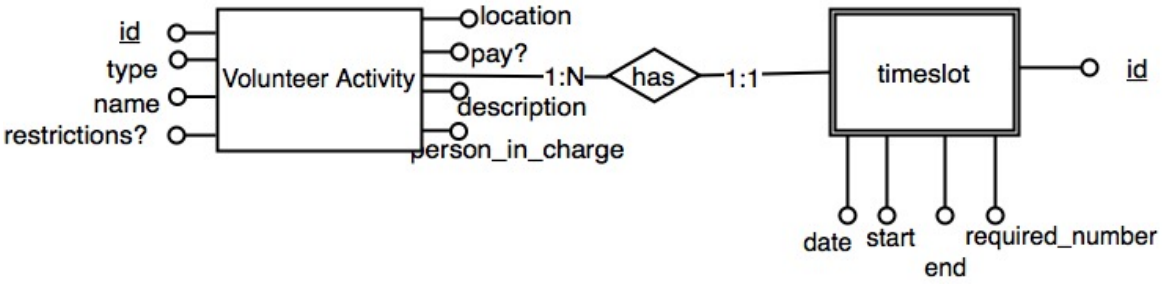


Figure 4.27: Voluntary Activities Modeling in Database

A voluntary activity model should not only contain basic activity information but also time slots of the activity. For each time slot, the number of people required may vary. As represented in Figure 4.27, *time slot* entity is a weak entity belonging to *VoluntaryActivity* entity, so it is implemented as a sub-schema. As for the relationship of registration between donor users and activities, it is mapped as a field of *time slot* entity, although it is a multiple-to-multiple relationship. The reason for this is because there is a restriction on the number of this relationship instances, so when the participants are in a field of the *time slot* document, it can generate a field for the number of current participants using *MongoDB virtual* function. Thus, when the required number of people for this time slot is full, the client-side can also make a response, such as disable signup button. This is a compliment to server check for time slot registration numbers. The detailed definitions of mapped schemata are in Table 4.17.

Field/ Collection/ Schema	**Type**	**Required?**	**Description**
Volunteers			**Volunteer Activities Schema**
timeslots	[ObjectId]	true	A list of time slots for the activities. ObjectId in reference to Timeslot Schema.
name	String	true	Activity name
location	String	true	The location that the activity is hosted in

pay	String	false	The subsidy that gives to the participants
description	String	true	The description of the activity
charity	ObjectId	true	The charity that hosts the activity.ObjectId in reference to Charity Schema
principal	String	false	People in charge of the activity
restrictions	String	false	The restrictions of the activity
study_type	String	true	The type of the activity. Two types: on side or remote.
Timeslots			**Timeslot Schema**
date	DateSchema	true	The date of the time slot
period	Period Schema	true	The starting time and the end time and the duration of the time slot.
required Number	Number	true	The number of people that required during the time slot of the activity.
date Timestamp	Number	true	The date timestamp of the begining of the time slot
registers	[ObjectId]	false	A list of registers that register to the time slot. ObjectId in reference to Donor User Schema.
Date Schema			**Auxiliary Schema for Timeslot Schema**
year	Number	true	Year of date
month	Number	true	Month of date
day	Number	true	Day of date
Period Schema			**Auxiliary Schema for Timeslot Schema**
start	TimeSchema	true	Starting time
end	TimeSchema	true	End time
duration	Number	true	Period duration.
Time Schema			**Auxiliary Schema for Period Schema**
hour	Number	true	Hour in 24-hour time systems
minute	Number	true	Minute of time

Table 4.17: Related Schemata definitions for voluntary work

4.5 Feedback Section

To build an open and transparent portal, it is essential for donors to make feedback to charities. This section illustrates the feedback features. Three means are available for donor users to give feedback as shown in Table 4.18.

The architecture for implementing such features in this project is depicted in Figure

Feedback	Rate charity by authenticated donor user
	Make comments by authenticated donor user
	Add to favorites by authenticated donor user

Table 4.18: Feedback Capabilities

4.28. Three routers are utilized by the client to make operations on three corresponding collections.

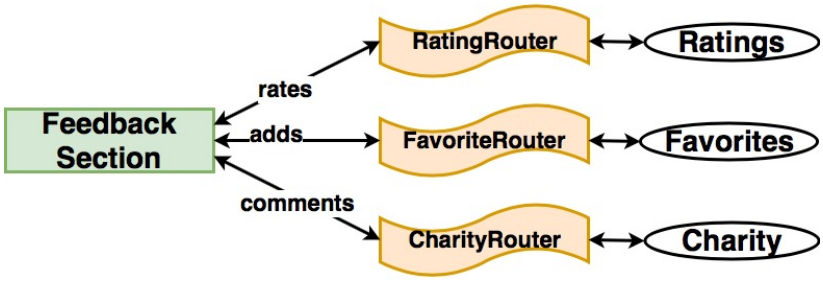


Figure 4.28: Feedback Section Implementation Architecture

4.5.1 Client-side Implementation

To enable user to quickly make feedback, in a charity card (Figure 4.31), users can quickly add the charity into favorites and see the amount of comments. Besides, in the *charity profile* page, authenticated donor users can make comments as shown in Figure 4.30 and rate charities and check average ratings (Figure 4.29).

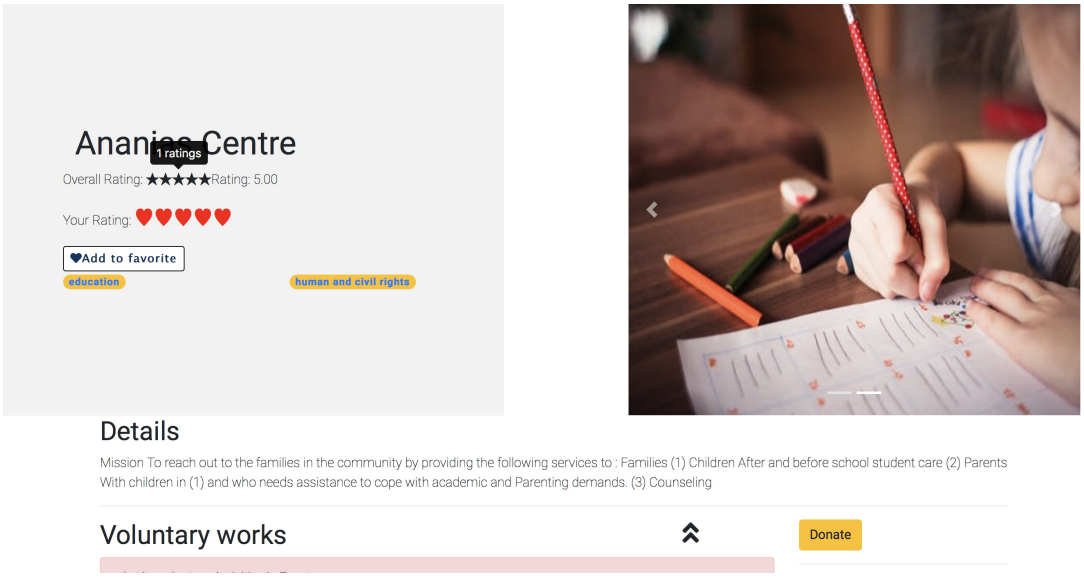


Figure 4.29: Rating User Interface

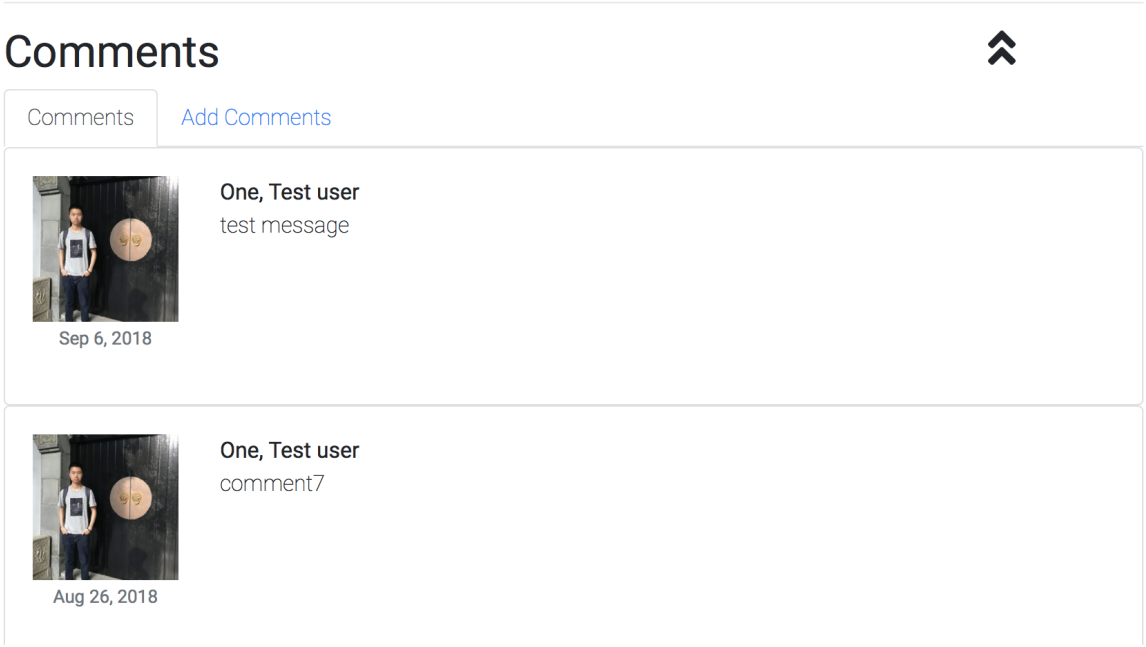


Figure 4.30: Comment Section User Interface

Three *services* are used to communicate with the server via different APIs as listed in Table 4.19.

FavoriteService	GET /favorites
	GET favorites/:charityID
	POST favorites/:charityID
	DELETE favorites/:charityID
GetCharitiesService	POST /charities/:charityId/comments
RatingService	GET /rating/:charityId/averageRating
	GET /rating/:charityId
	POST /rating/:charityId

Table 4.19: Services and APIs used in Feedback Section

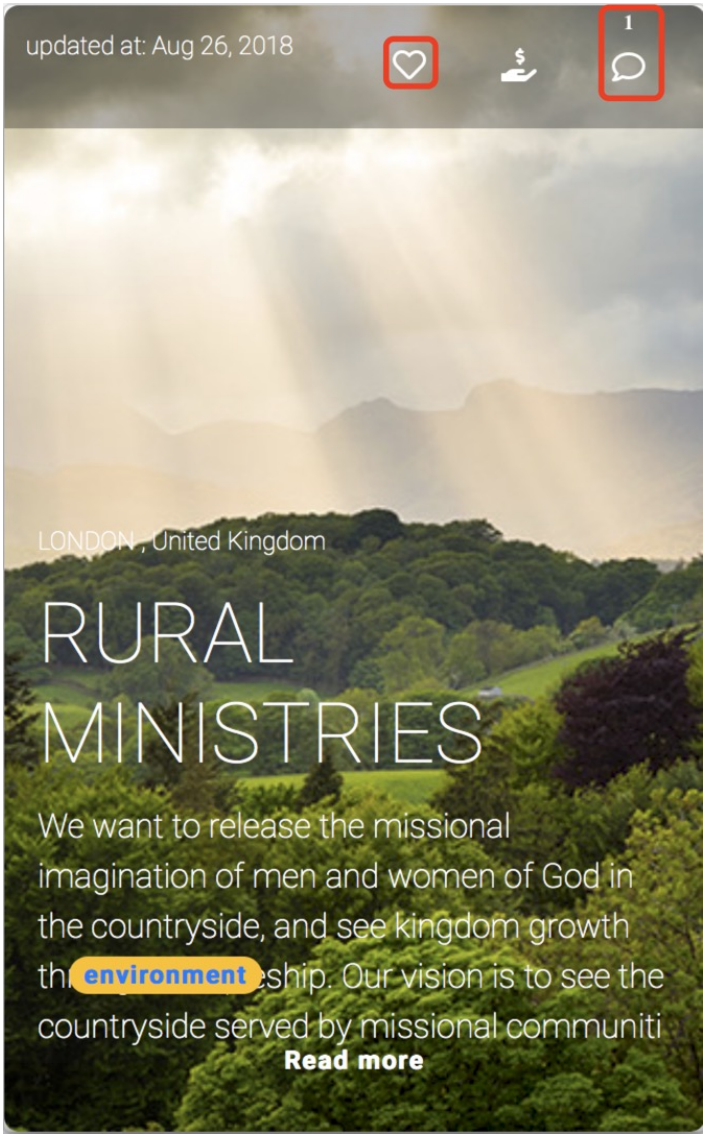


Figure 4.31: Charity Card User Interface

4.5.2 Server-side Implementation

Users can only make a rating for a charity and every time user check a charity profile, the rating he or she made should be displayed. Same as to add to *favorites* features, this is implemented by query the server to get the list of favorites/ratings.

To add to favorites, the client sends a *GET* request to the server, and remove a charity from favorites via *DELETE* request. For ratings, the same strategy is used.

To get average ratings of a charity, the server first query documents with the charityId in the *Ratings* collection and then uses aggregate functions, *avg*, to calculate the result.

4.5.3 Database Implementation

The feedback functionalities are the multiple-to-multiple relationships between donors and charities as presented in Figure 4.32. The *comment* relationship has been discussed in the section of authentication, so in this section only the others are discussed.

The *favorites* and *rates* relationships are mapped into two independent models in

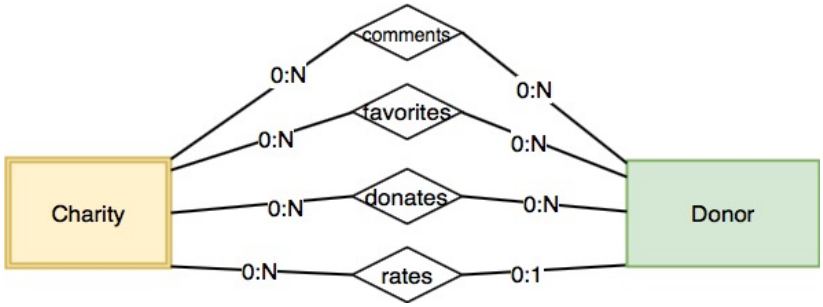


Figure 4.32: Feedback Section Modelling in Database

this project, this relationship does not belong to any types of user entities. The schemata of those two are in Table 4.20.

Field/ Collection/ Schema	Type	Required?	Description
Favorites			**Favorite Schema**
user	ObjectId	true	ObjectId in reference to User Schema.
charities	[ObjectId]	true	A list of charities that added to favorites by the user. ObjectId in reference to Charity Schema.
Ratings			**Rating Schema**
user	ObjectId	true	User that made the rating. ObjectId in reference to User Schema.
charity	ObjectId	true	Charity that be rated. ObjectId in reference to Charity Schema.
rating	Number	true	A rating on a scale from 1-5.

Table 4.20: Related Schema definitions for Feedback

4.6 Deployment

In order to make the website accessible to people around the globe, this project needs to be packed up and running in a production environment. A web application

running on the local machine cannot be accessed by non-local users.

There are many cloud services for web application and database deployment like Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform and so on. This project utilized the Azure cloud service. Specifically, App Service is used to set up server and client. Azure Cosmos DB serves as database and Storage Accounts is created to store images uploaded by users.

Following steps are needed to deploy the project into Azure Cloud.

4.6.1 Client-side Modification

Create an *App Service* in Azure with runtime stack set to be *Node.js*, and set the base URL that connects the Angular client to the back-end server into the web application IP address.

Use the *Angular CLI*[24] to build a distribution folder with the set of application files.

Replace the public folder in Express server application with the Angular distribution folder. After that, the Angular application is deployed into the local Express server.

4.6.2 Data Immigration

MongoDB Immigration Create an Azure *Cosmos DB* Service, which is the MongoDB service provided on Azure Cloud.

Migrating data from local storage to Azure Cosmos DB [25] using *MongoDB* immigration tool, *mongoexport* and *mongoimport*.

For the local collections that I created, as the amount of data is small, the immigration procedure works perfectly. But the two charity datasets retrieved from the Charity Commission are used in this project, which have 168297 and 358777 records respectively, are exported partially using the same method. Errors occurred when the large dataset was uploading because the request rate is large. In this case, MongoDB document partition [26] is required. This is because the default collection throughput in request unit (RU) is 1000 in Azure Cosmos DB, and 1 RU corresponds to the throughput of a read of a 1 KB document. But for large dataset immigration, the write rate is higher than the default throughput.

To solve this, in Azure *CosmosDB*, take the following steps:

[24]Source: https://cli.angular.io/. Accessed: 2018-09-02

[25]Tutorial available on https://docs.microsoft.com/en-us/azure/cosmos-db/mongodb-migrate. Accessed: 2018-09-02

[26]Source: https://docs.mongodb.com/manual/core/sharding-data-partitioning/. Accessed: 2018-09-02

- Firstly, create a collection container to which data will import to
- Then, set Shard key as well as the Throughput
- Lastly, re-import the collection

Storage Immigration Files uploaded by users were stored in local storage during the development period. This is not suitable practice for the public use so I created an Azure *Storage Account* Service to store the files. Three BLOB containers are created in the Storage Account for this project because three types of files are uploaded for this website, namely donors' profile images, charities' promotional figures and cover page images.

4.6.3 Server-side Modification

Three parts of the servers are needed to be modified.

Database Connection Change the *MongoDB* connection from local database into the newly created Azure Cosmos DB account via *Connection String* provided.

Storage Connection After cloud storage creation, on the server-side, the upload router should be configured to connect to Storage account and set the destination blob via access keys.

CORS Configuration For Cross-Origin Resource Sharing (CORS) options in the server, IP address and port of the *App Service* should be added into the whitelist, so that after deployment action, the client side can still access to the back-end server.

4.6.4 Push to Cloud

The last step is to deploy the server folder into the *App Service* via deployment options provided by Azure. The option of local git is chosen for the deployment of this project[27]. Azure App Service will automatically create secure communication using HTTPS protocol and the local generated secret keys and certification is not recognized by the public as they are not from a certification authority. So the local HTTPS server is only built for local testing, and make sure it is not uploaded into the cloud web service. The local server built during development period used HTTPS protocol and it failed to be deployed into Azure due to this issue.

[27]Tutorial is available on source: https://docs.microsoft.com/en-us/azure/app-service/app-service-deploy-local-git. Accessed: 2018-09-02

Chapter 5

Evaluation

In this chapter, analyses of the project are discussed. Implementation of this project is first analyzed, then potential security vulnerabilities faced by this project and methodologies adopted are detailed. Lastly, two surveys revealed the user experience of this site.

5.1 Analysis of Implementation

In this section, the analyses are discussed from quantitative and qualitative perspectives.

5.1.1 Objectives Complement Analysis

A table lists the objectives of this project and the implementation levels is given in Table 5.1.

Original Goal	Implementation Level
Donor user authentication	Fully implemented. Third-party authentication feature added.
Rate charities for feedback	Fully implemented. Make Comments and add to favorites are also added.
Search via name, categories and location	Fully implemented. Searching on map is also available.
Share donation information on social media	Fully implemented. Sharing charities profiles is also available.
Donation through this portal and potentially check the donation	Implement donation using Stripe gateway. Still cannot track the money.
Charity Registration and charity genuineness check	Fully implemented.

Table 5.1: Evaluation of Features Implementation

As shown in Table 5.1, all the aims of this project are implemented except for checking the monetary donation, which is an optional one. New ideas were generated during the development period through discussion with my supervisor and friends.

5.1.2 External Work Analysis

Apart from the implementation of objectives for this project, several other points were taken into consideration, such as user experience, efficiency and security. In addition, functionalities like voluntary work and new features were created.

Responsive User Interface This website is accessed by users using a variety of devices with different screen sizes. To enhance the user experience of visiting this website, I took advantages of the BootStrap4 framework, and designed responsive views for all web pages. In this case, CSS styles and content will be adjusted according to screen sizes of devices.

Methods for Optimizing Visit Performance Several means are adopted for better user experience as follows:

- As a single page application, this website reuses the $header$ and $footer$ component, so the client only needs to update the main contents when navigating between different web pages. This will largely reduce the time of reloading an HTML file.
- Paginations are used in both server and client sides. Search results of charities and activities are all return in paginated JSON format. A severe delay will be caused when requesting entire data at once from a large dataset. Enabling pagination in APIs are adapted to minimize the response times and improve the user experience.
- For authentication of the users, JWT are signed by the server and returned to the client side. The stateless server does not store the session information and verifies the user by checking the request header with token information. So unlike server with sessions, no session queries are required which implies faster response time.

Creative works In addition to fully implementing the objectives set by the project, some original ideas and genuine practices have also been adopted.

- To encourage individuals to volunteer time and their talents, voluntary work functionalities are created in this project creatively. There is nearly no online volunteer registration portal on the market.
- Third-party authentication is added for donor user registration, which largely reduces the time of registration for users.

- Search features enable users to search the charities nearby and find them on the map. Not any currently related portal supports this feature.

- To enable users to give feedback to charities, comments and favorites features are applied in addition to requested rating functionality.

5.2 Web Security Analysis

There are comprehensive kinds of security threats to a web application. In this section, practices used in this project will be covered from the Angular application and Express server perspectives [12].

5.2.1 Client-side Security Practices

The client-side of the website is vulnerable as it is directly open to the public users. Security risks such as Cross-site scripting(XSS), Cross-site Request Forgery etc. are common in the web security. To address such security vulnerabilities, several practices are used as follows.

Implement Content Security Policies(CSP) CSP is a computer security standard widely supported by modern browsers. It declares approved origins of resources that browsers are allowed to load. By doing so, attacks such as XSS, CodeInjection and other unsecured sources will be prevented. To implement the CSP in the Angular application, the $meta$ tag is defined in the *index.html* file. Specifically, the approved origins of sources used in this project such as scripts, styles, fonts and images are declared. In addition, the connect sources domains are also defined such that the client can only request APIs from the approved domains on this website.

Anti-CSRF Cross-site Request Forgery(CSRF or XSRF) is the attack that hackers steal the session data containing authentication information and make unsafe APIs invocation or form submission.

To address such issues, JWT-based Authentication is implemented. Authentication JWT token sent by the server is not stored in cookies, as it may suffer from the vulnerability, CSRF. A token stored in the cookie is easily be hijacked by hackers via a malicious link. Instead, the token signed by the server is sent back in the HTTP response body. Then it is stored on Local Storage in the *Angular* application. For every afterward request to the server, JWT is included in the request header by HTTP Interceptor. In this project, there are two types of tokens; one is donor user JWT token, another one is charity user JWT token, so the Interceptor should check if the local storage contains any of them. If there are any then intercept the token information in the following requests [8].

Validate Users' Submitted Data Untrust data such as malicious JavaScript sent back to the server can result in XSS attack. Without data validation or sanitization, the application can be attacked easily. *Angular* HTTP utility provides out-of-box support for dealing with this issue by *HttpClient* and *DomSanitizer*. Angular will automatically escape the values that are untrusted. In addition, all the form values created in the client application have their type definition to take advantage of *TypeScript*, so the value input by attackers will be validated and sanitized.

Other Good Practices Some good practices[1] recommended by Angular are used:

- Using Angular template instead of using DOM APIs such as *Document*, *ElementRef* as they are not protected by the Angular framework.
- Provide Interfaces for HTTP responses. All the HTTP requests in this application are all with response defined. So all the return results will be converted into corresponding interfaces.
- The Angular and all the dependencies used in the client-side are up to date.

5.2.2 Server-side Security Practices

Server-side protection is not only about the API security, but also involves database safety. This section illustrates the works used to protect the server and database.

NoSQL Injection In this project, *MongoDB* is used as the database, so the traditional SQL injection is not valid in this project. However, the NoSQL injection is still a threat. To address such security risks, *mongo-sanitize* package is used in Express server which replaces the sensitive keywords like $ sign in the *MongoDB* and overcome query selector injection attacks:

HTTP Headers Security As recommended by Express Official Security Document [2], $Helmet$[2] is a good third-party package which will help sets a collection of security-related headers in Express server, which include Content Security Policy, XSS protection, browser DNS prefetching etc.

HTTPS Server Secure communication is built using Hypertext Transfer Protocol Secure (HTTPS) Protocol which is an extension of the Hypertext Transfer Protocol (HTTP).

[1]Source: https://angular.io/guide/security. Accessed: 2018-09-02
[2]Source: https://helmetjs.github.io. Accessed: 2018-09-02

Cross-Origin Resource Sharing (CORS) Web application uses Same-Origin Policy which requires that all the resources are from the same origin. But cross-origin HTTP requests are required for some scenarios. In the server-side, Express server add CORS verifications for every API defined. There are two types of CORS configurations, for the *GET* request, the server allows all domains, but for the other request methods, such as *PUT*, *POST* or *DELETE*, only the origin of this application domain is allowed. This is because the *PUT*, *POST* or *DELETE* methods will directly perform operations on the database, so only the requests from the client domain are allowed.

Authentication Security Charity users or donor users' account information is stored in the *MongoDB*, but the passwords of the users are not directly stored. Two other fields, *salt* and *hash*, is saved instead. This is implemented so that the user accounts information will not be decrypted even the hacker cracked the database. For some APIs which returns or modifies data according to users, authentications are required. For example, if the user would like to check her or his favorites lists on this website, the *POST* request will first verify the user and then handling the database query. This will also improve the security of data.

5.3 User Experience Survey

A survey[3] based on the user experience of this website was designed and published. One version of the website was released first and then an improved version was then given back to users based on previous users' feedback.

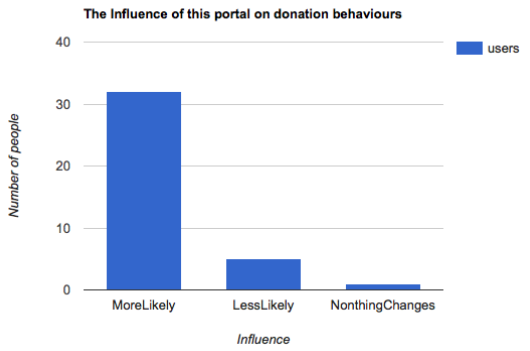


Figure 5.1: The Influence of This Portal on Donation Behaviors

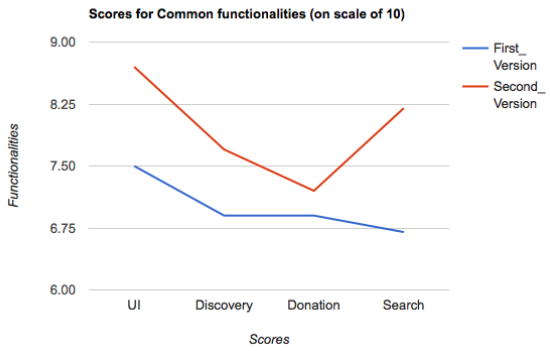


Figure 5.2: Scores for User Experience in Common Functionalities

In the first survey, 38 people took this survey, of which 8 are charity representatives and 30 are donor users. 32 (84%) of them think this kind of the portal will improve the probabilities of giving by them as shown in Figure 5.1. While in the second survey, 19 people took this survey (17 donor users, 2 charity users). For the common

[3]Available on source: https://imperial.eu.qualtrics.com/jfe/form/SV_9Z72QcWuHjVKjZz. Accessed: 2018-09-02

features that users can experience such as *UI*, *Discovery*, *Donation*, and *Search*, the average score (out of 10) is depicted in Figure 5.2. After the first release, responsible UI design in mobile devices was improved. More charities were signed up based on the feedback so that users can get more search results. After donation, users can share their donation information on social media. In the second survey, the user experience in all functionalities was improved. Valuable feedback on the donation section, like more payment methods and track donation, will be planed in future work.

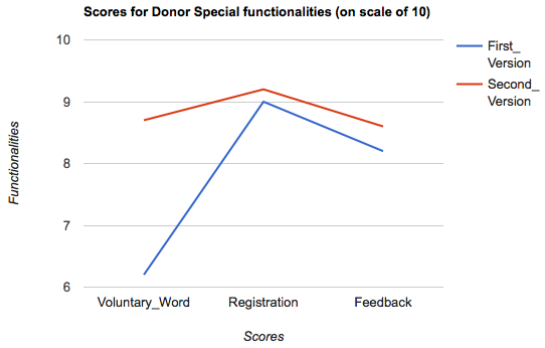


Figure 5.3: Scores for User Experience in Donor Special Functionalities

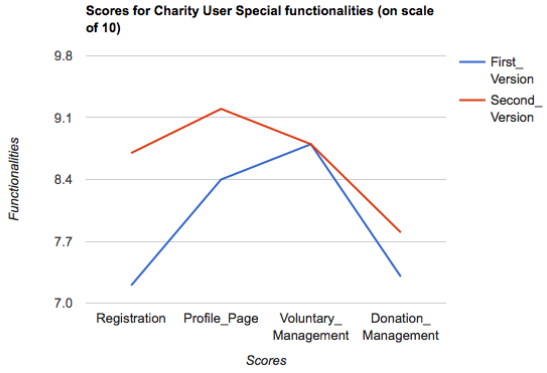


Figure 5.4: Scores for User Experience in Charity User Special Functionalities

For specific donor user experience, the survey results are displayed in Figure 5.3. Most of them like the user registration and management because the registration is easy and convenient via both account or *Facebook*. *Voluntary work* functionality did not perform well in the first version as there was very few *voluntary work* to search (only 5 at that time). The *comment* section was updated based on the feedback which enabled users to see comments after their posting. After creating more *voluntary work*, and add more details of each voluntary activity, the second version of this portal saw a rise in donor user experience.

Eight participants[4] in the survey experience the charities' journeys. In the first version, the registration experience for them was not quite good so as donation management. After adding tooltips for some terms in charities and add the preview feature after charity sign-up, the registration experience was improved. In addition, I also added pagination and the total amount of money for donation history based on the feedback. The comparison of user experience is in the Figure 5.4.

As shown in the Figure 5.5, the *voluntary work*, and *discovery* and *UI* of this portal are liked most by users, while *donation* and *search* are less. Searching charities based on different criteria were required by users. They also wish this portal to support different online payment and currencies. All helpful ideas were sincerely considered and will be added to functionalities in the future.

[4]As I mentioned in the survey, they do not be necessarily actual charities' representatives

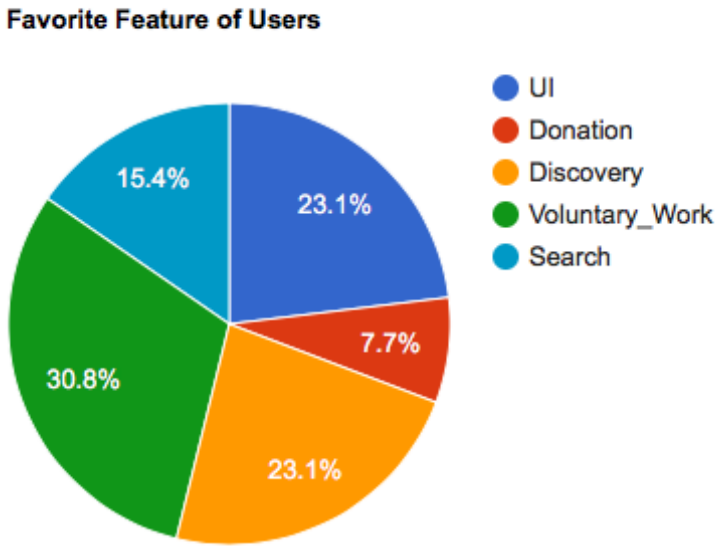


Figure 5.5: Favorite Feature of Users

5.4 Testing

Test-driven development strategy is applied during development. For every Angular component, unit tests were written before implementation. Apart from that, I also perform integration and end-to-end tests in the Angular application, which covers a large group of module interactions. The APIs created by the server is tested during developing using *PostMan*[5]. Unit tests are applied in both server and client application. All the test reports are available in Appendix F.

5.4.1 Testing Software

1. *Jasmine*[6] was used for unit testing in the Angular application.

2. The coverage report for Angular application is powered by *Karma*[7], a built-in Node.js application for running tests in the browser.

[5]Source: https://www.getpostman.com. Accessed: 2018-09-02
[6]Source: https://jasmine.github.io. Accessed: 2018-09-02
[7]Source: http://karma-runner.github.io/2.0/index.html. Accessed: 2018-09-02

3. End-to-end testing for the client-side is supported by *Protractor*[8], a test framework for Angular.
4. JavaScript unit tests in the Express server were written using *Mocha*[9] .
5. The coverage report was generated using *Istanbul*[10] which is integrated with Mocha.
6. Postman was utilized to test APIs created by the server.

5.4.2 Front-end Testing

Front-end is built via the Angular framework, which was built from different components as discussed in Chapter 3. Although unit testing is exceptional for examining isolated components, it does not support testing interaction between different units. Unlike small test coverage scope, end-to-end testing is a methodology used to test the entire application and cover several module interactions. In this section, testing is carried out by unit tests and end-to-end tests. Angular provides built-in support for these two testing strategies.

Unit Testing Angular framework uses the *Jasmine* testing framework to perform unit tests [1]. When an Angular unit is created such as component or service using the *Angular CLI*, the test file is created as well. Unit tests are applied in *Component*, *Service*, and *Routes* for this project. The test cases for each Angular units are listed below. After all the test cases are defined, the test environment is driven by *Karma* in Angular [15].

- **Component Tests** To test a component, only the UI and its auxiliary functions are tested. As is discussed in Chapter 4, normally a component is dependent on other services which provide data for rendering. The fake data would be provided manually and the services will be tested separately. $createComponent()$ method provided by *Angular* is used to first create the component. Following this, some important cases in this project test the form value validation, the *click* action trigger and return types. In addition, functions like $getLower()$ and $getUpper()$ are also tested to make sure the search results index is correct.
- **Service Tests** Eleven services are created in this project. Each service communicates with the server to fetch or modify data. For all the HTTP requests, a *MockBackend*[11] approach is used. First, the client request *MockBackend* using *POST* method, then verify the result using *GET* method. For some services that enable *PUT* or *DELETE* request, the same methodology is applied. After the *PUT*, *DELETE* requests are sent, check the result is as expected.

[8]Source: https://github.com/angular/protractor. Accessed: 2018-09-02
[9]Source: https://github.com/angular/protractor. Accessed: 2018-09-02
[10]Source: https://istanbul.js.org. Accessed: 2018-09-02
[11]Source: https://angular.io/api/http/testing/MockBackend. Accessed: 2018-09-02

- **Routes** There are twenty routes for this Angular application(Appendix E, the initial route, and illegal routes are all redirected to home and if the route is provided correctly, it will be directed to the corresponding component. The tests import *RouterTestingModule* to test the correctness of router navigation.

End-to-End testing End-to-end testing is designed to ensure that the integration of different components and modules is working functionally. In this project, the communication between client-side, server-side and database is tested.
Protractor, a node package for end-end-end tests, is built in the Angular framework using *AngularCLI*. It has features as follows:

- Run web application in a browser automatically using *Selenium*[12] browser automation framework.
- Provide direct controls of DOM elements
- Use *Jasmine* test syntax

Integration tests are performed from three perspectives:

- Following the unauthorized user's journey: test the search, discovery and donation features of this portal.
- Following the authorized donor user's journey: test the user sign-in, sign-up, profile, and voluntary works management as well as the feedback feature.
- Following the charity user's journey: test the functionalities such as authentication and charity management and voluntary works creation, modification etc.

5.4.3 Back-end Testing

In this project, the back-end server provides several APIs that perform operations on *MongoDB* collections. I utilized unit tests using *Mocha* on the mongoose model to test the operations on the database. And the different endpoints of server APIs are tested by *Postman*. The coverage of the unit tests on *MongoDB* is supported by *Istanbul*.

Database Unit Testing I tested eleven out of thirteen collections of the database except for the two externally imported collections. The strategy used for the unit test for each collection is as follows:

1. Connect to the local database and create a new collection
2. Test Create and query the collection to check expectations out.

[12]Source: https://www.seleniumhq.org. Accessed: 2018-09-02

3. Test Update and query the collection to check expectations out.
4. Test Delete and query the collection to check expectations out.
5. Drop the newly created collection.

API Testing *Postman* is a prevalent API testing tool that used to test the server. During the development, the development of client-side and server-side is isolated thanks to the *Postman*, which provided powerful API testing functionality. For all the APIs (Appendix C), *Postman* supports all kinds of request methods. For *GET* requests, check the results of expectation and the response status. For *POST* requests, define the request body and then check responses. For the request that requires authentication verification, add the JWT token to the request header and continue the request. *Postman* also provides form-data format request body to upload images to the *uploadRouter* in this project.

5.4.4 Load Testing

Azure provides performance tests for web services. This is used to test the efficiency and scalability of web applications. I conducted performance tests with user loads of 100, 250, 1000 and 5000. The average response time of this portal under different user loads is quick due to the SPA and other methods used to improve the performance as shown in Figure 5.6.

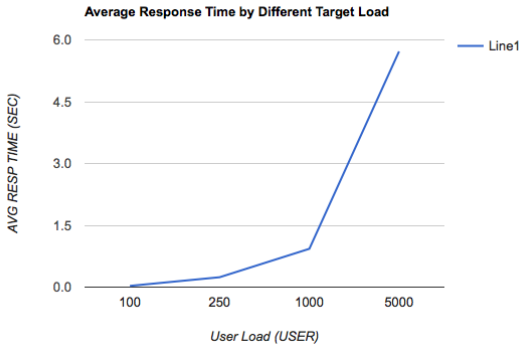


Figure 5.6: Average Response Time by Different Target Loads

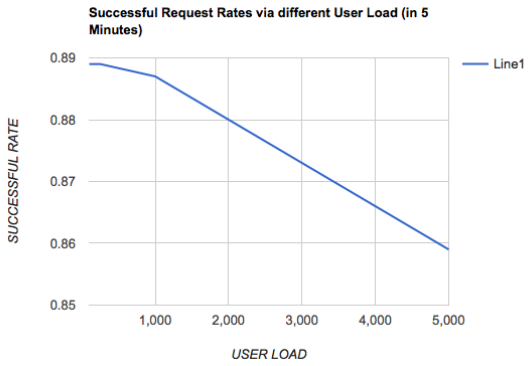


Figure 5.7: Successful Request Rates via Different User Loads in 5 Minutes

The successful request rates of this portal remain above 85% even under the user load of 5000 in 5 minutes. This means this portal is scalable. While there is an issue that the success rate is not high even under low user load (100 user load). This may result from some unstable API in this portal which will be examined in the future work. The line charity of success rate under different user load is in Figure 5.7.

All test reports are presented in the Appendix F.

5.4.5 Lessons Learned from Testing

Testing is essential for this project. When developing the APIs in the server, I used *Postman* to test the APIs, and check the results. On the front-end side, the tests were first written and all the codes are written to pass the tests. After deployment, the website cannot request the database correctly. To figure it out, I requested the API using the *Postman* and found request errors and no data returned as expected. Finally, I found that some advanced operations in *MongoDB* are not supported by Azure *CosmosDB*, so I rebuilt the API using the Azure styles. When doing the integration tests, I also found some logical errors in the project, for example the activities' time slots which are expired can also be signed up to. It is common to make mistakes, and testing is an optimal way to solve the bugs.

Chapter 6

Conclusion

6.1 Summary of Achievements

Considering all the specifications listed above, this project builds a platform for users in order to help associate donors with charities all over the world and overcome the limitation of geography. As a small step, I hope it can make a contribution in promoting and driving the act of giving by both donation and volunteering.

In addition to the contributions above, this platform is also equipped with some inventive features. The most significant one is the voluntary section which enables users to register to activities through this portal. Also, users can find charitable organizations on the map and check their profiles. In order to entitle donors to give feedback, the comments section is created. Once users find loved charities, they can add them to favorites for a quick find.

There are three types of target users: charities, donors and potential user. Each group of users can gain diverse experiences by utilizing this website, so the description of contributions will be presented below based on their purposes.

Charity Users Charity users can post their charities' information such as promotional photos, donation details, location and so on and post volunteer activities. In addition, they can also edit their charities' details page on this portal and track donations and voluntary registrations.

Donors Donors can register on this website by creating an account or Facebook authorization. As an authorized user, they can give feedback to charities by ranking, saving to favorites and making comments. They can also sign up to volunteer services and make monetary donations to charities they wish. In their donation management pages, they can see the amount of money they have donated and details of each donation. Similarly, voluntary registration details can also be managed through the voluntary management page.

Unregistered Users The potential users surfing the website can also make a donation without registration. They are able to search the nonprofits by name, location or category. Once they are interested in one charity, they can browse its details. In addition, they can also search the charities based on geographical information, and find out charities nearby or on a certain location. Potential users are of great significance because we would like to cultivate the spirit of giving worldwide.

The summary contributions of this web page are described in Table 6.1.

<table>
<tr><th></th><th>Un-registered user</th><th>Donor user</th><th>Charity user</th></tr>
<tr><td rowspan="3">Authentication</td><td rowspan="3">/</td><td>Sign up/Log in an account in this website</td><td>Sign up/Log in an account in this website</td></tr>
<tr><td>Sign up/Log in via Facebook</td><td>Charity Verification & Charity preview</td></tr>
<tr><td>Check user exists or not</td><td>Check user exists or not</td></tr>
<tr><td rowspan="2">Search</td><td colspan="3">Search charities by name/location/category</td></tr>
<tr><td colspan="3">Find a charity nearby/on map</td></tr>
<tr><td rowspan="3">Voluntary work</td><td colspan="3">Find all and search voluntary activities</td></tr>
<tr><td rowspan="2">/</td><td>Register to voluntary work</td><td>Create/Update/Delete voluntary activities</td></tr>
<tr><td>Check all registered voluntary works</td><td>Check all created activities and all registered users</td></tr>
<tr><td rowspan="3">Feedback</td><td rowspan="3">/</td><td>Rate charities</td><td rowspan="3">/</td></tr>
<tr><td>Add to favorites</td></tr>
<tr><td>Make Comments</td></tr>
<tr><td rowspan="2">Donation</td><td colspan="2">Make donations and share on social media</td><td>/</td></tr>
<tr><td>/</td><td colspan="2">Check donation history and total amount of money</td></tr>
<tr><td rowspan="3">Profile</td><td colspan="3">See all non-profits' profiles and share on social media</td></tr>
<tr><td>/</td><td>Update profile info and image</td><td>Update profile figures and information</td></tr>
<tr><td></td><td>Update a new password</td><td>Update a new password and preview charity detail page</td></tr>
<tr><td colspan="4">URL:https://webcharity.azurewebsites.net</td></tr>
</table>

Table 6.1: Contributions Lists

For the design choice, there are three parts of the project, front-end, back-end, and database. Angular and Bootstrap frameworks are used for front-end, and Express

framework is adopted in the server side. As for the database, NoSQL database is utilized, *MongoDB* specifically.

6.2 Future Work

In this section future work is discussed. Improvements to this project are planned by developing the existent functionalities and adding more functionalities. Those planned works are inspired by the feedback of user experiences from the survey conducted. Also, some imperfect technical implementations because of time limits or knowledge shortage will be covered.

Payment Methods For now, donors can only make donations by credit card through Stripe gateway. A comprehensive collection of online payment methods such as *PayPal*, *Alipay*, *Wechat Pay*, *Apple Pay* and so on will be supported in the future.

Donation Tracking The main concern for donors is that charities misuse donated funds. This platform will support donation tracking functionality. This can be implemented simply by letting charities upload financial reports regularly and sending emails or messages to all their donors. Another creative way to implement this is to enable cryptocurrency payment in this portal, and all the donations can be check by anyone, anytime, anywhere.

UI Design The User interface of some pages is still simple, so they will be redesigned and beatified. In addition, although the UI design is responsive, some page in the mobile platforms are still not user-friendly so more attention will be paid to this problem.

Voluntary Work Search In this stage, users can only search the voluntary by date. Voluntary work search by keywords will be implemented just like the charity search section.

Map Search Section Improvement For now, when a user searches charities on the map, the server will return the all the charities data to the client for map rendering. This is not scalable when the dataset is large. To solve this problem, only the charities locating within an area will be returned and the results will be adjusted when users change the scope of the map.

Cross-Platform Mobile applications play a more and more significant part in people's daily lives. Mobile applications are more suitable for users that frequently use this portal. As an Angular web application, it is easy to be modified to build an IOS/Android application without web views using NativeScript[1].

[1]Source: https://docs.nativescript.org. Accessed: 2018-09-02

Web Security More attention should be paid to web security. For example, *csurf*[2] middleware can be used to protect the server from cross-site request forgery (CSRF). As the regular expressions are used in the *MongoDB* query in this project, $safe{-}rege$[3] can be utilized to protect them from regular expression denial of service attacks.

Marketing Promotion and Language Support To truly link charities and donors all over the world, more languages and currency supports will be required. It is necessary to promote this website to more charities and attract more donors.

New Features Recommendation features in the home page will be based on the donors' donation behaviors to make recommendations using machine learning techniques. Charity users can also upload videos to promote their charity and reply to comments.

[2]Source: https://www.npmjs.com/package/express-csurf. Accessed: 2018-09-02
[3]Source: https://www.npmjs.com/package/safe-regex. Accessed: 2018-09-02

Bibliography

[1] Angular testing. https://angular.io/guide/testing. Last accessed: 03/09/2018.

[2] Production best practices: Security. https://expressjs.com/en/advanced/best-practice-security.html. Last accessed: 02/09/2018.

[3] Server-side web frameworks. https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks. Last accessed: 28/08/2018.

[4] What is database? what is sql? https://www.guru99.com/introduction-to-database-sql.html. Last accessed: 28/08/2018.

[5] What is database? what is sql? https://www.guru99.com/introduction-to-database-sql.html. Last accessed: 28/08/2018.

[6] Using stripe with angular. https://blog.mgechev.com/2016/07/05/using-stripe-payment-with-angular-2, 2016. Last accessed: 02/09/2018.

[7] Dynamic website. https://www.computerhope.com/jargon/d/dynasite.htm, 2017. Last accessed: 03/09/2018.

[8] Angular security - authentication with json web tokens (jwt): The complete guide. https://blog.angular-university.io/angular-jwt-authentication/, 2018. Last accessed: 02/09/2018.

[9] Samuel Andras. Javascript frameworks, why and when to use them. https://blog.hellojs.org/javascript-frameworks-why-and-when-to-use-them-43af33d0608d, 2017. Last accessed: 28/08/2018.

[10] Laurence Bradford. Front-end framework in web development. https://www.thebalancecareers.com/what-is-a-front-end-framework-and-why-use-one-2071948, 2018. Last accessed: 28/08/2018.

[11] Charities Aid Foundation. World giving index 2017, 2017.

[12] AJITESH KUMAR. Angular top 10 security best practices vis-a-vis security risks. https://vitalflux.com/

angular-top-10-security-best-practices-vis-vis-security-risks/, 2017. Last accessed: 02/09/2018.

[13] Lukas Marx. Is angular 2+ mvvm. https://malcoded.com/posts/angular-2-components-and-mvvm, 2016. Last accessed: 28/08/2018.

[14] Justyna Rachowicz. When, how and why use node.js as your backend. https://www.netguru.co/blog/use-node-js-backend, 2017. Last accessed: 28/08/2018.

[15] Gerard Sans. AngularŁŁtesting guide (v4+). https://medium.com/google-developer-experts/angular-2-testing-guide-a485b6cb1ef0/, 2016. Last accessed: 02/09/2018.

[16] John Sonmez. What is back-end development? https://simpleprogrammer.com/what-is-back-end-development, 2016. Last accessed: 03/09/2018.

[17] Serdar Yegulalp. What is nosql? nosql databases explained. https://www.infoworld.com/article/3240644/nosql/what-is-nosql-nosql-databases-explained.html, 2017. Last accessed: 28/08/2018.

Glossary

APIs Application Programming Interface. 9, 15, 17, 18, 24, 31, 32, 36, 37, 50, 51, 55, 65–68, 70–72, 74

BSON Binary JSON. 15

CORS Cross-Origin Resource Sharing. 63, 68

CRUD Create, read, update, and delete. 16

CSP Content Security Policies. 66

CSRF Cross-site Request Forgery. 66

CSS Cascading Style Sheets. 9–11, 13, 18

DOM Document Object Model. 13, 67, 72

ER diagram Entityrelationship diagram. 28

HTML Hypertext Markup Language. 9–11, 13, 17, 18

HTTP Hypertext Transfer Protocol. 15, 17, 66–68

HTTPS Hypertext Transfer Protocol Secure. 63, 67

JSON JavaScript Object Notation. 15, 35

JWT JSON Web Token. 43, 65, 66

MVC Model-View-Controller. 11

MVVM Model-View-ViewModel. 12

NoSQL databases Non-relational databases. 15

ODM Object Document Mapping. 16

REST Representational State Transfer . 9, 15, 18

SPA Single-page Application. 10, 20, 73

SQL Structured Query Language. 14, 15, 17

UI User Interface. 10, 11, 17, 31, 77

URIs Uniform Resource Identifiers. 15, 37

XML Extensible Markup Language. 15

XSS Cross-site scripting. 66, 67

Appendices

Appendix A

Ethics

A.1 Checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?	✓	
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.	✓	
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?	✓	
Section 5: ANIMALS		
Does your project involve animals?		✓

Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?	✓	
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?	✓	
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
Section 10: LEGAL ISSUES		

Will your project use or produce software for which there are copyright licensing implications?	✓	
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

A.2 Explanation

This project involves data collection for two types of users. None of the personal data obtained is sensitive. For the donors, the system will collect personal information such as email address, name and country of residence if they create accounts on this website. If donors registered using Facebook authentication, Facebook ID will also be recorded as well as email, country, first name and last name. Bank account information is not collected for donor users, and all the transactions are through Stripe gateway. This website does not save donors' bank details. As for charity users, more data is collected. In particular, three main kinds of information is collected, which are charity details, personal details and donation details. The charity details collected in this project are public information. Personal details collected for charity users are first name and last name. The donation details contain bank account information (and PayPal accounts if any), which should also be public information for charities. Under the EU General Data Protection Regulations (GDPR) scope of personal data, only the information collected for donor user registration is personal data, as the name and email address can narrow down to identify. As for charity user, only first and last names are collected, and it is not personal data because name alone cannot be used to identify someone[1].

Geographical information of charity is collected and processing by the Google Map API, to get Geocoding information for map searching. Only this data is further processed in this period. In the future, the donations trend may be analyzed.

This site will ask for a permission of localization information if the web visitor would like to search the map for charities, as it will initiate the map based on the location of the browser. The user can decide to not grant the permission and also can use this functionality. The location data will not be processed or saved by this website.

As a platform for people and charity organizations all over the world to use, charities from less developed countries are our principal target users as more technical and financial assistance is demanded by them. After their registration, more awareness and donations will be gained through this portal, so that they can receive more monetary and voluntary donations. In addition, this website can cultivate the habits of

[1]Source: https://www.itgovernance.eu/blog/en/the-gdpr-what-exactly-is-personal-data. Accessed: 2018-09-02

giving to people all over the world.

All the images and icons used for the user interface design are free to use. The cover images are download from $pixabay$,[2], which is a website that provides free images. In this website, it states:

> You can copy, modify, distribute, and use the images, even for commercial purposes, all without asking for permission or giving credits to the artist.

The icons used for this website are from $fontawesome$[3], and are all free to use. The logo of this website is download from an icon font website[4], and is design by LuoLuoDeXiaTianNe in English. I have contacted him by email for the use of it for this individual project purpose.

Copyrights are reserved for this project. I will respect and protect users' privacy and do not sell or provide users' information for any purposes.

[2]Source: https://pixabay.com/. Accessed: 2018-09-02

[3]https://fontawesome.com

[4]Source address: http://iconfont.cn/search/index?q=%E6%85%88%E5%96%84. Accessed: 2018-09-02

Appendix B

User Guide

Users can gain access to this website[1] via any devices with internet access.

B.1 Main structure

There are six main sections in this website, namely Homepage, Discover page, Volunteer page, About Us page, Search page and Contact page.

In the homepage, the upper part (Figure B.1) of the website is a jumbotron of images. This is designed for users to quickly catch the themes of this website and find meaningful activities or charities. Users can also search directly by the search bar in the homepage.
Down on the homepage as Figure B.2, it shows the latest 8 charities with a charity information card that has been created on this website. It is helpful to make the new charities easily found by potential donors or volunteers. Each information card contains essential charity information and is the quickest way to make donors aware of the charity. It contains the name, location and brief information about the charity. Once a mouse is hanging over, it will show the charity category and you can click the read more to see more details of the charity. In addition, users can directly click the donate icon to donate money to the charity without login and click to save the charity to the favorite when the user is logged in, as well as the number of comments

[1] https://webcharity.azurewebsites.net

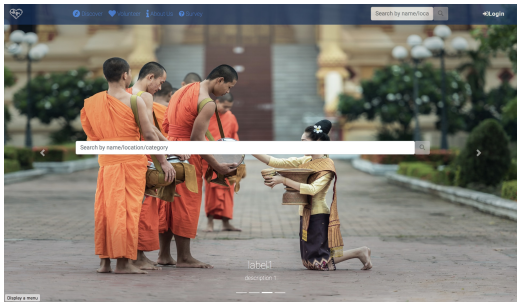


Figure B.1: Home Page 1

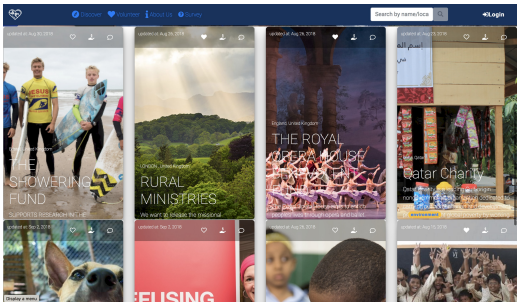


Figure B.2: Home Page 2

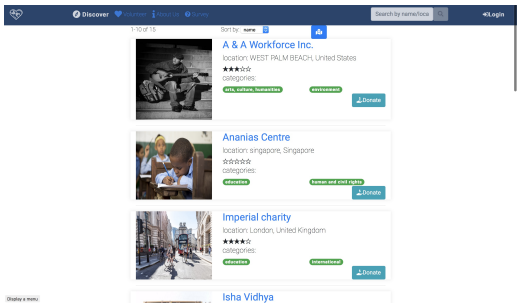


Figure B.3: Discovery Page

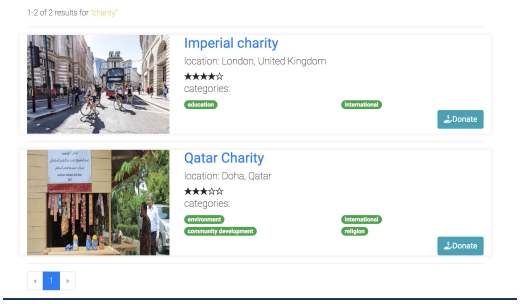


Figure B.4: Search Page

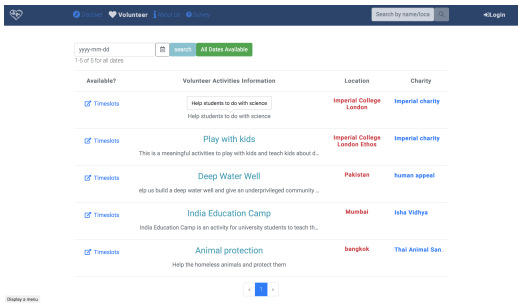


Figure B.5: Volunteer Page

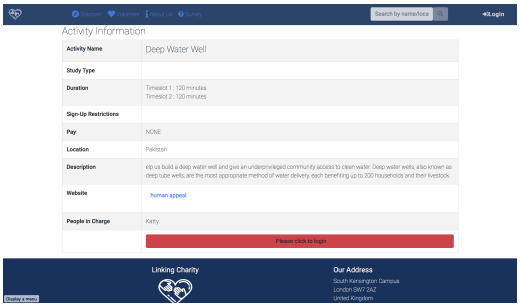


Figure B.6: Voluntary Work Detail Page

for this charity.

In the discover page as depicted in Figure B.3, the user can find all the charities registered on this website. Pagination technique is used; every page contains 10 charity cards and a user can choose between different pages. In each charity card, it contains a promotional image and basic information of the charity such as charity name, information, average ratings and categories. Users can click on the charity name to see more details if interested. Also, they can click the donate button to donate to the charity directly. If they are interested in one category of charities, they can click the category label, and the website will search the charities of that type. In this page, a user can sort the charities by name, geographical location or by rating (Figures are available in Chapter 4).

Once keywords are searched by users, the website will navigate to the search page (see Figure B.4). If no results are returned, the website will respond with no results. Otherwise, the search page will return the search results information. All results are paginated, each page contains 10 charities information. The page information is displayed on the top of the page, such as the index of the charities and the total number of results for the search keyword.

On the volunteer page (Figure B.5), users will find all the available volunteer activities. Once a volunteer activity has no available time slots, it will not be shown on this page. Similar to the search page, the volunteer page will return 10 volunteer activity records, and each will contain basic information of the activity, such as posted charity, location, activity name and a button to find available time slots. A

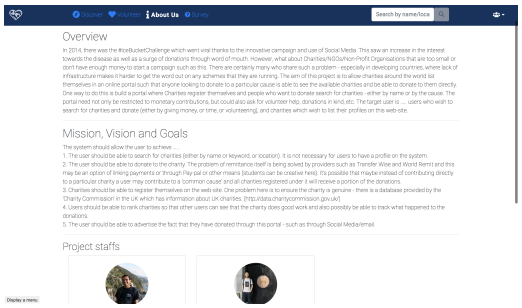


Figure B.7: About Us Page

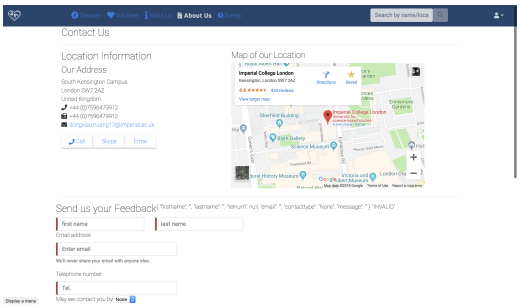


Figure B.8: Contact Us Page

user can click the post charity button to find more volunteer activities posted by it and click the name and time slots button to find out more about the details of the activity (Figure B.6). In the activity detail page, once the $ViewTimeSlots$ button is clicked, the user can see all the available time slots for this activity, and choose to register if this time slot still needs more volunteers. Users are able to find out the volunteering times on a specific day by clicking the calendar button at the top of the page and choosing the day. If any activities are available, the page will return results or it will show no results.

In about us page as shown in Figure B.7, users can find an overview explanation of the website. The mission, vision and goals of the project can also be found. In addition, a user can find the information about the project staff. In the contact page (Figure B.8), users can gain information on contact details and send feedback to us.

B.2 For Donors

Potential users can donate to registered charities without sign up on this website. But if a user would like to keep a log of the donation history and make comments about charities as well as sign up for certain volunteer activities, the user should sign up as a donor.

A donor can click the login button in the header of the website, and log in via a registered user account in the donor tab or login via Facebook authentication. If a user does not have an account, he or she can create an account by clicking $JoinUs$ button in the bottom of the pop-up model.

On the register page, choose the $registerasadonor$ tab, and fill the basic personal information. And click submit, it will show register successful message. After click close, it will redirect to the home page. All the pictures for this procedures are in Chapter 4. Then the user can log in by the account just created.

Once logged in, the login button in the header will turn into a user icon. Once the icon is clicked, a drop down menu will show up. There are five options; favorites, profile, donation history, volunteers management and log out. Users can

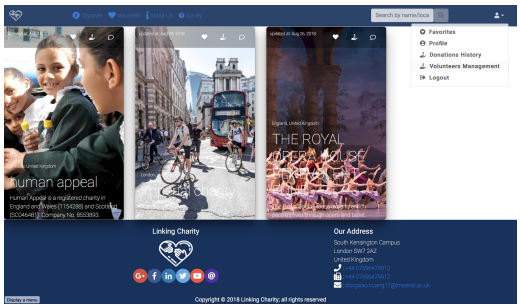


Figure B.9: Favorites Page

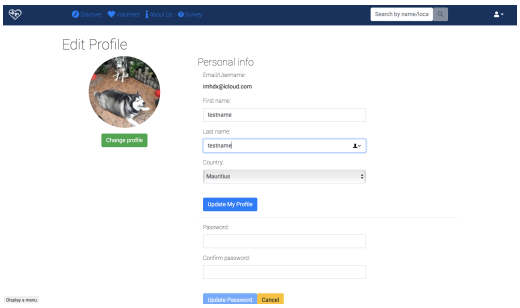


Figure B.10: Donor Profile Page

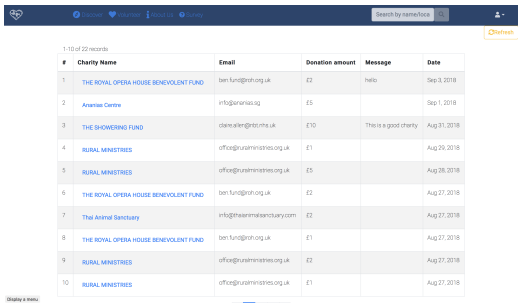


Figure B.11: Donor Donation History Page

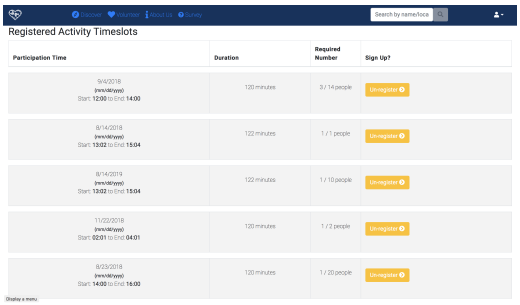


Figure B.12: Donor Voluntary Work Management Page

click the favorites option to see all the saved favorite charities (Figure B.9). When the profile option is clicked as shown in Figure B.10, the user can edit their personal details and upload a profile image. If the user clicks Donation history (Figure B.11), the page will show all the donation transactions made by the user. In the $VolunteersManagement$ page (Figure B.12), the user can find all the volunteer activities signed up by him or her, and cancel anytime.

The login status will stay live for 2 days, so even if you refresh the web page, login status will stay. Once the user is logged in, he or she can sign up for a time slot of an activity as well as save charities as favorites. In a specific charity detail page, the user can make comments for the charity and make a rating for the charity.

B.3 For CharityUser

Charity Users need to sign up as a charity to manage and publish their charity on the website. When using for the first time, charity user should click login button in the header, and click $JoinUs$. In $JoinUs$, choose Register as a charity user, and fill in the charity details, personal details, and donation details and finally upload promotion images sequentially. After clicking the submit button, the server will return a status message for registration. Figures for this procedure are in Chapter 4.

After successful registration, charity user can click the $login$ button in the header, and choose login as a charity. Once successfully logged in, the login button in the

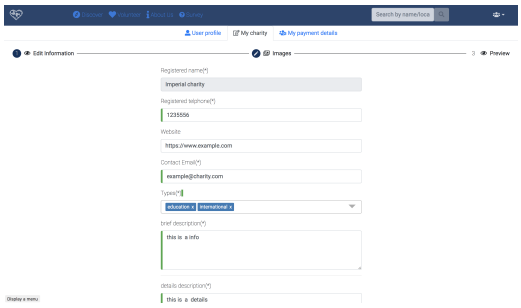


Figure B.13: Charity Profile Management Page

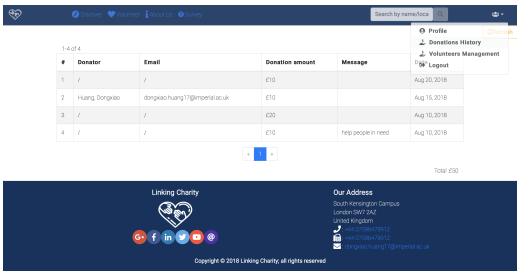


Figure B.14: Charity Donation History Page

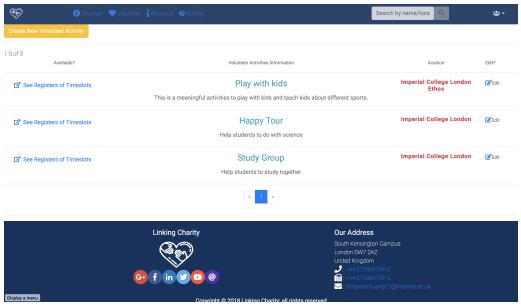


Figure B.15: Charity Voluntary Work Management Page

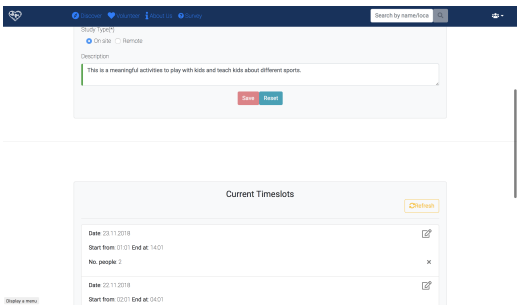


Figure B.16: Charity Volunteer Edit Page

header will be changed to charity icon. Charity users can click the icon and choose the options to manage the account. There are 4 options to choose from, namely profile, donation history, volunteers managements and log out.

In the profile page as depicted in Figure B.13, charity user can edit their charity details once anything is updated, as well as account information and donation account details. Charity profile preview is supported in this feature.

In donation history (Figure B.14), charity user can get access to all the history from donors. If the donor is a registered user in the web page, the record will show the donor's email address. If the donor donated anonymously, the record will only contain the message (if any) and amount of money for the donation. The total amount of money received will also be shown on the page.
To manage the volunteer activities created by the charity user, the user can click the **Volunteers Management** in the drop-down options. In the volunteer management page as shown in Figure B.15, charity user can see all the activities it has created, and click 'see registers of time slots' button, to check the participation status of each time slot and get all the participants' email addresses. If there are any updates of the activity, the charity user can click the edit button to go to edit the activity page.
In the edit activity page (Figure B.16), the information of the activity can be changed. Time slots of the activity can be added, deleted or changed.

Appendix C

APIs

C.1 Categories Router

Resource	GET read	POST create	PUT update	DELETE
/categories	Return all the categories for charities	Create new category for charities	Method not allowed (405)	Remove all the categories for charities
/categories/:categoryId	Return a category with id:categoryId	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table C.1: Supported APIs for Categories Router

C.2 Charity Router

Resource	GET read	POST create	PUT update	DELETE
/charities	Return JSON format data, containing information like current page, total pages, and a list of charities in current page	Create new charity	Method not allowed (405)	Remove all charities
/charities/allcharities	Return all charities	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

/charities/newCharities	Return 8 most newly created charities	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/charities/:charityId/	Return the charity with id: charityId	Method not allowed (405)	Update the charity with id: charityId	Delete the charity with id: charityId
/charities/:charityId/ -geocoding	Return the geocode of the charity with id: charityId	Create the geocode of the charity with id: charityId	Update the geocode of the charity with id: charityId	Method not allowed (405)
/charities/:charityId/ -comments	Return the comments of the charity with id: charityId	Post comment to the charity with id: charityId	Method not allowed (405)	Delete all the comments of the charity with id: charityId
/charities/ccn/ :regno	Return UK charity information if the registration number(regno) is valid and stored in UK charity database	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table C.2: Supported APIs for Charity Router

C.3 Charity User Router

Resource	GET read	POST create	PUT update	DELETE
/charityusers/signup	Method not allowed (405)	Create a new charity account	Method not allowed (405)	Method not allowed (405)
/charityusers/login	Method not allowed (405)	Log in with an charity account	Method not allowed (405)	Method not allowed (405)

/charityusers/logout	Log out current account	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/charityusers/check-JWTToken	Check JSON Web Token is valid or not	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/charityusers/profile	Return charity information	Method not allowed (405)	Update charity information	Method not allowed (405)
/charityusers/new-password	Method not allowed (405)	Method not allowed (405)	Update password	Method not allowed (405)
/charityusers/checkId	Method not allowed (405)	Check charity username exists or not	Method not allowed (405)	Method not allowed (405)

Table C.3: Supported APIs for Charity Users Router

C.4 Cover Picture Router

Resource	GET read	POST create	PUT update	DELETE
/coverpics	Return all cover pictures information	Create a new cover picture information	Method not allowed (405)	Delete all cover pictures information
/coverpics/:coverId	Return the cover pictures information (id=coverId)	Method not allowed (405)	Update the cover pictures information (id=coverId)	Delete the cover pictures information (id=coverId)

Table C.4: Supported APIs for Cover Picture Router

C.5 Donation Router

Resource	GET read	POST create	PUT update	DELETE

/donation	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of donation history for authenticated charity user in current page	Make a payment through Stripe API	Method not allowed (405)	Method not allowed (405)
/donation/user	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of donation history for authenticated donor user in current page	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table C.5: Supported APIs for Donation Router

C.6 Favorite Router

Resource	GET read	POST create	PUT update	DELETE
/favorites	Return all the favorite charities for a authenticated donor	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/favorites/:charityID	Return if the charity with id: charityID is one of favorites for the authenticated user or not	Add the charity with id: charityID to one of favorites for the authenticated user	Method not allowed (405)	Remove the charity with id: charityID from one of favorites for the authenticated user

Table C.6: Supported APIs for Favorite Router

C.7 User Router

Resource	GET read	POST create	PUT update	DELETE
/users/signup	Method not allowed (405[1])	Create a new donor account	Method not allowed (405)	Method not allowed (405)
/users/login	Method not allowed (405)	Log in with an account	Method not allowed (405)	Method not allowed (405)
/users/logout	Log out current account	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/users/checkJWTToken	Check JSON Web Token is valid or not	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/users/profile	Return donor information	Method not allowed (405)	Update donor information	Method not allowed (405)
/users/newpassword	Method not allowed (405)	Method not allowed (405)	Update password	Method not allowed (405)
/users/facebook/token	Sign in using Facebook Authentication	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/users/checkId	Method not allowed (405)	Check username exists or not	Method not allowed (405)	Method not allowed (405)

Table C.7: Supported APIs for Donor Router

C.8 Upload Router

Resource	GET read	POST create	PUT update	DELETE
/imageUpload	Method not allowed (405)	Post a profile image for a authenticated donor	Method not allowed (405)	Method not allowed (405)

[1]HTTP Response Codes:https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

/imageUpload/charitiesPics/	Method not allowed (405)	Post at most 3 promotional images and return images URLs	Method not allowed (405)	Method not allowed (405)
/imageUpload/charitiesPics/:chairtyId	Method not allowed (405)	Post at most 3 promotional images when charity registration	Method not allowed (405)	Method not allowed (405)

Table C.8: Supported APIs for Upload Router

C.9 Payment Details Router

Resource	GET read	POST create	PUT update	DELETE
/paymentdetails	Method not allowed (405)	Create a new payment detail information	Method not allowed (405)	Method not allowed (405)
/paymentdetails/:cardId	Return the payment detail with id:cardId	Update the payment detail with id:cardId	Method not allowed (405)	Method not allowed (405)

Table C.9: Supported APIs for Payment Details Router

C.10 Rating Router

Resource	GET read	POST create	PUT update	DELETE
/rating/:charityId	Return the rating for charity (id = charityId) made by an authenticated donor if any.	Make a rating for charity with id: charityId from an authenticated donor user	Method not allowed (405)	Method not allowed (405)

/rating/:charity-Id/averageRating	Return the average rating for the charity (id = charityId)	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table C.10: Supported APIs for Rating Router

C.11 Search Router

Resource	GET read	POST create	PUT update	DELETE
/search	Return JSON format data, containing information like current page, total pages, number of items in each page, and a list of charities searched by keywords in current page	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table C.11: Supported APIs for Search Router

C.12 Volunteer Router

Resource	GET read	POST create	PUT update	DELETE
/volunteer	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of voluntary activities that is available from today or on certain day in current page	Create a new voluntary activity	Method not allowed (405)	Method not allowed (405)

/volunteer/:volunteerId	Return the voluntary activity identified by volunteerId	Method not allowed (405)	Update the voluntary activity identified by volunteerId	Delete the voluntary activity identified by volunteerId
/volunteer/:volunteer-Id/timeslots	Return all the time slots for the voluntary activity with id:volunteerId	Create a new time slot for the voluntary activity with id:volunteerId	Method not allowed (405)	Delete all time slots for the voluntary activity with id:volunteerId
/volunteer/:volunteerId/timeslot/:timeslotId	Return the time slot with id:timeslotId for the voluntary activity with id:volunteerId	Method not allowed (405)	Update the time slot identified by volunteerId for the voluntary activity with id:volunteerId	Delete the time slot with id:timeslotId for the voluntary activity identified by volunteerId
/volunteer/:volunteerId/timeslots/get-registers	Return all the registers signed up for the voluntary activity with id:volunteerId	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/volunteer/user/volunteers	Return all the voluntary activities signed up by the authenticate donor user	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/volunteer/:volunteerId/timeslot/:timeslotId/register	Check if the authenticated donor user has registered to the timeslot identified by timeslotId of voluntary activity identified by volunteerId or not	Register to the time slot identified by timeslotId of voluntary activity identified by volunteerId	Method not allowed (405)	Unregister to the tim eslot identified by timeslotId of voluntary activity identified by volunteerId

/volunteer/charity/:charityId	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of **available** voluntary activities created by the charity identified by charityId in current page	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)
/volunteer/charity/activities/all	Return JSON format data, containing information like current page, amount of items per page, total pages, and a list of **all** voluntary activities created by the charity identified by charityId in current page	Method not allowed (405)	Method not allowed (405)	Method not allowed (405)

Table C.12: Supported APIs for Volunteer Router

Appendix D

Supplementary API MetaData

D.1 Donor Register

POST donor user information, user name and password, then create an encrypt user, returned registration status.

HTTP Request POST /users/signup

Input & Output Variables The input, output variables and explanations are illustrated in Table D.1.

Input	Donor user information and username and password in JSON format via request body	
Output	err	Error message if registration failed
	success	Flag to check if the registration is successful or not
	status	Registration message for successful registration

Table D.1: Input, Output Variables And Explanation for Donor Register

D.2 Username Check

POST donor user information, user name and password, then create an encrypt user, returned registration status.

HTTP Request POST /users/signup

Input & Output Variables The input, output variables and explanations are illustrated in Table D.2.

Input	username in JSON format via request body	
Output	err	Error message if registration failed
	exists	Flag to check if the user name exists or not

Table D.2: Input, Output Variables And Explanation for Username Check

D.3 Get Categries

Get detailed list of all charities' categories.

HTTP Request GET /categories

Output Variables The output variables and explanations are illustrated in Table D.3.

success	Boolean value to check if the request is successful or not
message	Error or success message for request
categories	A list of categories of Category Type in JSON format. Category type will be described with Category model in database section.

Table D.3: Output Variables And Explanation for Get Categries

D.4 Update Geocode Information for A Charity

Find a charity by its charity ID and update its geocoding field, and return the updated geocoding information.

HTTP Request PUT /charities/:charityId/geocode

Input & Output Variables The input, output variables and explanations are illustrated in Table D.4.

error	Error message if request failed
lat	Latitude of charity address
lng	Longitude of charity address

Table D.4: Input, Output Variables And Explanation for Update Geocode

D.5 Upload Images for A Charity

Upload promotional figures for requested charity.

HTTP Request POST /imageUpload/charitiesPics/:charityId

Input & Output Variables The input, output variables and explanations are illustrated in Table D.5.

Input	Image files in type of form-data with field key name: imageFile. Only images of types jpg, jpeg or png are accepted. At most three images are allowed.	
Output	urls	An JSON array of the URLs of the images uploaded.
	success	Boolean value indicates if the uploading is successful or not.
	message	Error message if the uploading failed

Table D.5: Input, Output Variables And Explanation for Image Uploading

Appendix E

Available Routers for This Website

Base URL : https://webcharity.azurewebsites.net/	
Router Link	**Description**
/aboutus	About us page
/charitydetail/:id	Charity profile page
/volunteerdetail/:id	Voluntary work detail page
/volunteerdetail/:id/registers	Charity voluntary works' registration page
/volunteerdetail/:id/timeslots	Voluntary works available time slots page
/contact	Contact us page
/discover	Discover page
/favorites	Favorites management page
/home	Home page
/signup	Signup page
/volunteer	Voluntary works discovery page
/volunteer/register	create new voluntary works page
/search;query=query;page=page	Search results page
/user/profile	User profile management page
/user/donations	User donation history page
/user/volunteer	User registered voluntary works management page
/charityuser/profile	Charity user profile management page
/charityuser/donations	Charity donation history page
/charityuser/volunteer	Charity voluntary works management page
/charityuser/volunteer/:id	Edit voluntary work page

Table E.1: Available Routers for This Website

Appendix F

Test Reports

F.1 Unit Tests for Front-end

Figure F.1: Client-side Unit Tests

F.2 End-to-end Tests for Client-side

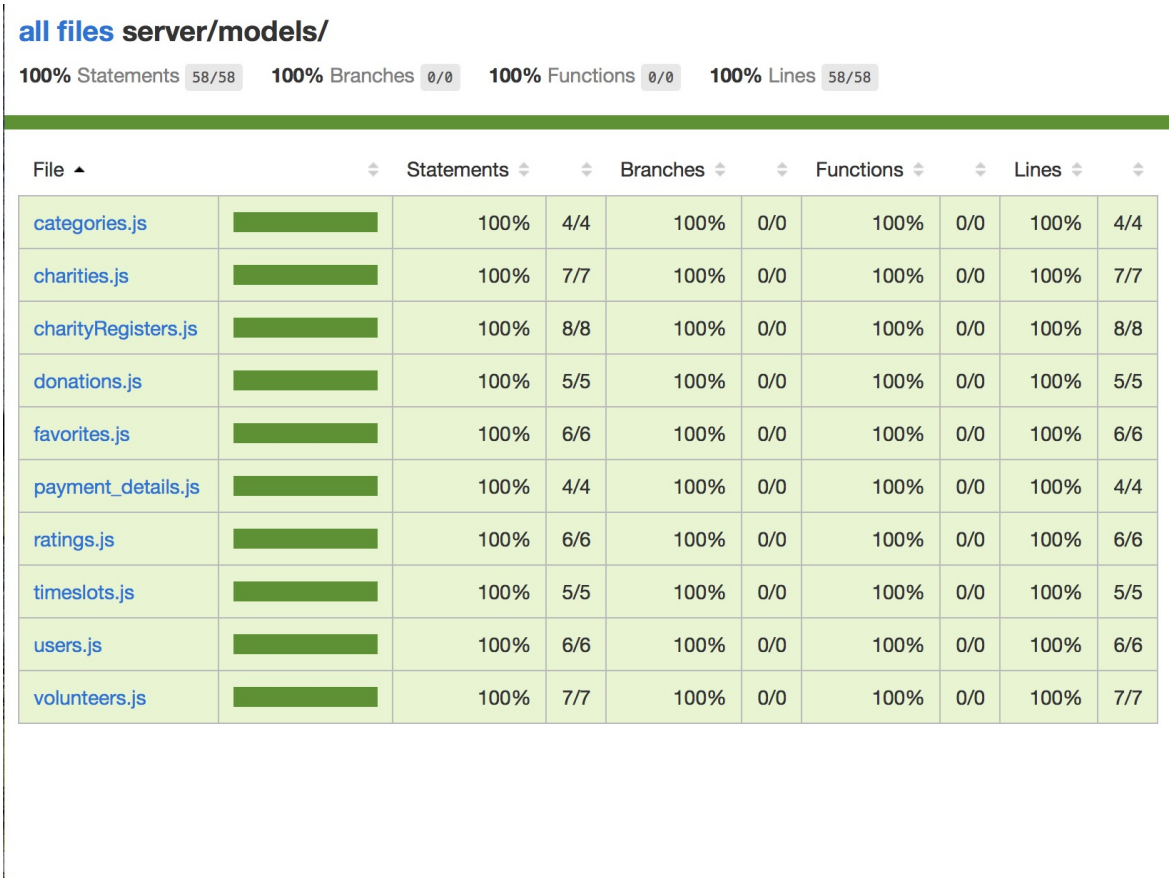
```
[22:30:51] I/direct - Using ChromeDriver directly...
Jasmine started

  workspace-project App
    ✓ Test in home page
    ✓ Test in Discovery page
    ✓ Test in Volunteer page
    ✓ Test in Aboutus page
    ✓ Test in Contactus page
    ✓ Test in Volunteer Detail page
    ✓ Test in Volunteer Signup page
    ✓ Test in Charity Volunteer check page
    ✓ Test in Charity Volunteer Detail update page
    ✓ Test in Charity Volunteer Detail update page
    ✓ Test in Charity Volunteer Detail update page
    ✓ Test in Charity Volunteer Detail update page
    ✓ Test in Donor Volunteer check page
    ✓ Test in Donor Favorite page
    ✓ Test in AGM page
    ✓ Test in Donation page
    ✓ Test in Search page
    ✓ Test in Signup
    ✓ Test in Profile Update page
    ✓ Test in Charity Profile Update page
    ✓ Test in Volunteer Management page

Executed 21 of 21 specs SUCCESS in 0.06 sec.
[22:30:53] I/launcher - 0 instance(s) of WebDriver still running
[22:30:53] I/launcher - chrome #01 passed
dyn3156-132:client dongxiaohuang$
```

Figure F.2: Client-side End-to-end Tests

F.3 Unit Tests for Back-end



all files server/models/

100% Statements 58/58 **100%** Branches 0/0 **100%** Functions 0/0 **100%** Lines 58/58

File		Statements		Branches		Functions		Lines	
categories.js		100%	4/4	100%	0/0	100%	0/0	100%	4/4
charities.js		100%	7/7	100%	0/0	100%	0/0	100%	7/7
charityRegisters.js		100%	8/8	100%	0/0	100%	0/0	100%	8/8
donations.js		100%	5/5	100%	0/0	100%	0/0	100%	5/5
favorites.js		100%	6/6	100%	0/0	100%	0/0	100%	6/6
payment_details.js		100%	4/4	100%	0/0	100%	0/0	100%	4/4
ratings.js		100%	6/6	100%	0/0	100%	0/0	100%	6/6
timeslots.js		100%	5/5	100%	0/0	100%	0/0	100%	5/5
users.js		100%	6/6	100%	0/0	100%	0/0	100%	6/6
volunteers.js		100%	7/7	100%	0/0	100%	0/0	100%	7/7

Figure F.3: Back-end Unit Tests

F.4 API Tests for Back-end

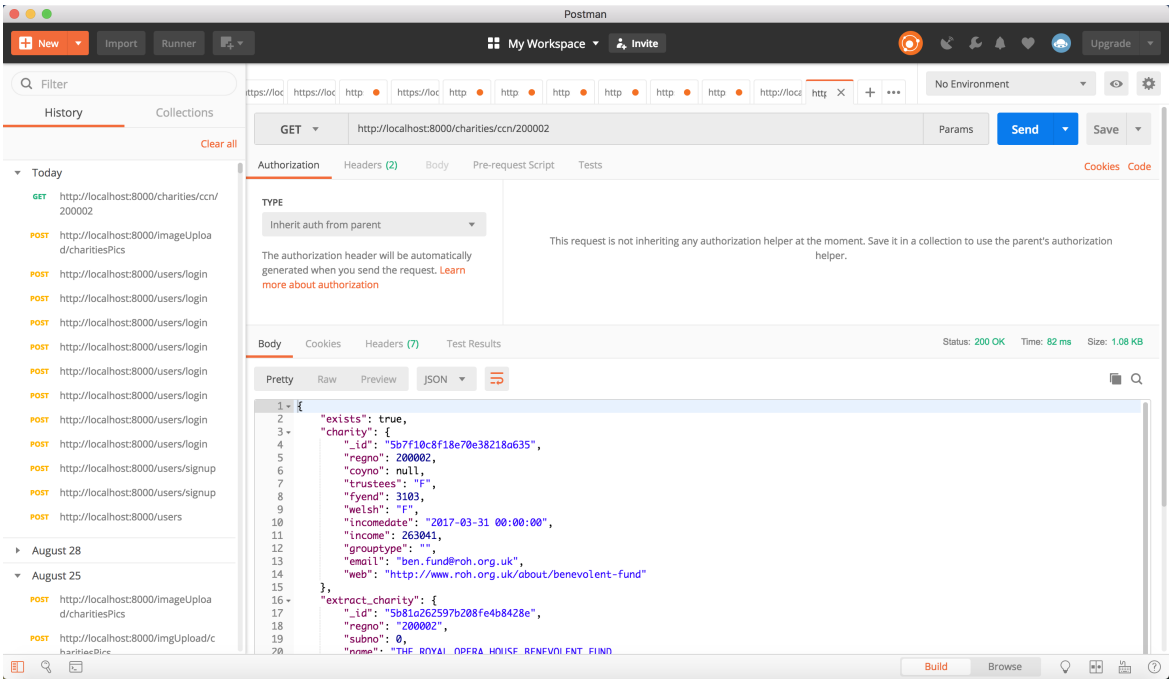


Figure F.4: Back-end API Tests

F.5 Load Tests

F.5.1 User Load 100

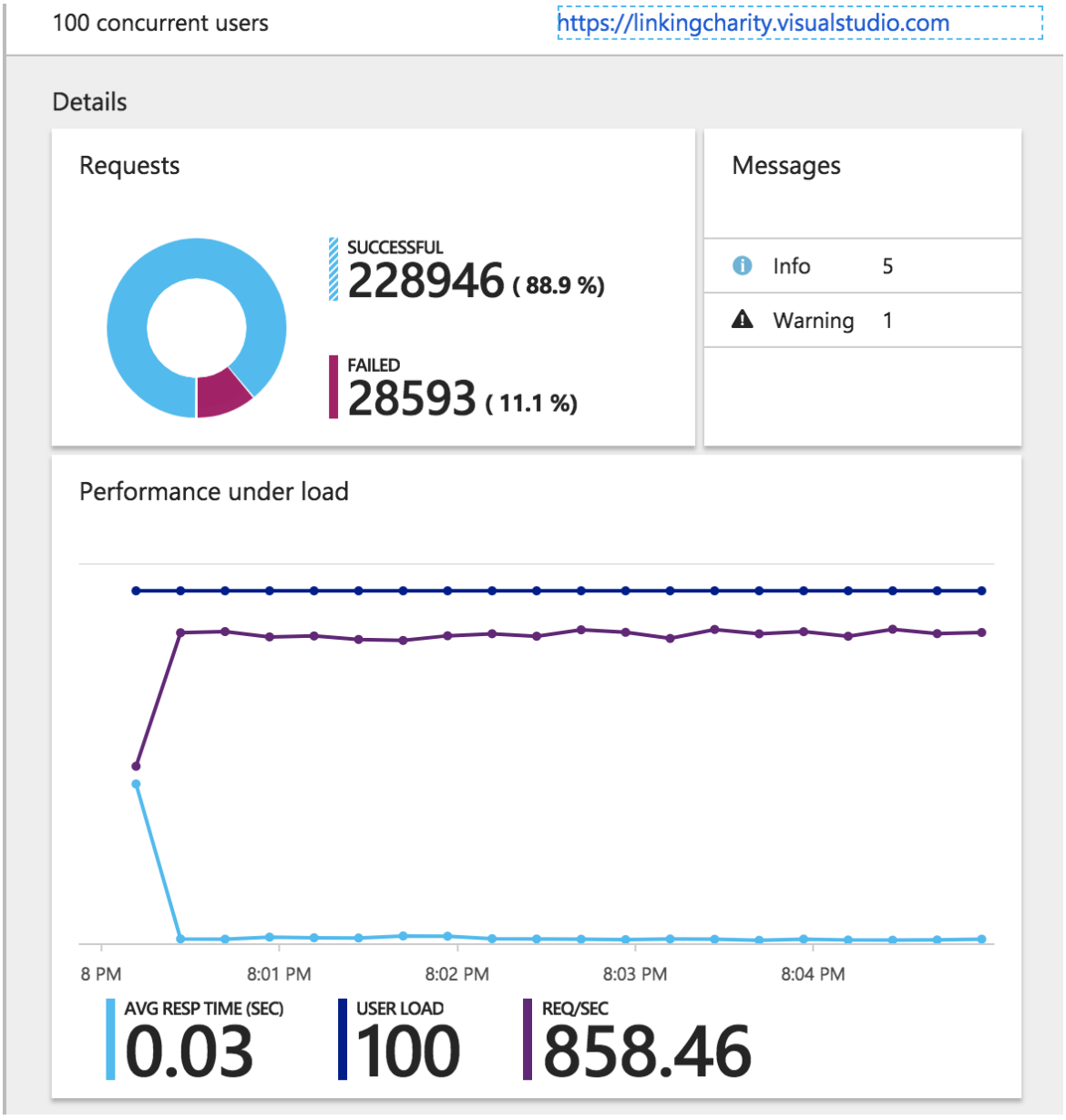


Figure F.5: Performance Test Under 100 User Load

F.5.2 User Load 250

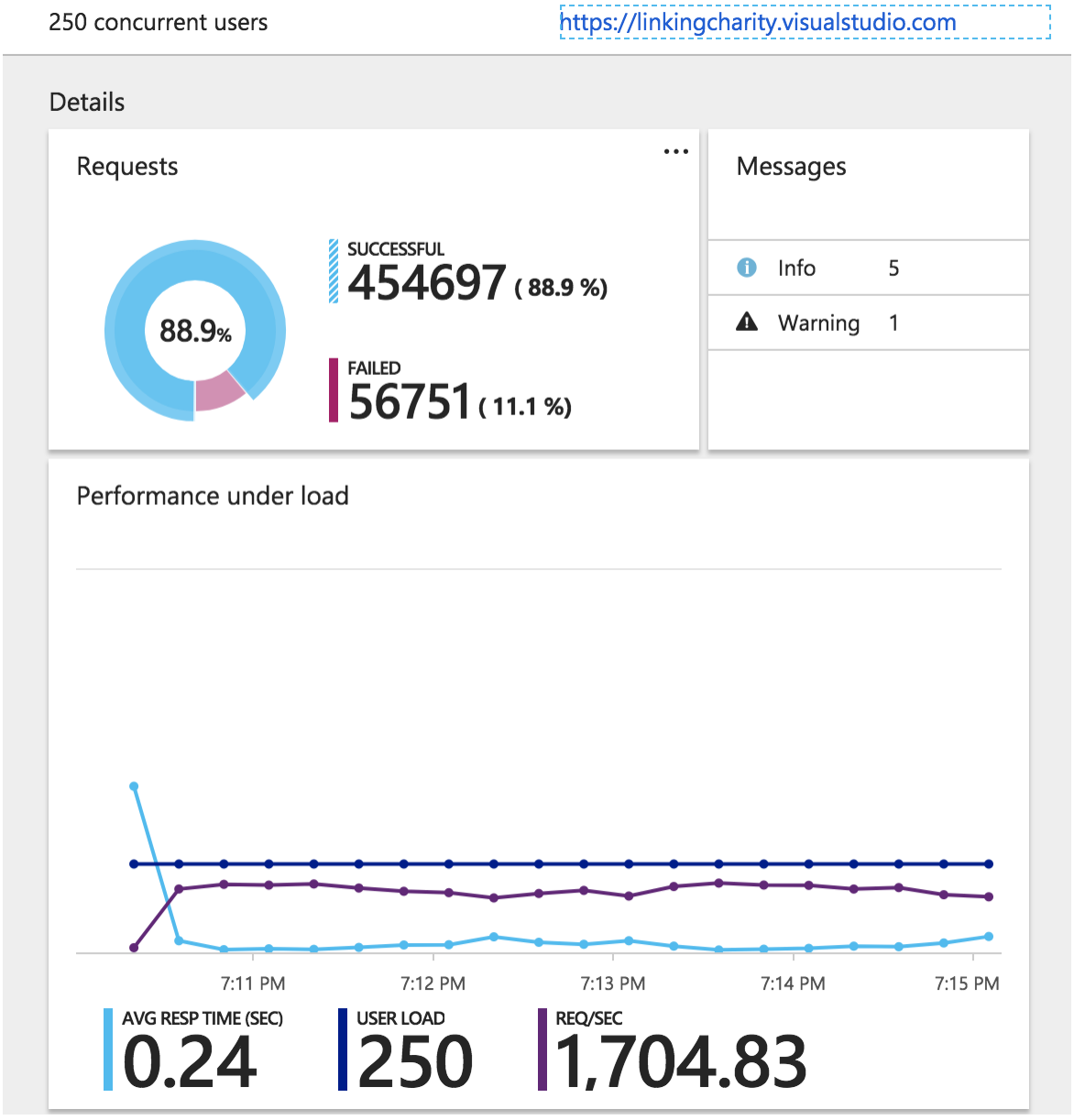


Figure F.6: Performance Test Under 250 User Load

F.5.3 User Load 1000

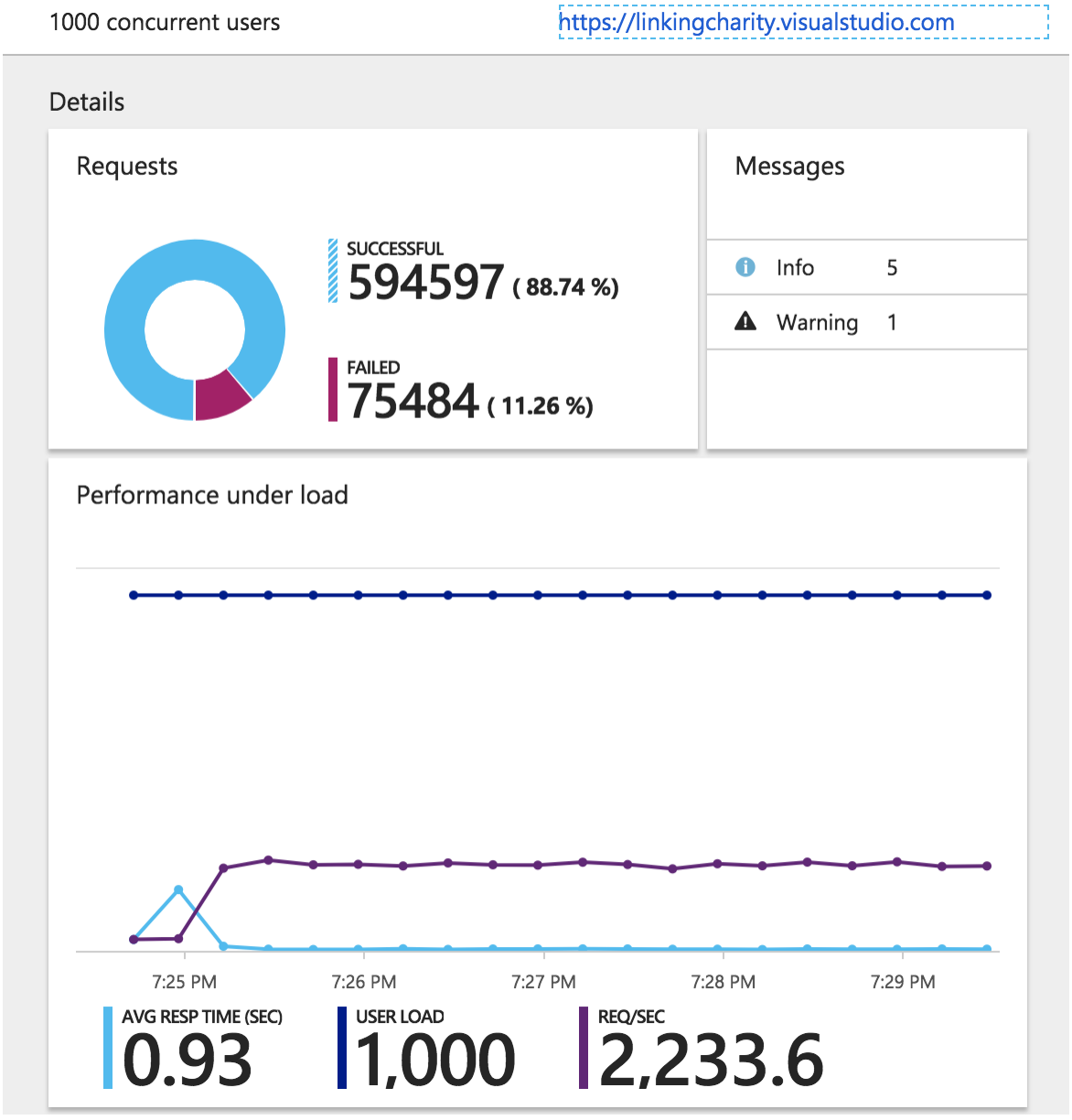


Figure F.7: Performance Test Under 1000 User Load

F.5.4 User Load 5000

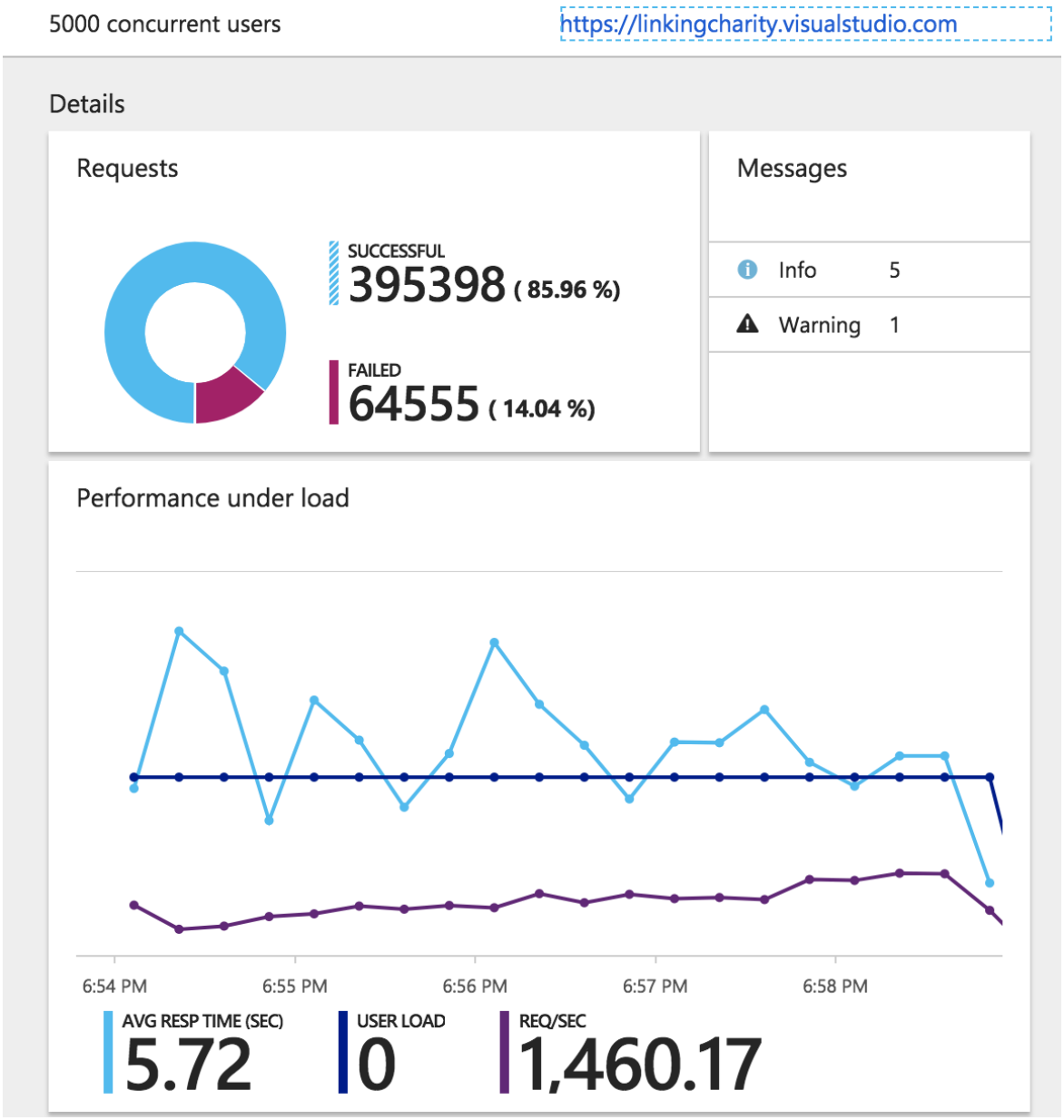


Figure F.8: Performance Test Under 5000 User Load