API Requirement and Design Document

Version 0.1

© Goldilocks Tech Solution Pvt. Ltd.

**CONFIDENTIALITY CLAUSE**

No part of this document may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, recording, photocopying or otherwise without the prior written permission of Goldilocks Tech Solution Pvt. Ltd.

The contents of this document are provided to **Gifting Network Agency** in confidence solely for the purpose of evaluating possible business relationship

| Point of Contact | | | |
| --- | --- | --- | --- |
| Name | Sarju Garg |  |  |
| Title | Sr. Solution Architect |  |  |
| Email | sarju@goldilocks-tech.com | Skype-ID | sarjug |
| Mobile | +91.9810304396 |  |  |

|  |  |
| --- | --- |
| Target Audience | |
| 1 | Gifting Network Team |
|  |  |
|  |  |

**VERSION CONTROL AND DOCUMENT HISTORY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Document Version** | **Date** | **Owner** | **Reviewed By** | **Remarks** |
| 1 | 0.1 | 19th April 2021 | GL Team |  | Initial Draft |
| 2 | 0.2 | 12nd May 2021 | GL Team | Alok, Anurag, Alkesh, Tony | Revised version. Section 5 of system design review added |
|  |  |  |  |  |  |

**Table of Contents**

1 Introduction 8

1.1 Background 8

1.2 Objective 8

1.3 Assumption 9

1.4 Definitions 9

1.5 Document Reference 9

1.6 Document Structure 9

2 System Overview 11

2.1 Integration Flow 11

3 Requirement 12

4 API Design Overview – Pull Method 13

4.1 Login/Logout API 13

4.1.1 Scenarios 14

4.1.2 Parameters 14

4.1.3 Filters 14

4.1.4 Scenarios 14

4.1.5 Functionality 14

4.2 Fund Recommendation 15

4.2.1 Scenarios 15

4.2.2 Parameters List 15

4.2.3 Filters 16

4.2.4 DB Tables 16

4.2.5 Functionality 16

4.3 Contacts 16

4.3.1 Scenarios 16

4.3.2 Parameters 17

4.3.3 Filters 17

4.3.4 DB Tables 17

4.3.5 Functionality 18

4.4 Organization 18

4.4.1 Scenarios 18

4.4.2 Parameters 18

4.4.3 Filter 18

4.4.4 DB Table 18

4.4.5 Functionality 19

4.5 Contribution 19

4.5.1 Scenarios 19

4.5.2 Parameters 19

4.5.3 Filters 20

4.5.4 DB Table 20

4.5.5 Functionality 20

4.6 New Account Setup 21

4.6.1 Scenarios 21

4.6.2 Parameters 21

4.6.3 Filters 21

4.6.4 DB Table 21

4.6.5 Functionality 21

5 Annex A 22

5.1 Contact Fund Table 22

5.2 Fund Recommendation Table 23

5.3 Organization Table 24

5.4 Contact Table 26

5.5 Organization contact Table 28

5.6 Activty Table 28

5.7 Activity Log Table 29

5.8 Auth User Table 29

5.9 Fund Table 30

5.10 Contact Type Contact Table 31

5.11 Contact Type Table 31

5.12 Org Interest Area Table 31

5.13 Interest Area Table 31

5.14 Geographic Area Table 32

5.15 Population Served Table 32

5.16 Donation Table 32

5.17 Config Table 34

5.18 User Fund Setup Table 35

5.19 Investment Pool Table 35

5.20 Email Address Table 36

Target Audience

The intended audience required to have access to this document is as follows:

* Gifting Network Project Team

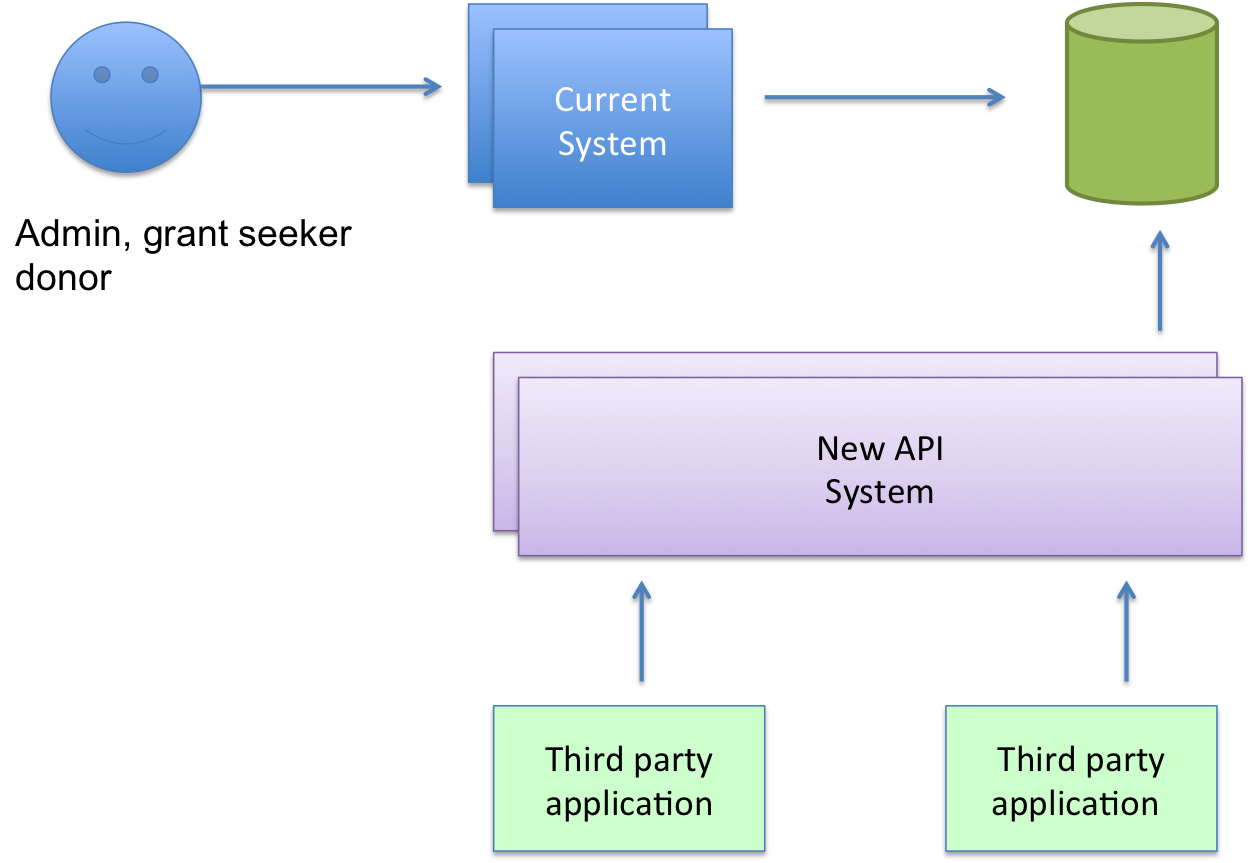
Abbreviations used in the document

The terminologies that will be used in the document are as follows:

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Name** | **Description** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |

# Introduction

## Background



As per above diagram, the current system is reffered to as Gifting network. There are 3 set of users

1. Grant seeker/Organization who seek the grant from donor.
2. Donor who recommend or contribute a grant.
3. Admin, who approves/reject the grant

This diagram depicts the pull method. Third party application will call the API to get response from new API system

## Objective

This document contains requirements and design/architecture inputs for the API subsystem on the Gifting network based on the discussion with client. This document is intended to list down all the features and functionalities which is intended to be developed in the API subsystem

## Assumption

* The third party platform who will consume the newly created API has the information about donors.
* Each XML/JSON parameter shall be mapped to DB parameter. If any XML parameter is not mapping to DB parmeter, the request would be rejected.

## Definitions

The terminologies that will be used in the document are as follows:

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Name** | **Description** |
| 1 | API | Application Programming Interface |
| 2 | HTTP | Hyper Text Transfer Protocol |
| 3 | JSON | Java Script Object Notation |
| 4 | XML | eXtensible Markup Language |
|  |  |  |

A RESTful API is a style of communication between systems where resources are defined by URI and its operations are defined by the use of HTTP methods.

## Document Reference

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Document Title** | **File Name** | **Description** |
| 1 | Innotrust 2 REST API Reference | Innotrust2 Rest Web ServicesAPI Documentation | Reference for API development |
| 2 | Microsoft Web API design | https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design#organize-the-api-around-resources | Reference site for API implementation |
| 3 | RFP 6750 | **The OAuth 2.0 Authorization Framework: Bearer Token Usage**) | Reference for Login/Logut API implementaton |
| 4 | API Design | **https://api.gov.au/standards/national\_api\_standards/naming-conventions.html** | Good reference for URL naming conventions and design principle |

## Document Structure

The document has been structured in the sections as follows:

Section 2 – This section describe our understanding of current system.

Section 3 – This section describe design overview.

Section 4 - The description of API flow in detail

1. Login / Logout
2. Recommendation
3. Contact
4. Organization
5. Contribution
6. New Account Setup

# System Overview

This section describe the current understanding of the system.

1. There are Community foundation and in each such foundation, there are multiple organisation (who are grant seekers) There is a Contact for each Organization
2. There are donor attached with each community foundation and also has its own funds. All Donor is aligned to one or more fund(s)
3. There are multiple community foundation but each community is independent (JCF, HGA, GMF..)
4. If a same donor want to donate to 2 organisation which are part of different community foundation, then donor has to register with each community foundation.
5. Fund is maintained at global level. Funds are visibile to community foundation level through intermediary accouting company like (HMS, Reliance, Innovest, Clear Water). So when the donor account is created, the community foundation platform and accounting companies platform are synced either by API/file sync mode so that each have the same donor and funds information.
6. So for example a donor can have two funds. Fund1 and Fund2, and may be connected to 2 community foundation (CF1 and CF2). Both Funds would be visible to community foundation platform when donor login in. He can recommend funds as per this wish from either fund or both fund.
7. For every organization there is a entry in organization contact table to identify the contact from that organization. For every organization contact there is entry in contact table.
8. Recommendation is done thru the fund. If it is happening from Credit Card, it is called contribution.
9. Recomendation can be immediate/recurring or on future date
10. There is a admin (may be a pool) for each community foundation who approve/reject the recommendation raised by donor.
11. The third party platform who will consume the newly created API has the information about donors.
12. All data to be raw data. No counter data to be shown like the number of donors in a given period. It would be like list of donors in period.

## Integration Flow

TBA

# Requirement

This section list down the requirement in detail

**Deployment Requirement**

A single API subsystem can handle multiple partner platforms. To resolve this, the partner platformwill be identifed by client ID.

**Technical Stack Requirement**

* DB is in Postgres
* Rest API
* Java but no Hibernate (simple JDBC query)
* Spring boot/Spring Framework

**API Design Requirement**

1. Data format - Json and XML to be supported
2. Pull model both
3. Pagination to be implemented
4. URL naming format - URL format can be used in this way (To be identified by the client ID)
5. POST vs GET - POST method to be supported
6. Field returned in API response should be selected by users.
7. The query should be in the form of SQL
8. Login/Logout method to be supported
9. In case filter parameter is not provided in user API, the complete datatable dump would be provided.
10. Orderby clause along with ascending /descending option would be supported.

**User Management Functionality Requirements**

1. Multiple session for same user name /password will not be allowed.
2. There would be a session timeout. The value would be 30 mins (configurable). This is the time for which the session would be maximum allowed in the system.
3. If the password is entered incorrectly N times (configurable), lock the user for N minutes (confiigurable)
4. Token (userSessionID) should be generated at the time of login which should be used for subsequent APIs.
5. For any user API, then usersessionId (UserSessionID) would be passed which will be checked from the system. If the ID exists in the system, then only it is allowed or else the request is rejected.

**Logging Requirement**

1. All the activity should be logged in the system

The following where clause operation to be supported:  
**=, !=, &lt, &lt=, &gt, &gt=, null, notnull, like, notlike, contain, notcontain, in,notin**

# API Design Overview – Pull Method

This section describe the API design overview for the pull scenarios. The high level algorithm for API design is as follows:

1. Receive the request on HTTP interface
2. Parse the request using either XML or json decoder as per the Content Type received in the request.
3. Get the select parameter, filter parameter and order by parameter from the request.
4. Find the corrosponding DB parameter for each such request parameter recevied
5. Fire the SQL query
6. Get the SQL response
7. Convert the SQL response into XML/JSON response using XML/JSON encoder. The selection of encoder would depend on the system configuration.
8. Send the Response.

## Login/Logout API

This section describes the fund login/Logout API.

### Scenarios

### Parameters

For login API, the parameter are username, password and clientID

For logout API, the parameter list us usersessionID

### Filters

There are no filters in this case.

### Scenarios

### Functionality

Login Flow

1. When user hits the Login url, the user send the username, password and client ID as parameter.
2. Check if a session already exist for this user by doing a lookup in local token table. If the username /client ID combination exist, return “Session Already Exist” with 200 Ok Response

<data>

<status type="Error" code="LoginError" message="Session Already Exists."

</data>

1. Basis on client ID, the respective server/partner DB would be chosen
2. These parameter is looked up in the auth\_user table. (The table has parameter with same name)
3. If parameter does not match, then a 200 Ok with Login error is returned.

<data>

<status type="Error" code="LoginError" message="Invalid user name or password."

</data>

1. If parameter matches, then a unique token ID is generated and returned

<data>

<status />

<result \_pageNumber="1">

<recordset> <record UserSessionID="9d1da3df-104b-42f2-830c-673bf6516e43" Status="Success" />

</recordset>

</result>

</data>

Token ID is saved in the system in a local table along with other details. The information stored is

1) Token ID, username, password, client ID, client IP, time of request

Logout

1. To logout, user call the logout API.
2. The parameter is “user session ID generated in the login API
3. If the session Id is valid, then following response is returned

<data>

 <status type="Info" code="Logout" message="Done" />

</data>

1. If the session is invalid or not present, the following response is returned

<data>

 <status type="Error" code="Logout" message="UserSessionID not found" />

</data>

Token ID to be checked in each user API as per below logic.

* For any API, token ID is mandatory. If token ID is missing, then return error
* Also validate the token ID, check currenttime-login\_time< 30 min value. If not so, then return error “Invalid Token ID” and delete the entry from table.
* If token ID is present, check the serverID from the table, get connection string and connect to the respective database.

## Fund Recommendation

This section describes the fund recommendation API.

### Scenarios

1. Find all the recommendations made by the donors. (Period), group by (Donor, organization). Ascending, descending  
This comprises all the details of funds, organizations, donors, etc.  
2. Find recommendations made by a particular donor3. Find the details of the organizations been recommended by a particular donor.  
4. Find the recommendations made over a certain period.  
5. List the organizations been recommended over a period of time.

### Parameters List

Date, DonorName (Fname+Mname+LName), OrganizationName, OrgContactname(Fname+Mname+LName), RecommendationAmount, Fund Name, Status, ApprovalDate, OrganizationState, OrganizationCity, OrganizationZip

### Filters

The list of filters is as follows  
1. Based on the amount  
2. Based on geographic area (city, zip, state)  
3. Based on donor  
4. Based on organization  
5. Based on the status  
6. Based on the date

If multiple filters are given, then the ADD operator would be followed.

### DB Tables

|  |  |  |
| --- | --- | --- |
| Table Name | Description | Key Column |
| Contact\_fund | Fund/Donor mapping table | Contact\_id, fund\_id |
| Fund recommendation | Contain recommendation information | Recommendation\_id, contact\_id, amount, organization\_id |
| Organization | Contain organization information | Organization\_id, name |
| Contact | Contain donor/grant-seeker/staff/admin information | Donor\_id, name |
| Fund | Contain information about fund | Fund\_id, |

### Functionality

The URL would be as follows

https://www.XYZ.com/recommend/FindRecommendation?

## Contacts

This section describes the fund contacts API. Contact here refers to donor who is giving grants.

### Scenarios

### Parameters

BudgetMatchAmount, CompanyName, CreatedOn, Deceased, DateOfBirth, FirstName, Gender, LastName,

LastUpdated, MiddleName, Notes, SSN, Suffix1, Suffix2, Website, RegistrationType, ContactType, Name (First name+middle name+last name)

### Filters

Filter: (all filter are optional)

1. Date Filter –
2. Contact Type (donor/GS/admin)
3. OperationType (Add/update) – Can find out if contact are added or modified.

Since no DB parameter like Operation Type exists ,the filter would be as follows

1. CreatedOn
2. LastUpdated
3. ContactType (donor/GS/admin)

If the user want to know the added contacts in a period of time, the user must use CreatedOn and if the user want to know updated contacts in a period of time, the user must use LastUpdated

### DB Tables

|  |  |  |
| --- | --- | --- |
| Table Name | Description | Key Column |
| Contact | Contain donor/grant-seeker/staff/admin information |  |
| Contact Type | Contain type of contact information | Contact\_type |
| Contact\_type\_contact | Mapping of contact with contact type |  |

### Functionality

## Organization

This section describe the fund organzation API. This organization signify the grant seeker. Within organization, there can be a contact who is the SPOC for that organization

### Scenarios

### Parameters

CreatedOn, EIN, GuidestarLink, IRSName, LastUpdated, Mission, Name, Email, Programs

, Status, WebSite, YearEstablished, AllowRecommendation, OrgType, InterestArea, Description, GeographicAreaID, PopulationServed

### Filter

Filter:

* Date - Date
* OperationType (Add/update) – OperationType

Since no DB parameter like Operation Type exists ,the filter would be as follows

1. CreatedOn
2. LastUpdated

If the user want to know the added contacts in a period of time, the user must use CreatedOn and if the user want to know updated contacts in a period of time, the user must use LastUpdated

### DB Table

|  |  |  |
| --- | --- | --- |
| Table Name | Description | Key Column |
| Organization | Contain organization information | Organization\_id, name |
| Org\_Interest\_area | Mapping of organization with interest Aread |  |
| Interest | Interest of each organization. Ther can be multiple interest area per organization | InterestArea, Description |
| Geographical Area | Geographical area of each organization. There can be multiple geographical area | GeographicAreaID |
| Population Served | List of population served per organization | PopulationServed |

### Functionality

## Contribution

This section describes the fund contribution API. When donor is paying thru CC card or other means like net banking, then it is called contribution. No credit card information is saved in the system.

### Scenarios

### Parameters

TargetType, TargetID, Amount , Interval, StartDate, NoEnd, EndDate, DedicatedToName, NotifyFName

NotifyLName, Notify\_Email, Notify\_Phone, Notify\_Address\_One ,Notify\_Address\_Two, Notify\_City, Notify\_State, Notify\_Country, Notify\_Zip, DonorOrgName, Guest\_Fname, Guest\_Lname, Guest\_Email, Guest\_Phone, Guest\_Address\_One, Guest\_Address\_Two, Guest\_City, Guest\_State, Guest\_Country, Guest\_Zip, Status , Message, AccountNumber, AccountType, Currency, TransactionType, Response

,CreatedOn, UpdatedOn

### Filters

### DB Table

|  |  |  |
| --- | --- | --- |
| Table Name | Description | Key Column |
| Donation | Contain contribution information |  |
| Org\_Interest\_area | Mapping of organization with interest Aread |  |
| Interest | Interest of each organization. Ther can be multiple interest area per organization | InterestArea, Description |
| Geographical Area | Geographical area of each organization. There can be multiple geographical area | GeographicAreaID |
| Population Served | List of population served per organization | PopulationServed |

### Functionality

getContributionListInfo

notify\_to character(1), - This is a flag – Y/N value if the value is Y, then only the notify value below will come….

donor\_contact\_id integer, - will decide if guest value is to be shown or not..

User can contribute as guest or without login

If donor\_contact\_id is there, then it is not a guest user.

Donor\_contact\_id is to be seen how it is mapped before

How to identify successful/unsuccessful transaction

Recommendation to adhoc organization

Organization ID in donation table

Target\_type – fund transfer / org transfer – organization or fund.

Target\_id – organization ID if transfer to organization, fund ID if transfer to fund

Fund to fund transfer

If target ID is null,then transfer to adhoc organization.

Donor contact ID is null in case of guest. If logged user, then has donor ID.

## New Account Setup

This section describes the new account setup API. Only approved account would be returned back. The new user creation is a multi step process (8 steps) and once all the step are completed, then the new account is created. The new account needs to be approved by admin.

### Scenarios

### Parameters

CreatedOn, EmailAddress, InvestmentPools, FundName

### Filters

CreatedOn

### DB Table

|  |  |  |
| --- | --- | --- |
| Table Name | Description | Key Column |
| User\_fund\_setup | Contain fund setup information for user |  |
| Contact | Contact info | CreatedOn, DonorApprovalStatus |
| Contact\_address |  |  |
| Email\_address | Contain email ID of contact | EmailAddress |
| fund |  | FundName |
| Contact\_fund |  |  |
| Investment\_pools | List of investment pool | InvestmentPools |

### Functionality

getNewApprovedAccountSetupInfo

The flow would as follows

* For Date, check for created\_on field in contact table
* Get the contact\_ID from contact table where donor\_approval\_status = approved
* For all such contact ID, check if they are present in user fund setup table
* Once this look up is done, then a list of contact ID is received which are approved and filter criteria is successful.
* For each contact ID, get email Address from email address table, There can be multiple values of email.
* For each contact ID, go to investment\_pool table and get the fund\_id and investment pool.
* For the fundID, get the name from the fund\_table (FundName)

# System Design Overview

This section describe the high level design of the system. There are 2 set of API:

1. User/Session Management API
   1. Login API
   2. Logout API
2. Service API
   1. Organization
   2. Contact
   3. New Account Setup
   4. Recommendation
   5. Contribution

The design would be explained in context of these 2 API set

## User / Session Management API Design Overview

The users are defined in the respsective database of master system. For example, if there are 4 master system behind the API system, then each master system will have their own postgres database which will contain the user table (auth\_user) containing the information about the user.

The DB details of each master system would be defined in the API subsystem and would be mapped with client ID.

When the login request is recevied on the API subsystem, then it also contains the client ID along with username and password which identify the master subsystem. Basis of client ID, the master DB is selected and the username and password are validated from the users table.

If the user credential are validated, then the a user session ID is generated and returned as response to login request in the body of the HTTP response. This user session ID is shortlived and is valid for a the session time period. The usersession ID is maintained in the table present in the API system along with other details .

The logic for generating usersessionID is MD5 digest algorithm where the input passed to the algorithm in username, password, date/time and client ID

For logout API, the API user should pass the user session ID as the parameter. If the usersessionID is valid and present in the system , then the user is logged out of the system.

Service API Design Overview

Each of the service API has a similar XML syntax. Each of the service API follow the same design flow so that the service flow can be generalized and implemented for all service implementation.

Each API will have the usersessionID as part of the HTTP header. The name of the HTTP header is x-usersessionid

There are 2 key tables in the API subsystem for same purpose.

1. API to DB mapping table.
   1. Each API has a corrosponding master table which will have the most of the parameter which need to be shown to the user.
   2. Also each API which have parameter which may not be present in the master table and hence would require join on some common ID parameter between table to get the list of all such parameter. This join string is also part of configuration in this table
   3. This table also contain the list of parameter to be included in the select list. If any parameter is present in request select list but not present in the table, then the same request is rejected. Also if the select list is empty, then this complete list is returned in the response.
   4. This table also contain the list of parameter to be included in the where list. If any parameter is present in request where list but not present in the table, then the same request is rejected. Also if the where list is empty, then this complete data obtained as part of executing the query is returned in the response.
   5. As per above point, returning the complete data from the table can be time consuming. This table also contain a parameter which signify whether the where list can be empty in the API request or not. The value of this parameter is Yes/No.

If the value is No, and where list is empty then the request is rejected

So this table has the following column

* + 1. API name
    2. Master table name
    3. Join string
    4. Select List
    5. Where list
    6. Where flag

1. API parameter to DB column mapping table
   1. Each API parameter has a corrosponding mapping to the DB column.
   2. This parameter can be a simple parameter or a composite parameter which combines multiple DB column into one parameter. Example is name which is composite parameter containing first name, middle name and last name.

So this table has the following column

* + 1. API name
    2. API parameter name
    3. Table name
    4. DB column name

For all the services, the data is configured in these 2 tables.

So, a generic flow would be as follows

1. Based on the API name, the following is done
   1. the API to DB table is refered and the join string is retrived along with the master table.
   2. Based on each parameter, the DB column is retrieved from the “API parameter to DB column mapping” table. This will help is building the select list and the where list
   3. Apart from the time page information and order by information is also added in the DB query
   4. By merging these the output of above steps, the DB query string would be generated.
   5. Fire the DB query and receive the response
   6. Generate the XML response based on the reverse mapping of the DB column to API parameter.

## Pagination

For each service request, the API user can refer to the page number and the page size in the API request. So if API user set page number=1 and page size= 20, then the first 20 records would be sent to the user. However if API user set page number=2 and page size= 20, then the 21st to 40th records would be sent to the user.

In the API response, the page number and page size is returned. Also the total row in the table based on the query is returned as well

If there are less row (page size) than what user is requested for, then less records would be returned to user.

## Rate Limiting

Rate limiting refers to preventing the frequency of an operation from exceeding certain limit. It is commonly used to protect underlying services and resources. We can apply rate limiting per user to provide fair and reasonable use, without affecting other users. These limits might be applied over longer time periods, or they might be applied to resources that are not measured by rate but by quantity allocated.

To enforce rate limiting, first understand why it is being applied in this case, and then determine which attributes of the request are best suited to be used as the limiting key (for example, source IP address, user, API key). After you choose a limiting key, a limiting implementation can use it to track usage. When limits are reached, the service returns a limiting signal (usually a 429 HTTP response).

There can be 3 limiting keys

1. User
2. API
3. Session

Within an active sesssion, the max TPS allowed per API or across all API.

With an active user, the TPS would be controlled across session per API.

In our case, limit on each user and at API level shall be provided

## Configuration

This section describe the configuration aspect of the system. The following are the list of parameters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SN | Parameter Number | Description | Value | Remarks |
| 1 | Session Timeout | Overall session time in minutes for which usersessionID would be valid | 30 |  |
| 2 | Page Size | Maximum page size | 100 |  |
| 3 | DB Connect Timeout | Maximum time in seconds to wait for DB connection | 5 |  |
| 4 | DB Read timeout | Maximum time in seconds to response for DB request | 60 |  |

# Annex A

List of tables are shown here with the key parameter highlighted.

## Contact Fund Table

CREATE TABLE public.contact\_fund (

contact\_id integer NOT NULL,

contact\_type\_id integer NOT NULL,

establisher character(1) DEFAULT 'N'::bpchar NOT NULL,

fund\_id character varying(256) NOT NULL,

mail\_statement character(1) DEFAULT 'Y'::bpchar NOT NULL,

make\_grant\_recommendation character(1) DEFAULT 'Y'::bpchar NOT NULL,

viewable character(1) DEFAULT 'Y'::bpchar NOT NULL,

CONSTRAINT contact\_fund\_establisher\_check CHECK (((establisher = 'Y'::bpchar) OR (establisher = 'N'::bpchar))),

CONSTRAINT contact\_fund\_mail\_statement\_check CHECK (((mail\_statement = 'Y'::bpchar) OR (mail\_statement = 'N'::bpchar))),

CONSTRAINT contact\_fund\_make\_grant\_recommendation\_check CHECK (((make\_grant\_recommendation = 'Y'::bpchar) OR (make\_grant\_recommendation = 'N'::bpchar))),

CONSTRAINT contact\_fund\_viewable\_check CHECK (((viewable = 'Y'::bpchar) OR (viewable = 'N'::bpchar)))

);

## Fund Recommendation Table

CREATE TABLE public.fund\_recommendation (

fund\_recommendation\_id integer DEFAULT nextval(('fund\_recommendationid'::text)::regclass) NOT NULL,

amount double precision,

anonymous character(1) DEFAULT 'Y'::bpchar NOT NULL,

contact\_id integer,

contact\_name character varying(128),

contact\_phone character varying(32),

contact\_title character varying(128),

date\_submitted timestamp without time zone DEFAULT now() NOT NULL,

fund\_id character varying(256) NOT NULL,

grant\_num text,

grant\_purpose text,

notes text,

organization\_id integer,

org\_address1 character varying(128),

org\_address2 character varying(128),

org\_city character varying(128),

org\_name character varying(128),

org\_need\_app\_id integer,

org\_state character varying(128),

org\_zip character varying(10),

payment\_schedule text,

is\_approved character(1) DEFAULT 'N'::bpchar,

status text,

last\_updated timestamp without time zone DEFAULT now() NOT NULL,

\_remote\_id text,

primary\_contact\_id integer,

approved\_date timestamp without time zone,

batch\_no text,

default\_grant\_date date,

default\_board\_date date,

default\_payment\_date date,

default\_action text,

default\_payment\_type text,

default\_grant\_status text,

org\_ein character varying(10),

org\_email character varying(256),

org\_country character varying(32),

CONSTRAINT fund\_recommendation\_anonymous\_check CHECK (((anonymous = 'Y'::bpchar) OR (anonymous = 'N'::bpchar)))

);

## Organization Table

CREATE TABLE public.organization (

organization\_id integer DEFAULT nextval(('organizationid'::text)::regclass) NOT NULL,

\_remote\_id text,

alt\_name text,

annual\_operating\_revenue double precision,

board text,

board\_last\_update timestamp without time zone DEFAULT now(),

certification character(1) DEFAULT 'N'::bpchar,

created\_on timestamp without time zone DEFAULT now(), - CreatedOn

date\_reviewed timestamp without time zone,

db\_stat integer DEFAULT 0,

ein character varying(32), - EIN

endowement character varying(56),

fims\_id character varying(64),

financials\_audited boolean DEFAULT false NOT NULL,

fiscal\_agent\_id integer,

funding\_opp text,

grantee\_type\_1 text,

grantee\_type\_2 text,

grantee\_type\_3 text,

guidestar\_link character varying(256),GuidestarLink

history text,

irs\_name text, IRSName

is\_profile\_complete character(1) DEFAULT 'N'::bpchar,

last\_updated timestamp without time zone DEFAULT now() NOT NULL, - LastUpdated

last\_updated\_by integer,

logo\_url character varying(256),

max\_interests integer DEFAULT 10 NOT NULL,

min\_donation double precision DEFAULT 100,

mission text, -Mission

name text NOT NULL, - Name

parent\_organization\_id integer,

past\_funding boolean,

pending\_fund boolean,

photo\_url character varying(256),

preferred\_email character varying(256), Email

priority integer,

programs text, Programs

rec\_code character varying(56),

reviewed character(1) DEFAULT 'N'::bpchar,

staff\_title character varying(256),

status text,- Status

sync character(1),

visible character(1) DEFAULT 'N'::bpchar NOT NULL,

volunteerism text,

web\_site character varying(256), WebSite

work\_log text,

year\_established date, YearEstablished

allow\_recommendation character(1) DEFAULT 'N'::bpchar, AllowRecommendation

profile\_notification character(1) DEFAULT 'Y'::bpchar,

org\_type text, OrgType

sync\_approval character varying(20),

CONSTRAINT organization\_certification\_check CHECK (((certification = 'Y'::bpchar) OR (certification = 'N'::bpchar))),

CONSTRAINT organization\_reviewed\_check CHECK (((reviewed = 'Y'::bpchar) OR (reviewed = 'N'::bpchar))),

CONSTRAINT organization\_visible\_check CHECK (((visible = 'Y'::bpchar) OR (visible = 'N'::bpchar)))

);

## Contact Table

CREATE TABLE public.contact (

contact\_id integer DEFAULT nextval(('contactid'::text)::regclass) NOT NULL,

\_remote\_id text,

assistant\_id integer,

auth\_user\_id integer,

being\_reviewed character(1) DEFAULT 'N'::bpchar,

budget\_match\_amount character varying(30), - - BudgetMatchAmount

budget\_match\_operator character(1), -

company\_name text, - COMPANY NAME – CompanyName

created\_on timestamp without time zone DEFAULT now() NOT NULL, - -CreatedOn

deceased character(1) DEFAULT 'N'::bpchar, Deceased

dob date, - - DateOfBirth

dod date,

end\_match date,

ethnicity text,

fims\_id character varying(256),

first\_name text NOT NULL, - FirstName

gender character(1), - Gender

informal text,

last\_name text NOT NULL, - LastName

last\_updated timestamp without time zone DEFAULT now() NOT NULL, LastUpdated

last\_updated\_by integer,

middle\_name text, - MiddleName

nosync character(1) DEFAULT 'N'::bpchar,

notes text, - Notes

other\_person\_type text,

pan character(1) DEFAULT 'N'::bpchar NOT NULL,

photo\_url text,

prefix character varying(20),

receive\_email character(1) DEFAULT 'Y'::bpchar,

religion text,

ssn character varying(11), -SSN

start\_match date,

suffix1 character varying(20), Suffix1

suffix2 character varying(20), Suffix2

title text,

volunteerism character(1) DEFAULT 'N'::bpchar,

web\_site text, Website

sync\_approval character varying(20),

registration\_type text, RegistrationType

donor\_approval\_status character varying(100) DEFAULT 'Pending'::character varying, DonorApprovalStatus

CONSTRAINT contact\_being\_reviewed\_check CHECK (((being\_reviewed = 'Y'::bpchar) OR (being\_reviewed = 'N'::bpchar))),

CONSTRAINT contact\_budget\_match\_operator\_check CHECK (((budget\_match\_operator = '<'::bpchar) OR (budget\_match\_operator = '>'::bpchar) OR (budget\_match\_operator = '='::bpchar))),

CONSTRAINT contact\_deceased\_check CHECK (((deceased = 'Y'::bpchar) OR (deceased = 'N'::bpchar))),

CONSTRAINT contact\_gender\_check CHECK (((gender = 'M'::bpchar) OR (gender = 'F'::bpchar))),

CONSTRAINT contact\_pan\_check CHECK (((pan = 'Y'::bpchar) OR (pan = 'N'::bpchar))),

CONSTRAINT contact\_receive\_email\_check CHECK (((receive\_email = 'Y'::bpchar) OR (receive\_email = 'N'::bpchar))),

CONSTRAINT contact\_volunteerism\_check CHECK (((volunteerism = 'Y'::bpchar) OR (volunteerism = 'N'::bpchar)))

);

## Organization contact Table

CREATE TABLE public.organization\_contact (

access\_level integer DEFAULT 1 NOT NULL,

contact\_id integer NOT NULL,

contact\_role character varying(64),

department\_id integer,

is\_default character(1) DEFAULT 'N'::bpchar NOT NULL,

is\_former\_resp character(1) DEFAULT 'N'::bpchar,

last\_updated timestamp without time zone DEFAULT now() NOT NULL,

organization\_id integer NOT NULL,

status text DEFAULT 'pending'::text NOT NULL,

\_assoc\_remote\_id text,

budget\_upload character(1) DEFAULT 'N'::bpchar NOT NULL,

CONSTRAINT organization\_contact\_access\_level\_check CHECK (((access\_level = 1) OR (access\_level = 2))),

CONSTRAINT organization\_contact\_is\_default\_check CHECK (((is\_default = 'Y'::bpchar) OR (is\_default = 'N'::bpchar))),

CONSTRAINT organization\_contact\_status\_check CHECK (((status = 'pending'::text) OR (status = 'approved'::text) OR (status = 'denied'::text)))

);

## Activty Table

CREATE TABLE public.activity (

activity\_id integer DEFAULT nextval(('activityid'::text)::regclass) NOT NULL,

auth\_user\_id integer,

contact\_id integer,

description text NOT NULL,

entered\_on timestamp without time zone DEFAULT now() NOT NULL,

ip\_address character varying(18)

);

## Activity Log Table

CREATE TABLE public.activity\_log (

id integer NOT NULL,

name character varying(64),

action character varying(64) NOT NULL,

url character varying(255),

description character varying(255),

target\_string\_id character varying(64),

target\_id integer,

target\_type character varying(64),

data text,

ip character varying(64),

agent character varying(255),

auth\_user\_id integer,

created\_on timestamp without time zone DEFAULT now() NOT NULL,

updated\_on timestamp without time zone DEFAULT now() NOT NULL

);

## Auth User Table

CREATE TABLE public.auth\_user (

auth\_user\_id integer DEFAULT nextval(('auth\_userid'::text)::regclass) NOT NULL,

active character(1) DEFAULT 'Y'::bpchar NOT NULL,

being\_reviewed character(1) DEFAULT 'N'::bpchar NOT NULL,

created\_on timestamp without time zone,

modified\_on timestamp without time zone,

password character varying(128) NOT NULL,

transaction\_password character varying(128),

username character varying(128) NOT NULL,

has\_changed\_password character(1) DEFAULT 'N'::bpchar NOT NULL,

CONSTRAINT auth\_user\_active\_check CHECK (((active = 'Y'::bpchar) OR (active = 'N'::bpchar))),

CONSTRAINT auth\_user\_being\_reviewed\_check CHECK (((being\_reviewed = 'Y'::bpchar) OR (being\_reviewed = 'N'::bpchar)))

);

## Fund Table

CREATE TABLE public.fund (

fund\_id character varying(256) NOT NULL,

agree\_uri character varying(256),

anonymous character(1) DEFAULT 'N'::bpchar NOT NULL,

balance double precision,

begin\_balance double precision,

closed character(1) DEFAULT 'N'::bpchar NOT NULL,

date\_established date,

description text,

end\_balance double precision,

fund\_type text NOT NULL,

grant\_rec\_type character varying(32) DEFAULT 'individual'::character varying NOT NULL,

investment\_month text,

investment\_performance double precision,

is\_locked character(1) DEFAULT 'N'::bpchar,

last\_updated timestamp without time zone DEFAULT now(),

min\_donation double precision,

name text NOT NULL,

pool\_id text,

pool\_name text,

trustee text,

trustee\_id text,

statement\_style character varying(50),

account\_id character varying(50),

statement\_balance double precision,

fund\_privacy character varying(20) DEFAULT 'private'::character varying,

customer\_quickref character varying(20),

CONSTRAINT fund\_anonymous\_check CHECK (((anonymous = 'Y'::bpchar) OR (anonymous = 'N'::bpchar))),

CONSTRAINT fund\_closed\_check CHECK (((closed = 'Y'::bpchar) OR (closed = 'N'::bpchar))),

CONSTRAINT fund\_is\_locked\_check CHECK (((is\_locked = 'Y'::bpchar) OR (is\_locked = 'N'::bpchar)))

);

## Contact Type Contact Table

CREATE TABLE public.contact\_type\_contact (

contact\_id integer NOT NULL,

contact\_type\_id integer NOT NULL

);

## Contact Type Table

CREATE TABLE public.contact\_type (

contact\_type\_id integer DEFAULT nextval(('contact\_typeid'::text)::regclass) NOT NULL,

contact\_type text

);

## Org Interest Area Table

Nothing to be picked from this table.

CREATE TABLE public.org\_interest\_area (

interest\_area\_id text NOT NULL, -

organization\_id integer NOT NULL

);

## Interest Area Table

There can be multiple interest area per organization…List output of interest\_area and description – InterestArea, Description

CREATE TABLE public.interest\_area (

interest\_area\_id text NOT NULL, -

description text, - DESCRIPTION

interest\_area text NOT NULL, - InterestArea

parent\_interest\_area\_id text

);

## Geographic Area Table

CREATE TABLE public.org\_geographic\_area (

geographic\_area\_id text NOT NULL,

organization\_id integer NOT NULL

);

Organization can be in multiple geograhy area - List of geographic area ID - GeographicAreaID

## Population Served Table

CREATE TABLE public.org\_population\_served (

organization\_id integer NOT NULL,

population\_served\_id text NOT NULL

);

Organization can be in multiple population - List of population - PopulationServed

## Donation Table

CREATE TABLE public.donation (

id integer NOT NULL,

ref\_id character varying(64),

target\_type character varying(20), - TargetType

target\_id character varying(20), - TargetID

amount double precision, - Amount -

"interval" character varying(20), - Interval

start\_date timestamp without time zone, - StartDate

end\_type character varying(20),

no\_end character(1), - ongoing contribution - NoEnd

end\_date timestamp without time zone, - EndDate

occurrences integer,

dedicated\_to\_name character varying(256), - DedicatedToName

notify\_to character(1), - This is a flag – Y/N value if the value is Y, then only the notify value below will come….

notify\_fname character varying(64), Notify\_FName

notify\_lname character varying(64), - Notify\_LName

notify\_email character varying(256), Notify\_EMail

notify\_phone character varying(20), - Notify\_Phone

notify\_address\_one character varying(256), Notify\_Address\_One

notify\_address\_two character varying(256), Notify\_Address\_Two

notify\_city character varying(64), Notify\_City

notify\_state character varying(32), Notify\_State

notify\_country character varying(32), Notify\_Country

notify\_zip character varying(10), Notify\_Zip

donor\_contact\_id integer, - will decide if guest value is to be shown or not..

donor\_org\_name character varying(256), - DonorOrgName

guest\_fname character varying(64), - Guest\_FName

guest\_lname character varying(64), Guest\_LName

guest\_email character varying(256), Guest\_EMail

guest\_phone character varying(20), Guest\_Phone

guest\_address\_one character varying(256), Guest\_Address\_One

guest\_address\_two character varying(256), Guest\_Address\_Two

guest\_city character varying(64), Guest\_City

guest\_state character varying(32), Guest\_State

guest\_country character varying(32), Guest\_Country

guest\_zip character varying(10), Guest\_Zip

status integer, Status

message character varying(256), Message

account\_number character varying(32), AccountNumber

account\_type character varying(32), AccountType

currency character varying(3) DEFAULT 'USD'::character varying, Currency

transaction\_id character varying(32),

transaction\_date timestamp without time zone DEFAULT now(),

transaction\_type character varying(20),TransactionType

response text, Response

response\_auth\_code character varying(10), ResponseAuthCode

note text,

created\_on timestamp without time zone DEFAULT now() NOT NULL,CreatedOn

updated\_on timestamp without time zone DEFAULT now() NOT NULL UpdatedOn

);

## Config Table

For adding configuration variables.

CREATE TABLE public.config (

config\_id integer DEFAULT nextval('public.configid'::regclass),

admin\_edit boolean DEFAULT false,

config\_area text NOT NULL,

config\_description text,

config\_param text NOT NULL,

help\_text text,

label text,

module\_id integer,

order\_seq integer,

param\_value text,

uifftype text

);

## User Fund Setup Table

CREATE TABLE public.user\_fund\_setup (

contact\_id integer,

form\_1 text,

form\_2 text,

form\_3 text,

form\_4 text,

form\_5 text,

form\_6 text,

form\_7 text,

form\_8 text,

email\_verified character(1)

);

## Investment Pool Table

CREATE TABLE public.investment\_pool (

id integer NOT NULL,

contact\_id integer NOT NULL,

fund\_id character varying(10),

investment\_pools text, InvestmentPools

created\_on timestamp without time zone DEFAULT now() NOT NULL,

updated\_on timestamp without time zone DEFAULT now() NOT NULL

);

## Email Address Table

CREATE TABLE public.email\_address (

email\_address\_id integer DEFAULT nextval('public.email\_addressid'::regclass) NOT NULL,

contact\_id integer NOT NULL,

email\_address text NOT NULL, EmailAddress

email\_address\_name character varying(256) NOT NULL,

is\_primary character(1) DEFAULT 'N'::bpchar NOT NULL,

organization\_id integer,

CONSTRAINT email\_address\_is\_primary\_check CHECK (((is\_primary = 'Y'::bpchar) OR (is\_primary = 'N'::bpchar)))

);

# Annex B

This section explain the design consideration of HTTP protocol in detail.

There are some deivation taken from the standard implementation of REST API and the handling of different header. This section covers the following points.

1. Use of HTTP Method: Get vs Post
2. Supported HTTP Headers and handling
3. HTTP Response Code for success/failure cases and handling
4. Naming definition
5. HTTP Based Authentication and Authorization

## Get vs Post

Going by standard REST implementation, the definition is as follows as per [2]

* **GET** retrieves a representation of the resource at the specified URI. The body of the response message contains the details of the requested resource.
* **POST** creates a new resource at the specified URI. The body of the request message provides the details of the new resource. Note that POST can also be used to trigger operations that don't actually create resources.

In addition,WIKI's extended explanation on [POST request](https://en.wikipedia.org/wiki/POST_(HTTP))(https://en.wikipedia.org/wiki/POST\_(HTTP))

There are times when HTTP GET is less suitable even for data retrieval. An example of this is when a great deal of data would need to be specified in the URL. Browsers and web servers can have limits on the length of the URL that they will handle without truncation or error. Percent-encoding of reserved characters in URLs and query strings can significantly increase their length, and while Apache HTTP Server can handle up to 4,000 characters in a URL,[5] Microsoft Internet Explorer is limited to 2,048 characters in any URL.[6] Equally, HTTP GET should not be used where sensitive information, such as user names and passwords, have to be submitted along with other data for the request to complete. Even if HTTPS is used, preventing the data from being intercepted in transit, the browser history and the web server's logs will likely contain the full URL in plaintext, which may be exposed if either system is hacked. In these cases, HTTP POST should be used.[7]

Normally GET is used where there query string is string as part of URL and not part of HTTP request body. However in case of POST, the input is send as part of HTTP request body.

Going by above definiton, API are requesting for information from server and hence GET should be used in the APIs.

However, we sugges to use POST body for anything non-trivial and line-of-business apps for these reasons:

1. Security -If we use GET with query strings and https, the query strings can be saved in server logs and forwarded as referral links. Both of these are now visible by server/network admins and the next domain the user went to after leaving your app. So if we send a query containing confidential PII data such as a customer's name this may not be desired.
2. URL maximum length - Not a big issue, but some browsers have a limit on the length. So if we have several items in our URL like query, paging, fields to return, etc....
3. POST is not cached by default. Some say caching is desired; however, how often is that exact same set of search criteria for that exact object for that exact customer going to occur before the cache times out anyway?

Although it's noble wanting to stick to pure REST, we must not forget: the standard is about 20 years old, but technology requirements have become much more complex ever since. So we think POST is acceptable for triggering complex queries. Other system like GraphGL uses the same principle allowing POSTs to retrieve data.

This stack flow discussion summarize the issue in a good way. Most of the above is a excerpt from there

<https://stackoverflow.com/questions/19637459/rest-api-using-post-instead-of-get>

Also, we have created an REST API to authenticate users, taking username/password and returning an access token. The API is encrypted under TLS, but exposed to public internet. After evaluating different options, we chose HTTP POST for the REST method of "GET access token," because that's the only way to meet security standards.

So, basis on above discussion, we decided to use POST request for getting data from the server.

## Supported HTTP Headers and handling

The following HTTP header are supported for request and response.

|  |  |  |  |
| --- | --- | --- | --- |
| SN | Header name | Type | Handling |
| 1 | Accept | HTTP Request | The Accept request-header field can be used to specify certain media types which are acceptable for the response.  If the Accept header include the media type not supported by server, the web server could generate an HTTP 406 (Not Acceptable) response message or return a message with a default media type.  In our case, the response is reverted in the format as suggested in the Content Type.  For login/logout request, the Content Type would be application/x-www-form-urlencoded, so the response is returned in default media type (XML) |
| 2 | Content Type | HTTP Request/Response | The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient  If the Content-Type is not supported on server, the server will return HTTP status code 415 (Unsupported Media Type). |
| 3 | Authentication | HTTP request | This request-header field can be used to specify token ID .  If this field is not present in the resource API, then the server will return HTTP status code 401 (UnAuthorized)  Value is: Bearer <access\_token> |
| 4 | WWW-Authenticate | HTTP Response | This response-header field defines the authentication method that should be used to gain access to a resource.  If the token is not valid or expired, then the header will include  HTTP/1.1 401 Unauthorized WWW-Authenticate: Bearer realm="Gifting Network", error="invalid\_token", error\_description="The access token expired" |

## HTTP Response Code for success/failure cases and handling

The following response code would be returned in the below scenarios

|  |  |  |  |
| --- | --- | --- | --- |
| SN | Error Code | Error Description | Remarks |
| 1 | 200 | Success Ok | If the request is processed successfully, 200Ok would be returned |
| 2 | 400 | Bad Request | If the XML or json format is not as per specified format OR any header is missing |
| 3 | 401 | UnAuthorized | If the username or password is incorrect or Token is not valid |
| 4 | 404 | Not found | In case the URI path is not correct |
| 5 | 405 | Not allowed | Using of any method other than POST |
| 6 | 415 | Unsupported Media Type | If the Content-Type is not supported on server |
| 7 | 422 | Unprocessable entity | This response status code indicates that the server understands the content type of the request entity, and the syntax of the request entity is correct, but it was unable to process the contained instructions. |
| 8 | 428 | Precondition | This response status code indicates that the server requires the request to be conditional. Typically, this means that a required **precondition** header, such as If-Match , is missing |
| 8 | 429 | Too many Request | When API limit is exceeded |
| 5 | 500 | Internal server Error | If there is some problem at server end |

## Naming definition

This is applicable for

1. URL naming convention
2. Resource naming
3. Parameter naming

For parameter naming convention, snake\_case is followed in the system

For URL naming conventons, the HTTP method like GET, POST identify the operation type and hence the URL does not contain verbs words like get . It contains the nouns like users, etc.

Since we are using POST method for sending request, the operation type is not coming out clearly from the HTTP method. Hence verbs like get are used. This is deviation from standard practises and hightlighted here.

The URL names including resource names would be as follows

|  |  |  |
| --- | --- | --- |
| Service name | URL | Remarks |
| Login | https://www.gn.com/authentication/login |  |
| Logout | https://www.gn.com/authentication/logout |  |
| Contact | https://www.gn.com/contacts/get-contact-information | Full word information and not info |
| Organization | https://www.gn.com/organizations/get-organization-information |  |
| Recommendation | https://www.gn.com/recommendations/get-recommendation-information |  |
| Contribution | https://www.gn.com/contribution/get-contribution-information |  |
| New Fund Setup | https://www.gn.com/funds/get-new-fund-information |  |

## HTTP Based Authentication and Authorization

There are many ways to authenticate and authorize the user

First, to be clear on definitions, there are two separate actions usually discussed together:

##### Authentication:

Involves verifying who the person says he/she is. This may involve checking a username/password or checking that a token is signed and not expired. Authentication does not say this person can access a particular resource.

##### Authorization

Involves checking resources that the user is authorized to access or modify via defined roles or claims. For example, the authenticated user is authorized for read access to a database but not allowed to modify it. The same can be applied to your API. Maybe most users can access certain resources or endpoints, but special admin users have privileged access.

We are only talking about authentication mechanism here.

There are different ways of authenticating user

1. Basic
2. Bearer
3. Digest

HTTP Bearer Authentication scheme will be used.

Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens.

The client must send this token in the Authorization header when making requests to protected resources:  
Authorization: Bearer <token>

There are 2 ways of generating token

1. JWT token
2. Opaque Token also known as API keys

In our case, we are creating Opaque Token.

Opaque tokens are literally what they sounds like. Instead of storing user identity and claims in the token, the opaque token is simply a primary key that references a database entry which has the data. Of course, there is the added complexity of maintaining the K/V store and the auth server. Depending on your architecture, each service has to handshake with the auth server to get the claims or roles

The reason being that, if an API is limited specifically in functionality where “read” is the only possible command, an opaque token can be an adequate solution. Without the need to edit, modify, or delete, security is a lower concern.

The second question is How the token should be passed in the HTTP request. ?

There are 5 ways

1. Placing in URL – not recommended as URL are exposed and subject to security risk
2. Cookies
3. HTTP Header (Authorization Header)
4. Custom Heder
5. Body Data

We are using HTTP header.

So in summary, we are implementing the following

1. Opaque Token
2. Bearer Authentication
3. Authorization Header to transmit the token