Intuit

intuit_logo

IDS Explorer

Technical Specification Document

Version 1.0 | 6 Nov 2012

Revision History

Please keep the latest version on top

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver** | **Change Description** | **Sections** | **Date** | **Author** | **Reviewer** |
| 1 | Initial Draft |  | 6th Nov 2012 | Manjunath S |  |
|  |  |  |  |  |  |

References

Reader should read this document in conjunction with the following documents

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Document Name** | **Ver** | **Location** |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[References 2](#_Toc340866606)

[Purpose of this document 5](#_Toc340866607)

[Scope of this Project 5](#_Toc340866608)

[Definitions, Abbreviation and Acronyms 5](#_Toc340866609)

[1 Introduction 6](#_Toc340866610)

[2 Project Overview 6](#_Toc340866611)

[3 Interface Specification 7](#_Toc340866612)

[3.1 Web APIs Considered 7](#_Toc340866613)

[3.2 Node Modules Considered 13](#_Toc340866614)

[3.2.1 Express 14](#_Toc340866615)

[3.2.2 Jade 14](#_Toc340866616)

[3.2.3 Redis 15](#_Toc340866617)

[3.2.4 OAuth 16](#_Toc340866618)

[3.2.5 Passport 16](#_Toc340866619)

[3.2.6 XSD 16](#_Toc340866620)

[3.3 External interfaces 17](#_Toc340866621)

[4 Module Design 17](#_Toc340866622)

[5 Design Approach 18](#_Toc340866623)

[5.1 Interface description 18](#_Toc340866624)

[5.2 High Level Design 20](#_Toc340866625)

[5.3 Application Flow Design 22](#_Toc340866626)

[5.4 Page level Design 23](#_Toc340866627)

[5.4.1 Home Page 23](#_Toc340866628)

[5.4.2 Level 2 Page (Console Page) 24](#_Toc340866629)

[5.5 Data Design 27](#_Toc340866630)

[5.5.1 Data Structures 27](#_Toc340866631)

[5.5.2 Entity Design (No SQL Structure) 28](#_Toc340866632)

[5.6 Program Specification 29](#_Toc340866633)

[5.6.1 getSavedInfo 29](#_Toc340866634)

[5.6.2 saveCredentials 30](#_Toc340866635)

[5.6.3 saveRequest 30](#_Toc340866636)

[5.6.4 removeCredentials 31](#_Toc340866637)

[5.6.5 handleCredentials 31](#_Toc340866638)

[5.6.6 oauth 32](#_Toc340866639)

[5.6.7 oauthSuccess 32](#_Toc340866640)

[5.7 Configuration 33](#_Toc340866641)

[5.7.1 apiconfig.json 33](#_Toc340866642)

[5.7.2 lowerlevelconfig.json 34](#_Toc340866643)

[5.8 Error & Exception handling 35](#_Toc340866644)

[5.9 Logging 36](#_Toc340866645)

[5.10 Unit testing Guidelines 38](#_Toc340866646)

Purpose of this document

The purpose of this document is to give high level overview, define the scope of work and to include a high and a module level design, the UI Mockups and detailed technical approaches for the project "IDS Explorer".

Scope of this Project

Development of a live Interactive Documentation System using Mashery/IODocs library for V2 QBO, V2 QBD, V3 QBO, V3 QBD and AggCat Restful web APIs. (Please refer the section 3.1 for more details).

Testing this console for all the above said Web APIs. Manual test cases are written and executed.

Development of a tool which converts XSDs provided by Intuit for Restful services into JSON format..

Web UI to support HTML 5 standards. UI mockup screens will be frozen during Sprint 1 upon Intuit review and approval.

All the browsers which supports HTML 5 will be considered and cross-browser testing scope is limited to – Internet Explorer (9, 8), Chrome (Latest two versions), Firefox (Latest two versions) and Safari (Latest two versions).

User personalization will be based on Intuit OpenId Authentication and user can persist Security Tokens.

Authorization and obtaining security tokens are limited to support given by IODOCs library.

Deployment to Cloud platform is limited to either AWS or Windows Azure.

Definitions, Abbreviation and Acronyms

Following table has description for terms used in this document:

|  |  |
| --- | --- |
| **Acronym** | **Description** |
| IDS | Intuit Data Services |
| IPP | Intuit Partner Platform |
| OAuth | Open Authorization |
| QBD | QuickBooks Desktop |
| QBO | QuickBooks Online |
| JSON | JavaScript Object Notation |
| AggCat | Aggregation and Categorization |

# Introduction

**One place to Learn, Test and Debug Intuit’s RESTful web APIs live!**

IDS API Explorer is an interactive interface that lets developers (also, testers) to execute V2 QBO, V2 QBD, V3 QBO, V3 QBD and AggCatAPI calls right from the API documentation.

With IDS API Explorer:

* There’s no need to spend time toggling between different interfaces or cut/paste codes. Spend more time in coding.
* Each API/Method is listed with a complete list of available parameters.
* Input the details and **with just one click**, get the request URI, response header and response body in JSON or XML formats.

# Project Overview

IDS Explorer makes it easier for developers to learn and use an API. You can think of a Console as a "GUI for an API". Featuring an easy-to-use interface, IDS Explorer provides a GUI for exploring an API's resources and executing its methods.

Under the scope of this project, IDS Explorer provides Web Consoles for V2 QBO, V2 QBD, V3 QBO, V3 QBD and AggCat Restful API providers. It has a simple functional workflow. Just navigate to the required API Provider, choose an API, select a method, and enter the authentication (Security Keys) information and parameters, and then click ‘Try It’ button. That’s it; your request and the response codes are displayed. Tracing API calls is just that easy.

Technology considered is Node.js with Mashery/IODocs framework. It uses Redis KeyValue Store (NoSQL) for persisting data. Application should support end users on standard HTML 5 supported browsers.

IDS Explorer is designed to accommodate any number of API providers (although scope of this project is to provide three of the above mentioned providers). The base requirement for creating a console for a new API Provider is JSON Description file which can be created using the JSON Meta data provider tool which is developed as part of this project. JSON Description file can be constructed manually as well.

# Interface Specification

This section describes the different interfaces that Node modules use to communicate with each other or with the environment.

## Web APIs Considered

|  |  |  |  |
| --- | --- | --- | --- |
| **Entity Name** | **Service** | **Operations Supported** | **HTTP Methods Supported** |
| Customer | V3 QBD | Create, Retrieve, Update, Delete, Sparse Update | GET, POST |
| Invoice | V3 QBD | Create, Retrieve, Update, Delete, Sparse Update | GET, POST |
| Item | V3 QBD | Retrieve | GET |
| TaxCode | V3 QBD | Retrieve | GET |
| TaxRate | V3 QBD | Retrieve | GET |
| Term | V3 QBD | Retrieve | GET |

|  |  |  |  |
| --- | --- | --- | --- |
| **Entity Name** | **Service** | **Operations Supported** | **HTTP Methods Supported** |
| Account | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Bill | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| BillPayment | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| CashPurchase | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| ChangeDataDeleted(Beta) | V2 QBO | Retrieve | POST |
| Check | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Class | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| CompanyMetaData | V2 QBO | Retrieve | GET, POST |
| CreditCardCharge | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Customer | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Employee | V2 QBO | Retrieve | GET, POST |
| Entitlements | V2 QBO | Retrieve | GET |
| Estimate | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Invoice | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Item | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Job | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| JournalEntry | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Payment | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| PaymentMethod | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Preferences | V2 QBO | Retrieve | GET, POST |
| SalesReceipt | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| SalesTerm | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| TimeActivity | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |
| Vendor | V2 QBO | Create, Retrieve, Update, Delete | GET, POST |

|  |  |  |  |
| --- | --- | --- | --- |
| **Entity Name** | **Service** | **Operations Supported** | **HTTP Methods Supported** |
| Account | V2 QBD | Create, Retrieve, Update | GET, POST |
| AccountBalances | V2 QBD | Retrieve | GET, POST |
| BalanceSheet | V2 QBD | Retrieve | GET, POST |
| BalanceSheetStd | V2 QBD | Retrieve | POST |
| Bill | V2 QBD | Create, Retrieve, Update | GET, POST |
| BillPayment | V2 QBD | Retrieve | GET, POST |
| BillPaymentCreditCard | V2 QBD | Retrieve | GET, POST |
| BOMComponent | V2 QBD | Retrieve | GET, POST |
| BuildAssembly | V2 QBD | Retrieve | GET, POST |
| Charge | V2 QBD | Retrieve | GET, POST |
| Check | V2 QBD | Create, Retrieve, Update | GET, POST |
| Class | V2 QBD | Retrieve | GET, POST |
| Company | V2 QBD | Create, Retrieve, Update | GET, POST |
| CreditCardCharge | V2 QBD | Create, Retrieve, Update | GET, POST |
| CreditCardCredit | V2 QBD | Create, Retrieve, Update | GET, POST |
| CreditCardRefund | V2 QBD | Retrieve | GET, POST |
| CreditMemo | V2 QBD | Create, Retrieve, Update | GET, POST |
| CurrencyInfo | V2 QBD | Retrieve | GET, POST |
| Customer | V2 QBD | Create, Retrieve, Update | GET, POST |
| CustomerMsg | V2 QBD | Retrieve | GET, POST |
| CustomersWhoOweMe | V2 QBD | Retrieve | GET |
| CustomerType | V2 QBD | Retrieve | GET, POST |
| CustomTxnDetail | V2 QBD | Retrieve | POST |
| Deposit | V2 QBD | Retrieve | GET, POST |
| Discount | V2 QBD | Retrieve | GET, POST |
| Employee | V2 QBD | Create, Retrieve, Update | GET, POST |
| Estimate | V2 QBD | Create, Retrieve, Update | GET, POST |
| FixedAsset | V2 QBD | Retrieve | GET, POST |
| IncomeBreakDown | V2 QBD | Retrieve | GET, POST |
| InventoryAdjustment | V2 QBD | Retrieve | GET, POST |
| InventorySite | V2 QBD | Retrieve | GET, POST |
| InventoryTransfer | V2 QBD | Retrieve | GET, POST |
| Invoice | V2 QBD | Create, Retrieve, Update | GET, POST |
| Item | V2 QBD | Retrieve | GET, POST |
| ItemConsolidated | V2 QBD | Retrieve | GET, POST |
| ItemReceipt | V2 QBD | Create, Retrieve, Update | GET, POST |
| Job | V2 QBD | Create, Retrieve, Update | GET, POST |
| JobType | V2 QBD | Retrieve | GET, POST |
| JournalEntry | V2 QBD | Create, Retrieve, Update | GET, POST |
| NameValue | V2 QBD | Retrieve | GET, POST |
| OtherName | V2 QBD | Retrieve | GET, POST |
| Payment | V2 QBD | Create, Retrieve, Update | GET, POST |
| PaymentMethod | V2 QBD | Retrieve | GET, POST |
| PayrollItem | V2 QBD | Retrieve | GET, POST |
| PayrollItemNonWage | V2 QBD | Retrieve | GET, POST |
| ProfitAndLoss | V2 QBD | Retrieve | GET, POST |
| Preferences | V2 QBD | Retrieve | POST |
| PurchaseOrder | V2 QBD | Create, Retrieve, Update | GET, POST |
| SalesOrder | V2 QBD | Create, Retrieve, Update | GET, POST |
| SalesReceipt | V2 QBD | Create, Retrieve, Update | GET, POST |
| SalesRep | V2 QBD | Retrieve | GET, POST |
| SalesSummary | V2 QBD | Retrieve | GET, POST |
| SalesTax | V2 QBD | Retrieve | GET, POST |
| SalesTaxCode | V2 QBD | Retrieve | GET, POST |
| SalesTaxGroup | V2 QBD | Retrieve | GET, POST |
| SalesTerm | V2 QBD | Retrieve | GET, POST |
| ShipMethod | V2 QBD | Retrieve | GET, POST |
| Status | V2 QBD | Retrieve | POST |
| Summary 1099 | V2 QBD | Retrieve | POST |
| SyncActivity | V2 QBD | Retrieve | POST |
| Task | V2 QBD | Retrieve | POST |
| TemplateName | V2 QBD | Retrieve | GET, POST |
| TimeActivity | V2 QBD | Create, Retrieve, Update, Delete, Revert | GET, POST |
| TopCustomersBySales | V2 QBD | Retrieve | GET, POST |
| UOM | V2 QBD | Retrieve | GET, POST |
| Vendor | V2 QBD | Create, Retrieve, Update | GET, POST |
| VendorCredit | V2 QBD | Retrieve | GET, POST |
| VendorType | V2 QBD | Retrieve | GET, POST |

|  |  |  |
| --- | --- | --- |
| **API Name** | **Service** | **HTTP Method Name** |
| getInstitutions | AggCat | GET |
| getInstitutionDetails | AggCat | GET |
| discoverAndAddAccounts | AggCat | POST |
| getCustomerAccounts | AggCat | GET |
| getLoginAccounts | AggCat | GET |
| getAccount | AggCat | GET |
| getInvestmentPositions | AggCat | GET |
| listFiles | AggCat (Batch) | GET |
| getFileData | AggCat (Batch) | GET |
| deleteFile | AggCat (Batch) | DELETE |
| updateInstitutionLogin | AggCat | PUT |
| updateAccountType | AggCat | PUT |
| deleteAccount | AggCat | DELETE |
| deleteCustomer | AggCat | DELETE |

## Node Modules Considered



Modules considered for this project are:

1. Express – Web application framework
2. Jade – Template Engine
3. Redis – Key value store. Usually used for data that lives for a small period of time
4. Oauth – allows users to authenticate against OAUTH providers, and thus act as OAuth consumer
5. Passport – authentication middleware used for OpenId authentication in this project
6. XSD – XSD parser for node.js

Since node.js is relatively a new language, we will cover some basics of this language in this document. Node.js is a single threaded, non-blocking, asynchronous and event based programming language.

Instead of running one process, waiting for the client to respond and then starting another process, Node.js runs the processes it has the data to run in the order it receives them. Then when the response comes back that's a new process in the queue, and the application just keeps juggling these requests. The overall design is such that Node developers are forced to keep each process very short. This makes node.js a great language for light messaging application with a high user volume.

### Express

Express is a Node application framework specifically built for web applications where developers have complete control over, how requests are handled. It is very easy to specify the static folders such as CSS and JavaScript. Also, specifying URLs and how to handle them is very straight forward and easy. Developer can also define dynamic URLs using patterns (Regular Expression) in URL. Express lets developers handle requests and specify views to render them. Express makes use of template engines to compile views to HTML.

**Example**:

app.get('/api', function(req, res) {

res.render('apiTemplate', { title: ‘example’ });

});

The above example shows how the URL ‘/api’ handles the GET requests and renders template with name “apiTemplate.jade” (considering express is configured to handle view as jade templates) and variable with name ‘title’ will be set with value ‘example’ in jade apiTemplate.

### Jade

Jade is a high performance template engine for Node.js and the default rendering engine for the express web framework. It is heavily influenced by Haml. Template engines are used in most web frameworks to generate HTML. Typical features include variables and loops.

Jade is an indentation based template engine.

* It is much more terse than HTML
* It uses indentation to declare the hierarchy of an HTML document.
* Developers do not have to use tags in jade. The <> characters are automatically added when the template is compiled.

When a web page is generated, Jade will compile the template to HTML. The output is exactly the same as vanilla HTML. Jade allows applications to output data dynamically to HTML. While building a page structure the real power of Jade comes from manipulating data and outputting it to HTML. Jade uses two characters to decide how it should interpret code:

1. The minus sign (-) is used to tell Jade that the code that follows the sign should be executed and the result of the execution (if any) should be outputted. This is most commonly used with loops where developers want to output an array of data to the page.
2. The second is the equal sign (=). This tells the interpreter that the code should be evaluated and then outputted.

**Example**:

**HTML**

<div class=”wrapper”>

<h1>My holiday in Prague</h1>

<p>I had a great holiday in Prague where I met some great people.</p>

<img src=”images/photo.jpg” alt=”Me on holiday!” />

</div>

**Jade**

.wrapper

h1 My holiday in Prague

p I had a great holiday in Prague where I met some great people

img(src=’images/photo.jpg’ alt=’Me on holiday’)

### Redis

Redis is an open-source, networked, in-memory, key-value data store with optional durability. In its outer layer, the Redis data model is a dictionary where keys are mapped to values. One of the main differences between Redis and other structured storage systems is that values are not limited to strings. In addition to strings, the following abstract data types are supported:

* Lists of strings
* Sets of strings (collections of non-repeating unsorted elements)
* Sorted sets of strings (collections of non-repeating elements ordered by a floating-point number called score)
* Hashes where keys and values are strings

The type of a value determines what operations (called commands) are available for the value itself. Redis supports high level atomic server side operations such as, intersection, union, sets, sorted sets and sorting of lists.

### OAuth

This module allows users to authenticate against OAUTH providers, and thus act as OAuth consumers. It also has support for OAuth Echo, which is used for communicating with 3rd party media providers such as TwitPic and yFrog. It has been tested on websites such as, Twitter (http://twitter.com), term.ie (http://term.ie/oauth/example/), TwitPic, and Yahoo!

It also provides rudimentary OAuth2 support, tested against Facebook, github, foursquare, goggle and Janrain.

### Passport

Passport is authentication middleware for Node.js. Since it is extremely flexible and modular, Passport can be unobtrusively plugged into any [Connect](http://senchalabs.github.com/connect/)/[Express](http://expressjs.com/)-based web application. A comprehensive set of strategies such as Facebook and Twitter, authentication using a username and password, and more.  It is designed to serve a singular purpose: authenticate requests. When writing modules, encapsulation is a virtue, so Passport delegates all other functionality to the application. This separation of concerns keeps code clean and maintainable, and makes Passport extremely easy to integrate into an application.

In modern web applications, authentication can take a variety of forms. Traditionally, users log in by providing a username and password. With the rise of social networking, single sign-on using an OAuth provider such as Facebook or Twitter has become a popular authentication method. Services that expose an API often require token-based credentials to protect access.

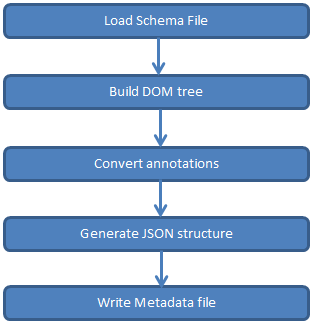
Passport recognizes that each application has unique authentication requirements. Authentication mechanisms, known as strategies, are packaged as individual modules. Applications can choose which strategies to employ, without creating unnecessary dependencies. Despite the complexities involved in authentication, code does not have to be complicated.

**Example**:

app.post('/login', passport.authenticate('local', { successRedirect: '/', failureRedirect: '/login' }));

### XSD

This module parses the XSD file and builds DOM structure of schema definition in JSON format. We will modify this module to generate Metadata JSON file which will have the parameters for each entity and enums as well as its definition. The key names in this JSON structure would be added in lowerlevelconfig.json to link the entity with endpoints in api service.

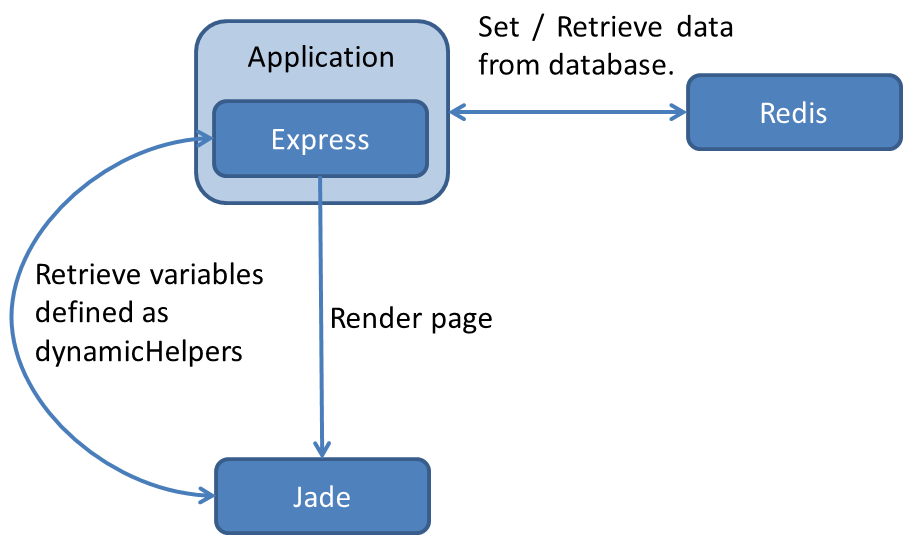


## External interfaces

Following are the list of external interfaces which are dealt by Node console application:

* OpenId Authenticatoin:
* Intuit OpenID provider URL: <https://openid.intuit.com/OpenId/Provider>
* OpenID discovery URL: <https://openid.intuit.com/openid/xrds>
  + OAuth Authorization:
* OAuth Server Name with URI: <https://oauth.intuit.com/oauth/v1/>
  + SAML Assertion:
* Aggregation & Categorization services use SAML Assertions to get Access tokens.
* Like OAuth, SAML Gateway is another external interface.

# Module Design



Express let developers define variables as dynamicHelpers which are retrieved by template engines every time a page needs to be rendered. The dynamicHelpers are used to define variables that are to be used in multiple templates.

**Example of dynamicHelper:**

app.dynamicHelpers({

defaultParams: function(req, res) {

if(req.params.api && req.session[req.params.api] && req.session[req.params.api]['params'])

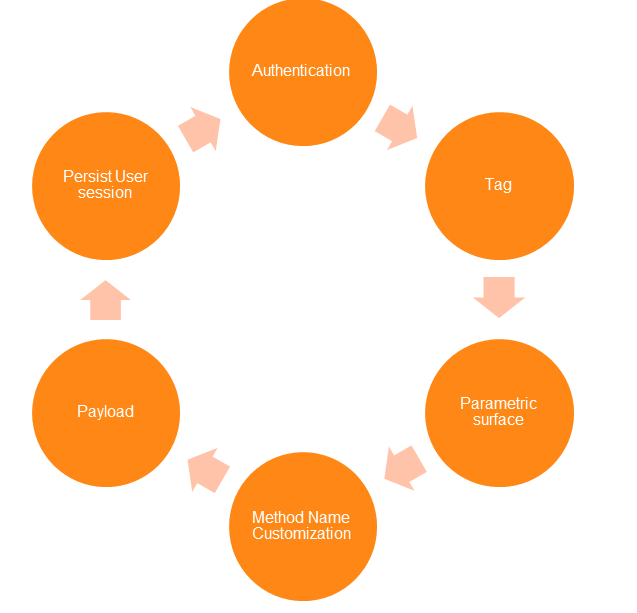
return req.session[req.params.api]['params'];

}

});

# Design Approach

## Interface description



**Authentication:** Indicates whether an API method requires authentication. Out of box, IODocs framework supports OAuth three legged flow which can be used for V2 QBO, v2 QBD, V3 QBO and V3 QBD services with few small modifications. But AggCat services require custom Node module which will pass SAML Assertion to OAuth end point to obtain the Access token. As per requirement, all the API Methods require OAuth Access tokens to work.

**Tag:** Set of configuration settings for an API provider that enables the Console to group related API methods into categories. Currently following categories are considered:

1. AggCat
2. AggCat Batch
3. V2 QBO
4. V2 QBD
5. V3 QBO
6. V3 QBD

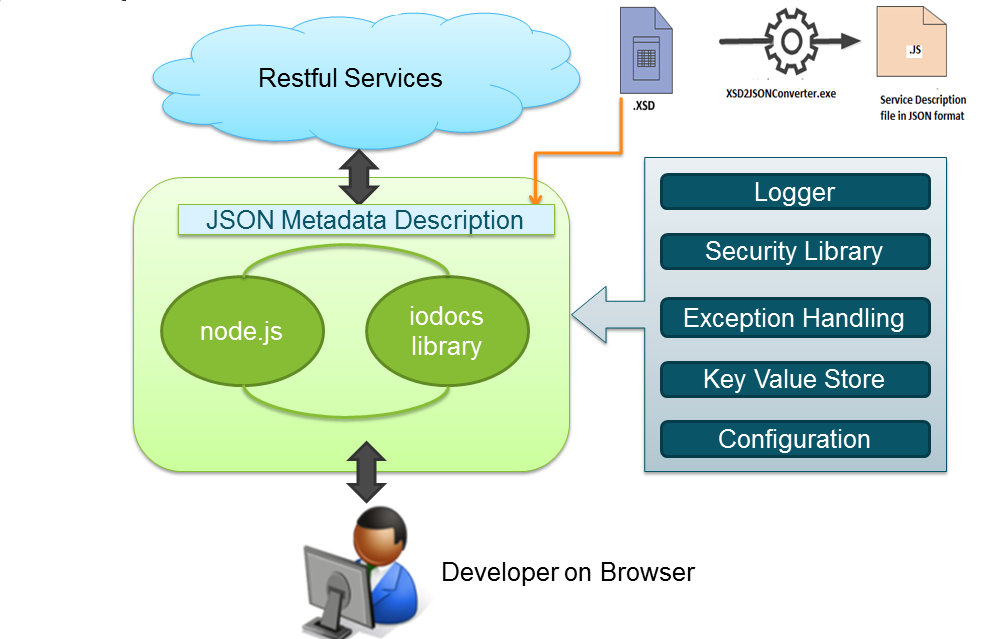
**Parametric Surface:** Enables end users to define and pass a set of parameters to the HTTP Request using HTML input controls (Text boxes and dropdowns). These controls are auto generated using JSON Definition file.

**Method Name Customization:** None of the intuit services (Services which are under scope) follow true RESTful Architecture. They are more hybrids (RPC-REST Style) in nature than true RESTful Architecture. None of them use true HTTP Methods, instead use custom web method names and it becomes impossible to generate JSON Application Definition files (or WADL) through automation tool. Although we are generating JSON Metadata definition files through a tool, Method name customization is still a manual process and IODOC framework customization is also needed.

**Payload:** A JSON or XML request body (often used with POST or PUT operations) which is displayed to the user in the Console and is submitted to Intuit’s API as part of the request message.

**Persist User Session:** EndUser can save Access Tokens and any other security tokens needed for Web API calls. The application also allows users to store and retrieve details of last Web API request made. Persisting data is done through Redis module.

## High Level Design



**XSD2JSONConvertor tool:** This tool will convert XSDs obtained from intuit to JSON Service Description file. Since Intuit services are not developed on true RESTful principles, we need to have another configuration file which gives information about custom web methods used. Creating this method description file is a manual process.

**Logger**: Winston library is considered as it is designed to be a simple and universal logging library with support for multiple transports. Winston library can output log information to a console or a local file.

Refer the **Logging** section for more information.

**Error and Exception Handling:** All the errors are handled through err object of JavaScript and, capturing and handling errors are going to be manual. Refer the **Error and Exception Handling** section for more information.

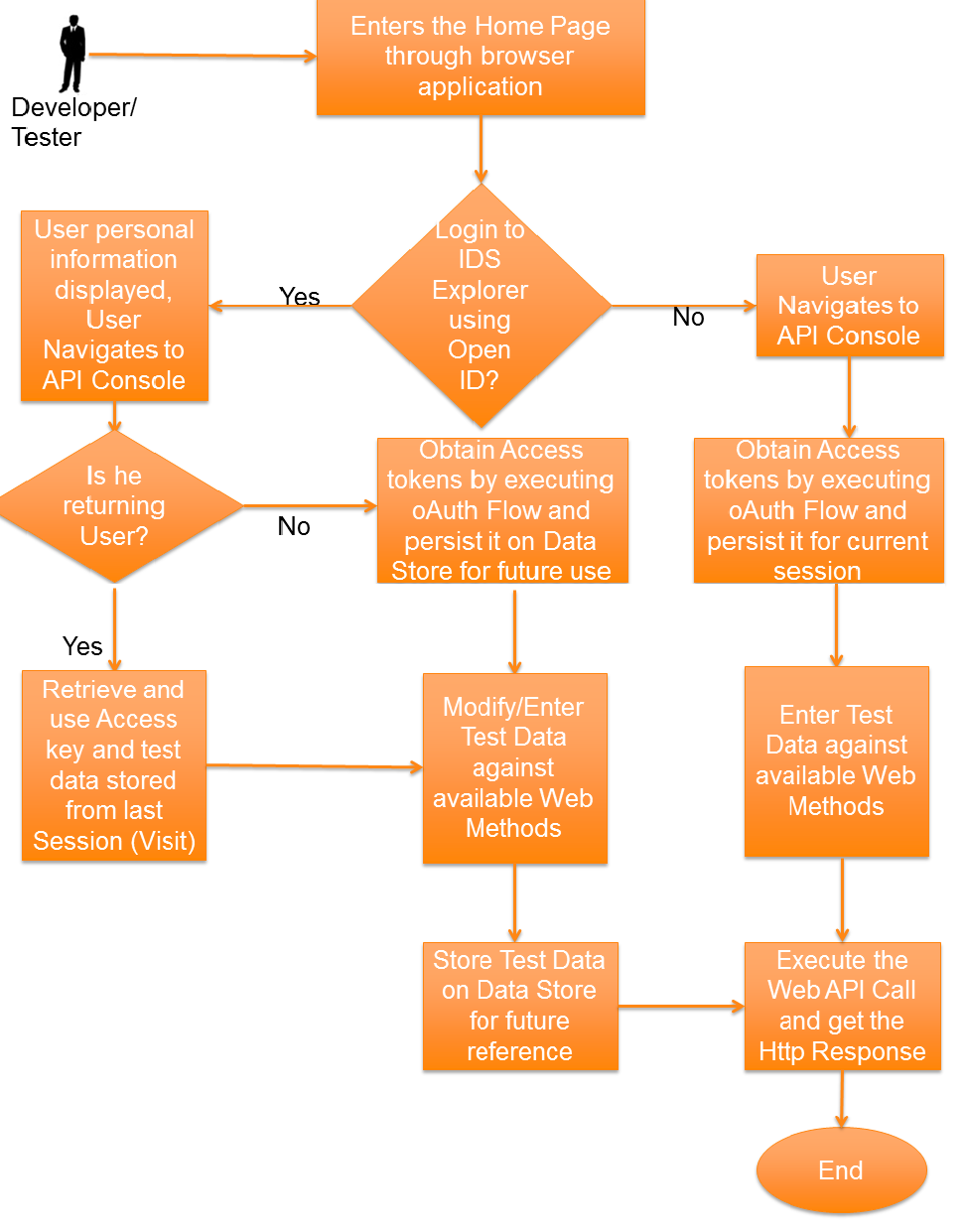
**Security Library:** Passportmodule handles both OpenId and SAML Assertion and, OAuth module handles getting Access token using various input parameters. Each and every Web API request needs Access token to execute the call.

**Key Value Store:** Redis service which works on NoSQL principle is used as data store. Redis is an open source, advanced key-value store. It is often referred as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets.

**Configuration:** This component uses two levels of configuration files:

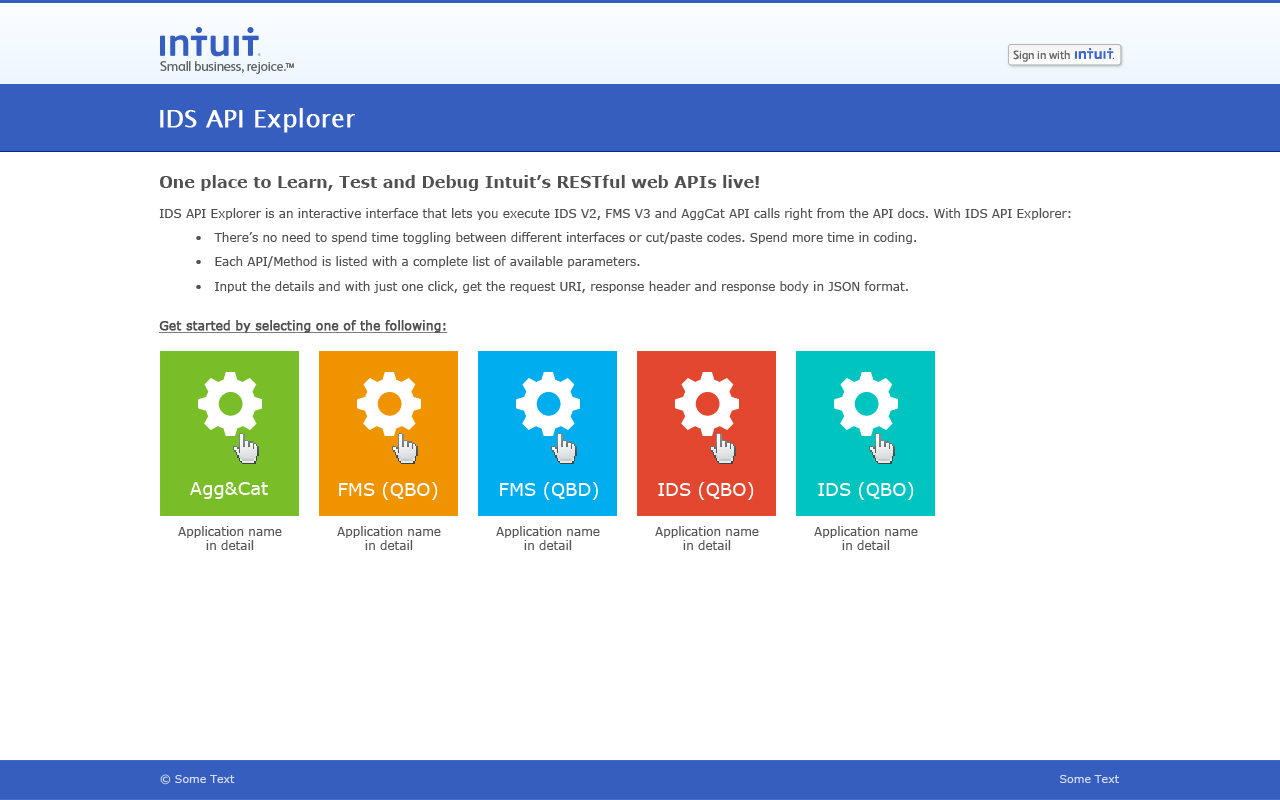
* APIConfig.json: This contains about top-level service information. This will be the master configuration file for entire application.
* LowerLevelconfig.Json: This contains information about methods and endpoints for API Providers. Under the scope of this project we will be having around 6 lowerleveconfig.json files.

## Application Flow Design



## Page level Design

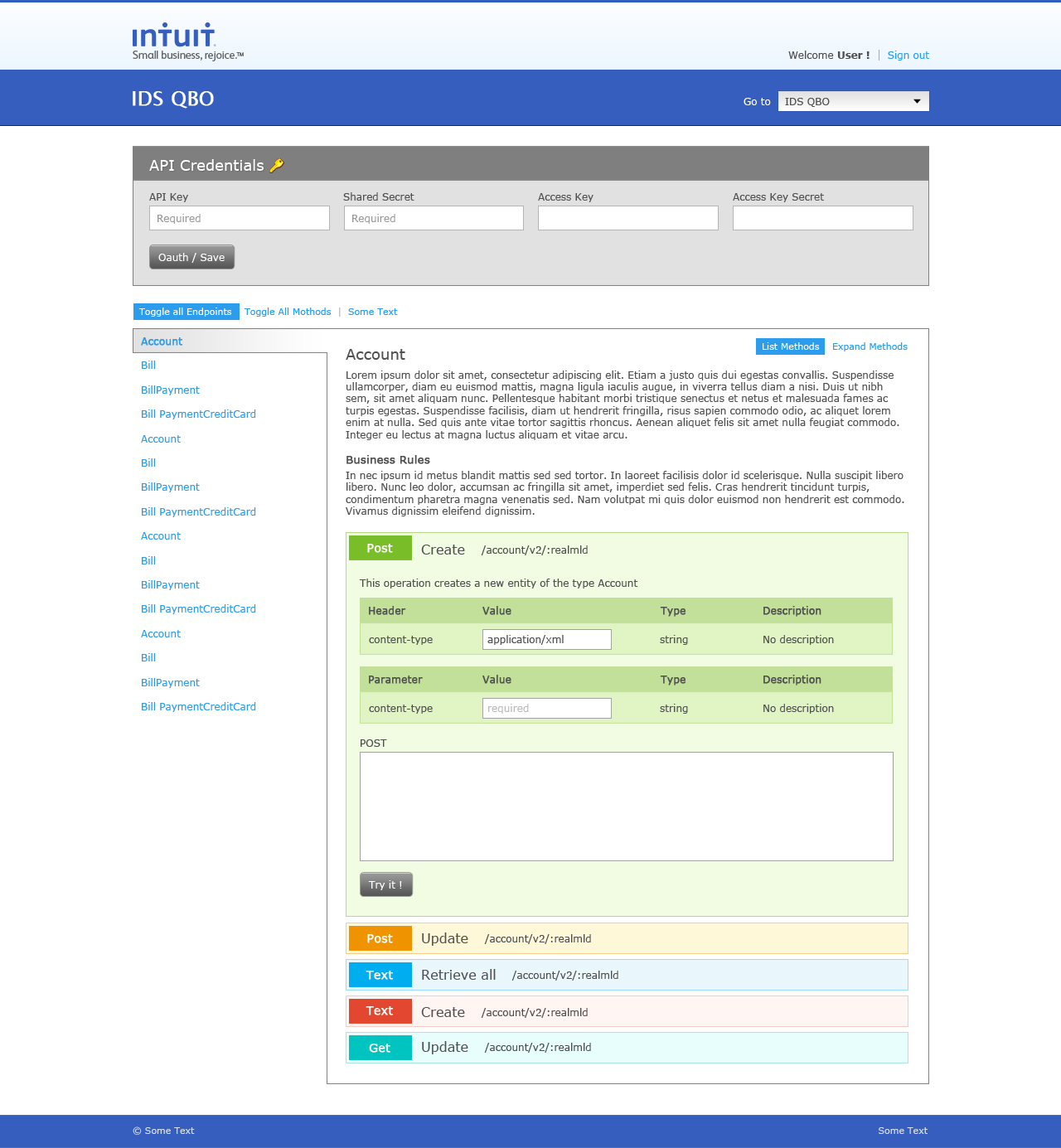
### Home Page



Home page lists all the categories for the developer to select. Application reads file “apiconfig.json” present in folder “data” for the list of categories. For the Json structures in apiconfig.json refer 5.7.1.

Developer will be provided with “sign in with intuit” button which will redirect user to “openid.intuit.com” for logging in. The control then returns to the application with user’s profile information. Application then displays “Welcome <user>” and displays logout button. On clicking the logout button application logs user out from system.

### Level 2 Page (Console Page)



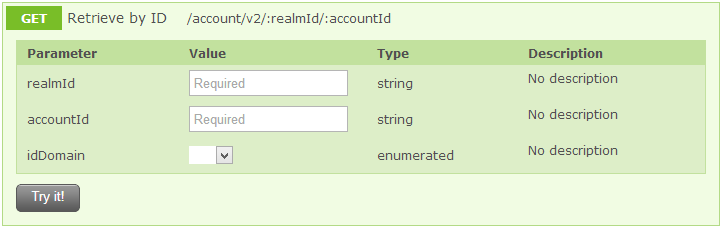
This page lists all the endpoints for API service. Application will lookup and read endpoint information from file with name corresponding to category key defined in apiconfig.json. For example: category V2 QBD will have a file named V2-QBD.json.

Each endpoint will have documentation and interactive methods (CRUD operations). Each method will have three sections:

1. Header
2. Parameter
3. Request Body

Refer 5.7 for the Json structure to configure above mentioned sections.

#### **Handling Get Request:**

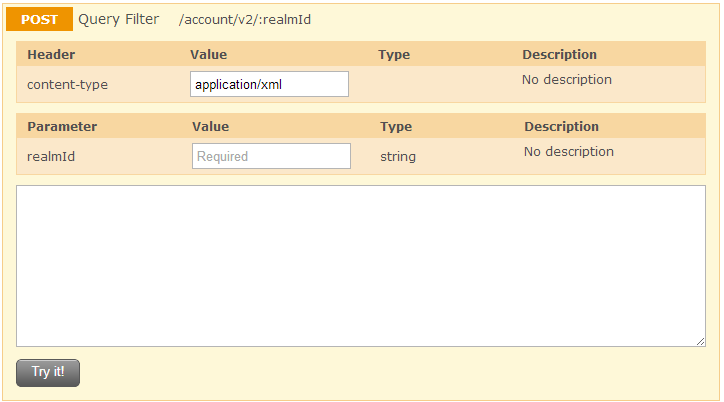


The URL which has dynamic values should be present in parameters list. The value of the parameter text box will replace the variable in URL. The rest of the parameters will be appended to the URL.

URL formed will be:

https://<baseURL>/<privatePath>/account/v2/<realmId>/<accountId>?idDomain=<idDomain>

#### **Handling Post Request:**



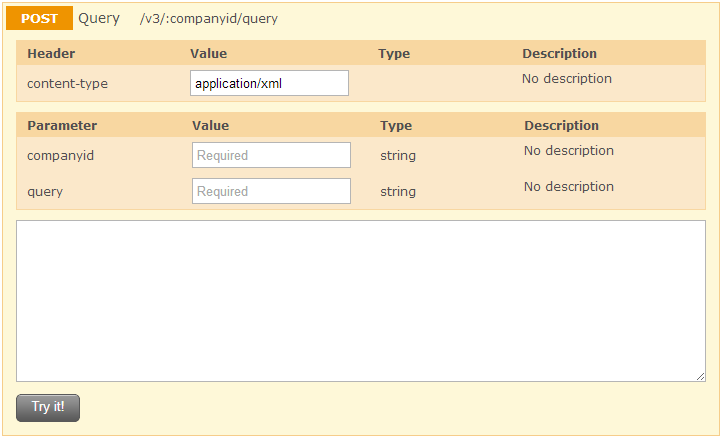
The URL which has dynamic values should be present in parameters list. The value of the parameter text box will replace the variable in URL. The rest of the parameters will be appended to the URL.

URL formed will be:

https://<baseURL>/<privatePath>/account/v2/<realmId>

The filter XML (for V2 QBD) or filter string (for V2 QBO) that is written in Request body textbox will be sent as POST body.

#### **Handling Post Request along with parameter:**



The URL which has dynamic values should be present in parameters list. The value of the parameter text box will replace the variable in URL. The rest of the parameters will be appended to the URL.

URL formed will be:

https://<baseURL>/<privatePath>/v3/<companyid>/query?query=<query string>

The query string written in Request body textbox will be sent as POST body.

## Data Design

### Data Structures

Redis is an open source, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets. In order to achieve its outstanding performance, Redis works with an in-memory dataset. Depending on the use case, user can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log. Redis also supports trivial-to-setup master-slave replication, with very fast non-blocking first synchronization, auto-reconnection on net split and so forth. Other features include a simple check-and-set mechanism, pub/sub and configuration settings to make Redis behave like a cache.

Our application generates the keys and keeps the values against those keys. The format of the key is <<**Key\_String**>>**:**<<**Service\_Name**>>**:**<<**Variable\_Name**>>.

Where,

<<Key\_String>> is either the SessionId or the logged in users Email Id.

<<Service\_Name>> is the name of the service or service provider.

<<Variable\_Name>> is the name of the variable which describes the value.

**Example**:  
If you want to store the API key for AggCat Rest service for session Id “xyz” and user email id “[abc@a.com](mailto:abc@a.com)” then the key in Radis DB would be “xyz\_abc@a.com:AggCat:apiKey”.

The same format is being used for all kind of keys. Few are given below:

// key for API Key

<<Key\_String>>:AggCat:apiKey

// key for API secret

<<Key\_String>>:AggCat:apiSecret

// key for access token

<<Key\_String>>:AggCat:accessToken

// key for access secret

<<Key\_String>>:AggCat:accessSecret

Following are a few Redis commands, (Run the commands from redis-client):

// to display all the available keys

redis>keys \*

// to display only AggCat related keys

redis>keys \*AggCat\*

// to create new key-value

redis>set key1 “value1”

// to get value for the key

redis>get key1

// to delete key

redis>del key1

// to remove all the keys from DB

redis>flushdb

### Entity Design (No SQL Structure)

Keys used for different REST services are listed in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Rest Services** | **Field Name** | **Key Name** | **Value** |
| AggCat,  AggCat Batch,  V2 QBD,  V2 QBO,  V3 QBD,  V3 QBO | defaultKey | <<Key\_String>>:<<Service\_Name>>:defaultKey | The default API key value |
| defaultSecret | <<Key\_String>>:<<Service\_Name>>:defaultSecret | The default API key secret value |
| defaultAccessKey | <<Key\_String>>:<<Service\_Name>>:defaultAccessKey | The default access key value |
| defaultAccessSecret | <<Key\_String>>:<<Service\_Name>>:defaultAccessSecret | The default access secret value |
| apiKey | <<Key\_String>>:<<Service\_Name>>: apiKey | The API key provided by user |
| apiSecret | <<Key\_String>>:<<Service\_Name>>:apiSecret | The API key secret provided by user |
| accessKey | <<Key\_String>>:<<Service\_Name>>:accessKey | The access key provided by user |
| accessSecret | <<Key\_String>>:<<Service\_Name>>:accessSecret | The access secret provided by user |
| oauthParams | <<Key\_String>>:<<Service\_Name>>:oauthParams | The params which are retrieved from oAuth flow |
| savedParams | <<Key\_String>>:<<Service\_Name>>:savedParams | The params which are provided by the user for each API call |
| savedHeaders | <<Key\_String>>:<<Service\_Name>>:savedHeaders | The header information which are stored while making a service call |
| savedRequestBody | <<Key\_String>>:<<Service\_Name>>:savedRequestBody | The actual request body which is being used |

## Program Specification

### getSavedInfo

|  |
| --- |
| **Usage :**  This function retrieves entity fields from the database for a particular user (if user is logged in) or for a particular session. And sets all fields in array req.session[apiName] where apiName is category selected by user.  **Input Parameters:**  req, res, next  **Errors Generated:**  on error this function will log the error and call next().  **Explanation:**  var apiName = req.params.api;  if(req.session.loggedin)  key = req.session.passport.user.emails[0].value + ":" + apiName;  else  key = req.sessionIDi + ':' + apiName;  read from database key + ':', key + ':accessTokenSecret', key + ':apiKey', key + ':apiSecret', key + ': oauthParams ', key + ':savedParams', key + ':savedRequestBody', key + ':savedHeaders'  on return of data from database (either err or result)  if err is set then call next()  if req.session[apiName] does not exist create req.session[apiName]  if result contains accessToken, accessTokenSecret, apiKey, apiSecret then set req.session[apiName].authed as true else set req.session[apiName].authed as false  if result contains oauthParams set req.session[apiName] as oauthParams else set as {}  if result contains savedParams set req.session[apiName] as savedParams else set as {}  if result contains savedRequestBody set req.session[apiName] as savedRequestBody else set as {}  if result contains savedHeaders set req.session[apiName] as savedHeaders else set as {}  call next() |

### saveCredentials

|  |
| --- |
| **Usage :**  This function saves entity fields corresponding to oauth (apiKey, apiSecret, accessToken, accessTokenSecret) to database for a particular user (if user is logged in) or for a particular session.  **Input Parameters:**  req, res, next  **Errors Generated:**  on error this function will log the error and call next()  **Explanation:**  var apiName = req.params.api;  if(req.session.loggedin)  key = req.session.passport.user.emails[0].value + ":" + apiName;  else  key = req.sessionID + ':' + apiName;  if req.body.key and req.body.secret and req.body.accessKey and req.body.accessSecret are set then  save req.body.key as key + ':apiKey' and save req.body.secret as key + ':apiSecret' and save req.body.accessKey as key + ':accessToken' and req.body.accessSecret as key + ':accessTokenSecret' in database  if req.session[apiName] does not exist or not set then set req.session[apiName] as {}  set req.session[apiName].authed as true  call next() |

### saveRequest

|  |
| --- |
| **Usage :**  This function saves entity fields corresponding to a request call (savedParams, savedRequestBody, savedHeaders) to database for a particular user (if user is logged in) or for a particular session.  **Input Parameters:**  req, res, next  **Errors Generated:**  on error this function will log the error and call next()  **Explanation:**  var apiName = req.params.api;  if(req.session.loggedin)  key = req.session.passport.user.emails[0].value + ":" + apiName;  else  key = req.sessionID + ':' + apiName;  set req.session[req.body.apiName].savedParams[req.body.endpointName+':'+req.body.methodName] = req.body.params // saves params for particular method in endpoint.  Serialize and save req.session[req.body.apiName].savedParams in database as savedParams  Set req.session[req.body.apiName].savedRequestBody[req.body.endpointName+':'+req.body.methodName] as req.body.requestBody  Serialize and save req.session[req.body.apiName].savedRequestBody in database as savedRequestBody  Set req.session[req.body.apiName].savedHeaders[req.body.endpointName+':'+req.body.methodName] as req.body.headers  Serialize and save req.session[req.body.apiName].savedHeaders in database as savedHeaders  call next() |

### removeCredentials

|  |
| --- |
| **Usage :**  This function removes entity fields corresponding to oauth (apiKey, apiSecret, accessToken, accessTokenSecret) from database for a particular user (if user is logged in) or for a particular session.  **Input Parameters:**  req, res, next  **Errors Generated:**  on error this function will log the error and call next()  **Explanation:**  var apiName = req.params.api;  if(req.session.loggedin)  key = req.session.passport.user.emails[0].value + ":" + apiName;  else  key = req.sessionID + ':' + apiName;  delete entity fields key + ':apiKey', key + ':apiSecret', key + ':requestToken', key + ':requestTokenSecret', key + ':accessToken', key + ':accessTokenSecret' from database  set req.session[apiName].authed as false  call next() |

### handleCredentials

|  |
| --- |
| **Usage :**  This function is called when user submits the credential form.  **Input Parameters:**  req, res, next  **Errors Generated:**  on error this function will log the error and call next()  **Explanation:**  If req.body.action is set as “remove” call removeCredentials().  Else if req.body.accessKey is not set or req.body.accessSecret is not set //i.e. not filled by the user  Call oauth()  Else  Call saveCredentials() //i.e all four fields are filled by user  call next() |

### oauth

|  |
| --- |
| **Usage :**  This function initiates the oauth flow.  **Input Parameters:**  req, res, next  **Errors Generated:**  on error this function will log the error and call next()  **Explanation:**  var apiName = req.body.apiName  if service requires oauth  set variables apiKey, apiSecret, refererURL, callbackURL with appropriate values and create new oauth object oa using these variables  if oauth type is configured as 'three-legged' and req.session[apiName].authed is not true  call oa.getOAuthRequestToken() // this fetches request token and secret  if err is set send back error message  else  if(req.session.loggedin)  key = req.session.passport.user.emails[0].value + ":" + apiName;  else  key = req.sessionID + ':' + apiName;  set apiKey as key + ':apiKey', apiSecret askey + ':apiSecret', oauthToken as key + ':requestToken', oauthTokenSecret as key + ':requestTokenSecret'  send back json { 'signin': apiConfig.oauth.signinURL + oauthToken } // which will open another window where user will authorize the access  call next() |

### oauthSuccess

|  |
| --- |
| **Usage :**  This function is called when a user authorizes the access.  **Input Parameters:**  req, res, next  **Errors Generated:**  on error this function will log the error and call next()  **Explanation:**  if(req.session.loggedin){  key = req.session.passport.user.emails[0].value + ':' + apiName;  }  else {  key = req.sessionID + ':' + apiName; // Unique key using the sessionID and API name to store tokens and secrets  }  Read key + ':requestToken', key + ':requestTokenSecret', key + ':apiKey', key + ':apiSecret' from database  If err returns log error and send back error message  Else create new oauth object as oa and call oa.getOAuthAccessToken()  If err is set send back error message  Else set oauthAccessToken as key + ':accessToken', oauthAccessTokenSecret as key + ':accessTokenSecret' and serialized req.query as key + ':params'  Then set req.session[apiName] as {}  Set req.session[apiName].authed as true  Set req.session[apiName].params as req.query  call next() |

## Configuration

### 

### apiconfig.json

This file defines the categories of API services available in the application and is kept in the data folder. The structure for JSON is as follows:

{

<<category Name1>> : { name value pairs of configuration for category Name1},

<<category Name2>> : { name value pairs of configuration for category Name2}

}

Name-Value pair for configuration includes:

|  |  |
| --- | --- |
| **Property** | **Description** |
| name | Name of Category |
| protocol | Protocol used by api service (http or https) |
| baseURL | Base URL of api service |
| auth | Type of authorization (oauth, keyValue) |
| privatePath | Base folder structure for Apis |
| Oauth | If auth type is oauth then value for this key will be JSON structure consisting:   |  |  | | --- | --- | | **Name** | **Description** | | type | three-legged or two-legged | | requestURL | URL for getting request token | | signinURL | URL for authentication | | accessURL | URL for getting access token | | Version | Version of oauth | | crypt | HMAC-SHA1 | |

### lowerlevelconfig.json

Every category will have its own JSON file describing the endpoints of api service. The structure of JSON is as follows:

{ “endpoints”: [ { name value pair describing endpoint } ] }

Name value pair for endpoint JSON contains:

|  |  |
| --- | --- |
| name | Name of endpoint |
| synopsis | Documentation for this endpoint |
| methods | JSON structure describing all the methods for this endpoint |

Name value pair for describing methods contains:

|  |  |
| --- | --- |
| methodName | Name of method |
| synopsis | Description of method |
| httpMethod | Type of HTTP Verb |
| uri | URI for the method excluding base URL and privatePath. If URI has dynamic value, then parameter with same name should be present in parameters array  For dynamic URI, append colon (:) before folder name  Ex: /account/v2/:**realmId**  Then realmId parameter should be present |
| requiresOAuth | Tells application whether oauth is required or not for this method call |
| requiresPostData | Tells application whether this method requires request body |
| headers | Array of JSON structure containing   |  |  | | --- | --- | | Name | Name of header | | Default | Default value to be displayed to user | |
| parameters | Array of JSON structure containing   |  |  | | --- | --- | | Name | Name of parameter | | Required | If parameter is part of URL it should be “Y” | | Default | Default value to be displayed to user | | Type | String, enumeratedList | | EnumeratedList | If type is enumeratedList value will be array of values |   **Note: These parameters will always be appended to the URL and not sent as POST request body.** |

## Error & Exception handling

Error handling in Node.js isn’t well-developed and we have to handle it internally. Error handling is very important for long running Node.js servers, since a single uncaught exception can bring the whole server down. Here are few possible errors that could happen in the API Explorer:

* The http request might encounter an error. These are signaled using an 'error' event.
* The JSON parse might fail. In which case, this synchronous call will throw an exception. Alternatively, if the structure is wrong, an exception will be thrown after a successful parse.
* The database store might fail. Our callback from the database uses a common Node practice of passing us a first parameter, "error", whose value is either undefined (nothing went wrong) or an Error object.
* Something goes unexpectedly wrong in the library. For example, if there's a DNS failure, Node's HTTPClient doesn't tell us using an 'Error' event, it just throws an (unhandled) exception to Node.

The above mentioned errors could be possible for both Synchronous and Asynchronous calls. How to handle those errors for Synchronous and Asynchronous calls are given below.

* For asynchronous code, the first argument of the callback is “error”, if an error happens it is the error object, if it doesn't it is null. Any other arguments follow the err argument.

// sample callback function

function (error, data, ...) {

if (error) {

// handle our error here

return(null);

}

// handle our data here

}

// how to use the callback error handling

divide (4, 2, function(err, result){

if ( err ) {

// handle the error, or forward it to the next callback

console.log(err);

return;

}

console.log('4/2=' + result);

});

* For synchronous code, if an error happens, returns the error.

// sample code to handle the error

var result = divideSync(4, 2);

if ( result instanceof Error ) {

// handle the error

console.log(result);

process.exit();

});

// how to use the callback error handling

divide (4, 2, function(err, result){

if ( err ) {

// handle the error, or forward it to the next callback

console.log(err);

return;

}

console.log('4/2=' + result);

});

As mentioned Node.js isn’t well developed, there could be other unhandled errors which would cause the server to go down. Here is the common code to catch unhandled errors:

// code to catch unhandled errors

process.on('uncaughtException', function(err) {

console.log(err);

});

// in addition to the above code, we can use the traditional

// JavaScript way of handling the error

try {

throw new Error('uncaughtException');

} catch (e) {

console.log(e);

}

## Logging

IDS Explorer provides application level logging using Winston library. Winston is designed to be a simple and universal logging library with support for multiple transports. A transport is essentially a storage device for your logs. Each instance of a Winston logger can have multiple transports configured at different levels. For example, one may want error logs to be stored in a persistent remote location (like a database), but all logs output to the console or a local file. Logging to console or a file is called transport.

There are two different ways to use Winston: directly via the default logger or by instantiating your own Logger.

* **Using the Default Logger:**

The default logger is accessible through the Winston module directly. Any method that you could call on an instance of a logger is available on the default logger. By default, only the Console transport is set on the default logger.

// importing Winston module

var winston = require('winston');

winston.log('info', 'Some text here!');

winston.info('Some text here');

* **Installing your own Logger:**

If you would prefer to manage the object lifetime of loggers you are free to instantiate them yourself:

// creating logger instance

var logger = new (winston.Logger)({

transports: [

new (winston.transports.Console)( { level: 'info' }),

new (winston.transports.File)({ filename: 'somefile.log' })

]

});

// logging

logger.log('info', 'Some text here!');

logger.info('Some text here');

**On Exception:**

By default, winston will exit after logging an uncaughtException. To avoid this, we need to set the property exitOnError to false.

// way to set the exitOnError to false to avoid exit

var logger = new (winston.Logger)({ exitOnError: false });

// or, like this:

logger.exitOnError = false;

It is also worth mentioning that the logger also emits an 'error' event which you should handle or suppress if you don't want unhandled exceptions:

// way to handle the ‘error’ event

logger.on('error', function (err) {

// Do Something

});

// Or just suppress them.

logger.emitErrs = false;

**Callbacks:**

Each instance of winston.Logger is also an instance of an EventEmitter. A log event will be raised each time a transport successfully logs a message:

logger.on('logging', function (transport, level, msg, meta) {

// [msg] and [meta] have now been logged at [level] to [transport]

});

logger.info('Some Text Here!', { seriously: true });

## Unit testing Guidelines

This section outlines the testing methodologies which could be used to test the web application.

**Usability testing:**

Usability testing plays a pivotal role with applications that are designed to make manual tasks easier. Usability testing includes the following guidelines:

The applications should:

* comply with accessibility standards
* have proper navigation between web pages
* use appropriate color combinations and best practices
* have user friendliness to all types of users, from novice to expert

**User acceptance testing:**

The objective of user acceptance testing is to make sure the application meets the expectations of the user. It ensures that the application is fit enough to be deployed and used effectively. The following are guidelines for user acceptance testing:

The application should:

* have browser compatibility
* have proper time outs and field widths

**Functional testing:**

Functional testing ensures that individual functions are working well. Test cases should ensure that boundary conditions are tested. Invalid inputs should prompt appropriate error messages. Functional testing can range from checking whether links are working to checking whether changes made by users on a web page are reflected.

Following are few of the functional tests:

* Database testing
* Configuration testing
* Compatibility testing
* Flow testing

**Interface testing:**

Conduct interface testing to ensure that individual components are connected properly. The output of one module should be fed to the intended module without any issues. The following are considerations for interface testing:

* Ensure that data flow occurs smoothly and as expected
* Ensure that the interfaces exposed by components are generic and extensible