# 

NASA SMG Reverse Engineering System Design Specification

|  |  |  |
| --- | --- | --- |
| **Author** | **Revision Number** | **Date** |
| Architect | 1.0 | June 9, 2014 |
|  |  |  |
|  |  |  |

NASA SMG Reverse Engineering System Design Specification 1

System Design Specification 4

1. Introduction 4

1.1 Purpose 4

1.2 Glossary 4

2. Logical Architecture 4

2.1 Frontend Controllers and EJS Templates 5

2.2 Helpers and Utilities 5

2.3 Persistence Data Models (Node.js orm) 5

3. Modular Architecture 5

3.1 Backend Module 5

3.2 Frontend Module 5

4. Data Model 5

5. Process Architecture 6

5.1 Overall Workflow 6

5.2 Authentication 6

5.3 Utilities and Helpers 6

5.3.1 crudHelper 7

5.3.2 helper 7

5.3.3 logging 7

5.3.4 validator 7

5.3.5 dbTypes 8

5.3.6 BadRequestError 8

5.3.7 IllegalArgumentError 8

5.3.8 NotFoundError 8

5.3.9 configuration.js 8

5.3.10 routes.js 8

5.3.11 app.js 8

5.3.12 config.js 8

5.3.13 db.js 8

5.3.14 generateData.js 9

5.3.15 generateFrontendData.js 9

5.3.16 resetDb.js 9

5.4 SQL Scripts 9

5.5 Controllers 9

5.5.1 Characteristic 9

5.5.2 CharacteristicType 9

5.5.3 Dashboard 9

5.5.4 HelpTopic 10

5.5.5 SearchForm 10

5.5.6 SMG 10

5.6 EJS Templates 10

5.6.1 /views/common/breadcrumb.ejs 10

5.6.2 /views/common/footer.ejs 11

5.6.3 /views/common/head.ejs 11

5.6.4 /views/common/header.ejs 11

5.6.5 /views/common/koTemplates.ejs 11

5.6.6 /views/common/menu.ejs 11

5.6.7 /views/common/menuAdmin.ejs 11

5.6.8 /views/admin/addEditForm.ejs 11

5.6.9 /views/admin/addEditSmg.ejs 11

5.6.10 /views/admin/addOrEditHelp.ejs 11

5.6.11 /views/admin/characteristics.ejs 11

5.6.12 /views/admin/forms.ejs 11

5.6.13 /views/admin/help.ejs 11

5.6.14 /views/admin/smg.ejs 11

5.6.15 /views/admin/smgDetails.ejs 11

5.6.16 /views/example.ejs 11

5.6.17 /views/examples.ejs 11

5.6.18 /views/help.ejs 11

5.6.19 /views/helpDetails.ejs 11

5.6.20 /views/home.ejs 11

5.6.21 /views/layout.ejs 11

5.6.22 /views/prelogin.ejs 12

5.6.23 /views/smg.ejs 12

5.6.24 /views/smgCompare.ejs 12

5.6.25 /views/smgDetails.ejs 12

5.7 Postman Collection Export 12

6. Security 12

6.1 Authentication & Authorization 12

6.2 Audit 12

6.3 Data Integrity 12

7. Deployment and Development Concerns 12

7.1 Logical Deployment 12

7.2 Physical Deployment 12

7.2.1 Software 12

7.2.2 Database 13

7.2.3 Application Server 13

7.2.4 Operation System 13

7.2.5 Technical Overview 13

7.2.6 User Interface Implementation 13

7.3 Concurrency and Transaction Management 13

7.4 Configuration 13

7.4.1 Routing Configurations 13

7.4.2 Application Configurations 13

7.5 Logging 14

7.6 Error Handling 14

# System Design Specification

# Introduction

## Purpose

NASA need to develop a software version of the Solution Mechanism Guide so that users can navigate a search form to a filtered subset of solution mechanisms that are a best fit for their specific needs based on the criteria they input (experience similar to finding a car rental or airline ticket online but more fun and NASA-y).

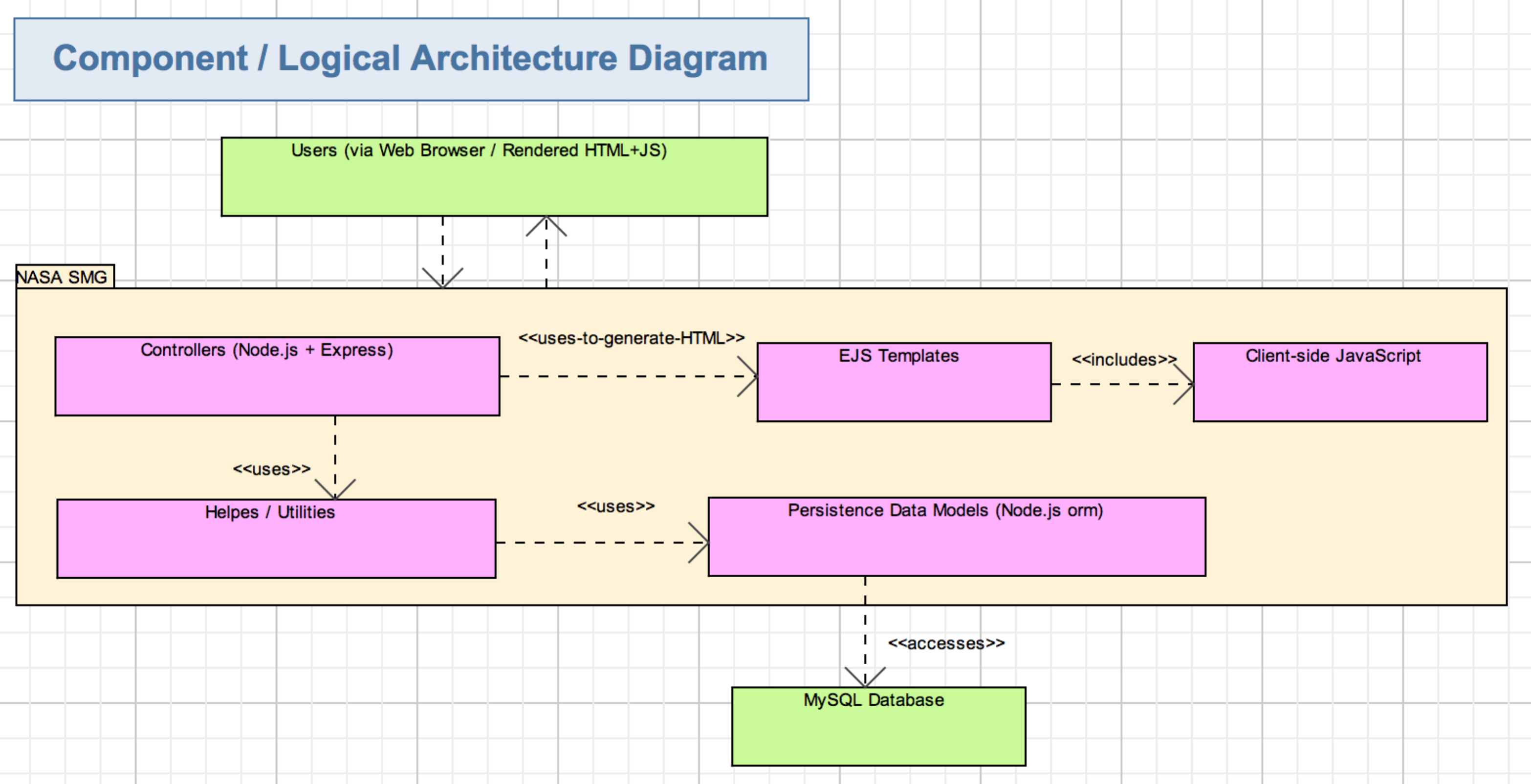
The purpose of this system architecture is to reverse engineer the SMG (Solution Mechanism Guide) application and produce architecture documentations.

## Glossary

| **Acronym, statement** | **Description** |
| --- | --- |
| SMG | Solution Mechanism Guide |
| API | Application Programming Interface |
| REST, REST-ful | Representational State Transfer, an architectural style consisting of a coordinated set of constraints applied to components, connectors, and data elements, within a distributed hypermedia system. |
| ORM | Object-Relational Mapping, a programming technique for converting data between incompatible type systems in object-oriented programming languages. |
| JavaScript | A popular programming language used widely in web applications. |
| Node.js | Node.js is a software platform for scalable server-side and networking applications written in JavaScript. |

# Logical Architecture

SMG application is written in JavaScript, based on [Node.js](http://nodejs.org/) and [Express framework](http://expressjs.com). It uses MySQL database for persistence. The overall logical architecture is depicted as follows.



From the perspective of logical architecture, this application is a simple 3-tiered system:

* Frontend Controllers and [EJS Templates](http://embeddedjs.com)
* Helpers and Utilities
* Persistence Data Models (Node.js orm)

## Frontend Controllers and EJS Templates

The frontend controllers are Node.js Express controllers that handle API requests or page requests and respond with appropriate handling results. They generally define a handler function per route, and the handler function has access to the Node.js Express Request and Response objects.

Frontend controllers will be called by Node.js Express server when an API / page request is received via HTTP interface - Express server determines the right controller and handler function to call based on the pre-defined routes. Please refer to [Express documentation](http://expressjs.com/guide.html) for more information on Express framework and how it works.

Frontend controllers make extensive use of Helpers and Utilities to perform business logic and access persistence.

The Frontend controllers are organized under /controllers directory.

Views are implemented as [EJS templates](http://embeddedjs.com).

## Helpers and Utilities

Helpers and Utilities are Node.js modules that are implemented to encapsulate business logic and database access.

Helpers and Utilities access the database persistence via Node.js [orm](https://npmjs.org/package/orm) data models.

Most Helpers and Utilities are organized directly under /helpers directory.

Some utilities are organized directly under the directory root / (mainly application entry point and database preloading utilities).

## Persistence Data Models (Node.js orm)

This is the persistence layer of this application, which is based on [Node.js orm module](https://npmjs.org/package/orm).

This layer includes a set of Node.js [orm](https://npmjs.org/package/orm) data models, which define the data structures (properties, structure, restrictions) that need to be persisted, and relationships among the data structures.

Please refer to [Node.js orm site](https://github.com/dresende/node-orm2) for information of defining orm data models.

Persistence Data Models are organized under /models directory.

# Modular Architecture

Logically this application can be separated to a backend module and frontend module.

## Backend Module

The backend module includes the helpers, utilities and persistence, this module implements the common business logic.

## Frontend Module

The module includes the Node.js controllers, EJS templates and other web artifacts such as CSS and client-side JavaScript. This module depends on the utilities, helpers and persistence in Backend Module.

# Data Model

The data model is provided in the TCUML “Model Class Diagram”. All data models are Node.js [orm](https://npmjs.org/package/orm) model definitions.

Please refer to [Node.js orm site](https://github.com/dresende/node-orm2) for information of defining orm data models.

Please refer to TCUML "Model Class Diagram" for details of property definitions and relationships between data models.

* **CharacteristicType**

This data model represents a characteristic type.

* **Characteristic**

This data model represents a characteristic.

* **SearchFormField**

This data model represents a search form field.

* **SearchForm**

This data model represents a search form.

* **SMGCharacteristic**

This data model represents an SMG characteristic.

* **CharacteristicTypeValue**

This data model represents a characteristic value type. Note that the data model should be really named "CharacteristicValueType", its current name is likely to be an accidental mistake.

* **Dashboard**

This data model represents dashboard.

* **SearchFormFieldValue**

This data model represents a search form field value.

* **SMG**

This data model represents an SMG.

* **Example**

This data model represents an example.

* **HelpTopic**

This data model represents a help topic.

* **FileUpload**

This data model represents a file upload.

# Process Architecture

## Overall Workflow

Once started, Node.js Express server will listen for incoming API / page requests over HTTP interface.

When an API / page request is received, Node.js Express server determines the controller and function that handle the request as per the pre-configured routing information (configured in /config/routes.js for REST APIs, or /app.js for pages), then dispatches the request to corresponding controller function for further handling.

Controller function extracts parameters from the request, handles the request and responds with appropriate response. During the request handling, controller function may make use of Utilities and Helpers to perform business logic or access persistence.

For API requests, Express framework renders the result as JSON.

For page requests, Express framework renders the result using EJS templates.

## Authentication

SMG application will be deployed in IIS and use [Windows Authentication](http://www.iis.net/configreference/system.webserver/security/authentication/windowsauthentication) for user authentication. The authenticated username will be extracted from "x-iisnode-logon\_user" HTTP header in app.js for each request.

## Utilities and Helpers

This section briefly introduces the utilities and helpers in the application, please refer to relevant script for detailed implementations.

### crudHelper

Script: /helpers/crudHelper.js

This is a helper module that encapsulates CRUD (create, retrieve, update, delete) operations for persistence data.

* Create single entity or entities in batch (createSingleOrBatch)
* Update entities in batch (updateBatch)
* Remove entities in batch (removeBatch)
* Update single entity (updateSingle)
* Filter entities (filter)
* Get single entity (getSingle)
* Remove single entity (removeSingle)

### helper

Script: /helpers/helper.js

This is a helper module that provides various miscellaneous helper functions.

* Get single entity without validation (getSingle)
* Check if exist any entity that match given criteria (checkExists)
* Check if all elements of the array contain a unique value for the sort field (checkUnique)
* Check if array contains duplicated ids (checkUniqueIds)
* Get a function delegate that populate results and call original callback (getPopulateDelegate)
* Populate the SMG entity (populateSMG)
* Populate the CharacteristicType entity (populateCT)
* Populate the Characteristic entity (populateCharacteristic)
* Populate the SMGCharacteristic entity (populateSMGCharacteristic)
* Populate the SearchForm entity (populateSearchForm)
* Populate the SearchFormField entity (populateSearchFormField)
* Populate the SearchFormFieldValue entity (populateSearchFormFieldValue)
* Populate the HelpTopic entity (populateHelpTopic)
* Check if Characteristic can be created (checkCanCreateCharacteristic)
* Check if Characteristic can be updated (checkCanUpdateCharacteristic)
* Check if Characteristic can be deleted (checkCanDeleteCharacteristic)
* Check if CharacteristicType is references by other tables and return the error if can't be deleted (checkCanDeleteCT)
* Check if HelpTopic can be created (checkCanCreateHelpTopic)
* Check if HelpTopic can be updated (checkCanUpdateHelpTopic)

### logging

Script: /helpers/logging.js

This is a helper module that provides logging functions.

* Create a delegate for the express action to perform logging (wrapExpress)
* Handle error and return as JSON to the response (handleError)

### validator

Script: /helpers/validator.js

This is a helper module that provides validation function.

* Validate an object (validate)

### dbTypes

Script: /helpers/dbTypes.js

This is a helper module that defines data types representing node-orm data types.

* Required string (string)
* Optional string (optional\_string)
* Required integer (int)
* Required boolean (bool)

### BadRequestError

Script: /errors/BadRequestError.js

This is the error that indicates bad request.

### IllegalArgumentError

Script: /errors/IllegalArgumentError.js

This is the error that indicates illegal argument.

### NotFoundError

Script: /errors/NotFoundError.js

This is the error that indicates an entity cannot be found.

### configuration.js

Script: /config/configuration.js

This script defines the application-level configurations.

* Database configurations
* Reset database tables flag
* Download directory
* REST API routes (by importing routes.js)

### routes.js

Script: /config/routes.js

This script defines the routes for REST APIs.

### app.js

Script: /app.js

This is the application entry point.

This script does the following:

* initialize the application server, database, Node.js, Express server and routes.
* start he [Node.js](http://nodejs.org/) and Express server to enter event-loop of receiving API / page requests via HTTP interface and dispatching the requests.

Express server dispatches received requests to appropriate controllers according to the pre-defined routes. Controllers then handle the requests and respond with appropriate handling results.

### config.js

Script: /config.js

This script initializes Express server.

### db.js

Script: /db.js

This script initializes database tables.

### generateData.js

Script: /generateData.js

This script recreates all tables and inserts test data.

### generateFrontendData.js

Script: /generateFrontendData.js

This script recreates all tables and inserts test data for frontend application.

### resetDb.js

Script: /generateData.js

This script recreates all tables and leaves all database tables empty.

## SQL Scripts

The SQL scripts for this application are organized under /sql directory.

## Controllers

This section briefly introduces the controllers in the application, please refer to relevant script (as well as relevant sequence diagrams) for detailed implementations.

### Characteristic

Script: /controllers/Characteristic.js

This is the Node.js Express controller that handles operations related to Characteristic:

* List (index)
* Retrieve (show)
* Create single (createSingle)
* Create batch (createBatch)
* Update single (updateSingle)
* Update batch (updateBatch)
* Remove single (removeSingle)
* Remove batch (removeBatch)

### CharacteristicType

Script: /controllers/CharacteristicType.js

This is the Node.js Express controller that handles operations related to Characteristic Type:

* List (index)
* Retrieve (show)
* Create single (createSingle)
* Create batch (createBatch)
* Update single (updateSingle)
* Update batch (updateBatch)
* Remove single (removeSingle)
* Remove batch (removeBatch)

### Dashboard

Script: /controllers/Dashboard.js

This is the Node.js Express controller that handles operations related to dashboard:

* Get dashboard (getDashboard)
* Update dashboard (update)

### HelpTopic

Script: /controllers/HelpTopic.js

This is the Node.js Express controller that handles operations related to help topics:

* List (index)
* Retrieve (show)
* Create single (createSingle)
* Update single (updateSingle)
* Remove single (removeSingle)
* Remove batch (removeBatch)
* Download (download)
* Upload file (uploadFile)

### SearchForm

Script: /controllers/SearchForm.js

This is the Node.js Express controller that handles operations related to search forms:

* List (index)
* Retrieve (show)
* Create single (createSingle)
* Create batch (createBatch)
* Update single (updateSingle)
* Update batch (updateBatch)
* Remove single (removeSingle)
* Remove batch (removeBatch)

### SMG

Script: /controllers/SMG.js

This is the Node.js Express controller that handles operations related to SMG:

* List (index)
* Retrieve (show)
* Create single (createSingle)
* Create batch (createBatch)
* Update single (updateSingle)
* Update batch (updateBatch)
* Remove single (removeSingle)
* Remove batch (removeBatch)
* Search via POST request (search)
* Search via GET request (search2)

## EJS Templates

This section briefly introduces the frontend views (EJS templates), please refer to relevant EJS templates for detailed implementations.

The EJS templates are organized under /views directory.

### /views/common/breadcrumb.ejs

This template renders breadcrumb of pages.

### /views/common/footer.ejs

This template renders footer of pages.

### /views/common/head.ejs

This template renders the HTML <head> of pages.

### /views/common/header.ejs

This template renders header of pages.

### /views/common/koTemplates.ejs

This template renders the Knockout templates.

### /views/common/menu.ejs

This template renders the menu.

### /views/common/menuAdmin.ejs

This template renders the admin menu.

### /views/admin/addEditForm.ejs

This template renders the form to add or edit search form.

### /views/admin/addEditSmg.ejs

This template renders the form to add or edit SMG.

### /views/admin/addOrEditHelp.ejs

This template renders the form to add or edit help topic.

### /views/admin/characteristics.ejs

This template renders the page to manage characteristics.

### /views/admin/forms.ejs

This template renders the page to manage search forms.

### /views/admin/help.ejs

This template renders the page to manage help topics.

### /views/admin/smg.ejs

This template renders the page to manage SMG's.

### /views/admin/smgDetails.ejs

This template renders the page to show SMG details.

### /views/example.ejs

This template renders the page to show example.

### /views/examples.ejs

This template renders the page to show examples.

### /views/help.ejs

This template renders the page to show help topics.

### /views/helpDetails.ejs

This template renders the page to show help topic details.

### /views/home.ejs

This template renders the page to show homepage.

### /views/layout.ejs

This template defines the page layout.

### /views/prelogin.ejs

This template renders pre-login content.

### /views/smg.ejs

This template renders the page to show SMG's.

### /views/smgCompare.ejs

This template renders the page to compare two SMG's.

### /views/smgDetails.ejs

This template renders the page to show SMG details.

## Postman Collection Export

File: postman.json

The postman.json can be imported to [Postman](https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm?hl=en) and can be used to test the REST APIs.

# Security

## Authentication & Authorization

SMG application will be deployed in IIS and use [Windows Authentication](http://www.iis.net/configreference/system.webserver/security/authentication/windowsauthentication) for user authentication. The authenticated username will be extracted from "x-iisnode-logon\_user" HTTP header in app.js for each request.

## Audit

There is no auditing done in this application.

## Data Integrity

Node.js [orm](https://npmjs.org/package/orm) data models are defined with well-defined constraints (e.g. whether a column is required, unique, etc.).

[transaction plugin](http://dresende.github.io/node-orm-transaction/) is used with the Node.js [orm](https://npmjs.org/package/orm) module to support transactions.

# Deployment and Development Concerns

## Logical Deployment

This application is deployed as a Node.js application that serves both frontend page requests and REST API requests. It is deployed in IIS with [IISNode plugin](https://github.com/tjanczuk/iisnode).

## Physical Deployment

### Software

* Node.js
* 3rd-party Node.js modules
  + express 3.4.8
  + rox 0.0.1
  + async 0.2.10
  + winston 0.7.2
  + underscore 1.6.0
  + orm 2.1.3
  + mysql 2.1.0
  + orm-transaction 0.0.2
  + ejs 1.0.0
  + moniker 0.1.2
  + lorem 0.4.0

### Database

* MySQL

### Application Server

* IIS

### Operation System

Any operation system that supports Node.js.

Windows Server is used since IIS is the hosting web server.

### Technical Overview

* Node.js
* JavaScript
* MySQL

### User Interface Implementation

Frontend views are implemented as EJS templates.

## Concurrency and Transaction Management

Applications built with Node.js and JavaScript is inherently asynchronous, callbacks(JavaScript) are used extensively to coordinate asynchronous tasks/operations between controller, helper and persistence layers. There's generally no thread safety concern with Node.js based applications.

[transaction plugin](http://dresende.github.io/node-orm-transaction/) is used with the Node.js [orm](https://npmjs.org/package/orm) module to support transactions.

## Configuration

### Routing Configurations

HTTP routing configurations are defined in

* /config/routes.js for REST APIs
* /app.js for frontend pages

### Application Configurations

All application configurations are defined in dedicated /config/configuration.js script, each configuration item is literally a JavaScript object.

The configurations are organized hierarchically, for example all database related configurations are configured under "database" configuration item.

The following summarizes the configurations, please refer to /config/configuration.js for detailed information.

* database
  + host: the database host
  + database: the database name
  + protocol: the database access protocol
  + port: the database port
  + user: the database user
  + password: the database login password
  + query: the query options
* reset\_tables: whether the database tables should be reset at startup time
* downloadsDir: the download directory
* routes: the routes

## Logging

The application logs activities and exceptions using Node.js [winston](https://npmjs.org/package/winston) module. logging.js module provides wrapper function that automatically performs logging for Express controller.

## Error Handling

In each controller function, if there’s any error occurs, it calls the “next(error)” to inform the Node.js Express server to handle the error.

In Utilities and Helpers, and persistence data models, errors are passed back to function caller via "error" parameter in the callback function.