***Knexaa***

Knowledge. Exam. Anytime. Anywhere.

Design Guide V 0.1

Author: Suman Barick

Revision Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rev.No** | **Date** | **Revised By** | **Reviewed By** | **Reason / Comment** |
| 1 | 28 Oct 2017 | Suman Barick | Suman Barick | First draft |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[1 Configurations and Properties 4](#_Toc512100946)

[1.1 Property angular.\_9 4](#_Toc512100947)

[2 Common Services 5](#_Toc512100948)

[2.1 Secret Service 5](#_Toc512100949)

[2.1.1 Public Functions of Secret Service 6](#_Toc512100950)

[3 Home Module 7](#_Toc512100951)

[4 Library Module 7](#_Toc512100952)

[4.1 Library Index.JSON 8](#_Toc512100953)

[5 Topic Module 10](#_Toc512100954)

[5.1 Topic Index.JSON 10](#_Toc512100955)

[6 Data Definitions / Presentation / Structures 12](#_Toc512100956)

[6.1 MCQ Questions 12](#_Toc512100957)

[6.1.1 Formulas for Encrypted-Answer-Key 12](#_Toc512100958)

[7 Node Tools 17](#_Toc512100959)

# Configurations and Properties

## Property angular.\_9

To prevent population of global / window scope, all the application wide properties are exposed under angular.\_9, since angular is already available with framework.

**angular.\_9 properties**

|  |  |
| --- | --- |
| **name** | **description** |
| hostType | “web” for website.  “app” for mobile apps |
| webRoot | Optional. The domain where data resides when accessed from web. Defaults to $window.location.origin |
| appRoot | Mandatory if hostType=app. Online server domain from where the mobile app will fetch the data. |
| dataRoot | The data origin after webRoot / appRoot. Both will be concatenated to get the data origin.  dataRoot = “data” [for production]  dataRoot = “devData” [for development] |
| xlib | ‘x’ for exam. Path to exam libraries after dataRoot. If a raw path is given, it must be prefixed with “raw:”, else preferred way will be giving an encoded path, which will be deciphered using secretService.decipher. The library name is basically the lowercased string of the original library name. Usage described in section 4. |
| config | complete paths to any other config files, generally inside configs folder |
| uniKey | The universal key to decipher all AES encoded strings allover the app |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Common Services

## Secret Service

This service makes CryptoJS a part of global angular variable. Hence, CryptoJS global library is no longer available in global window object, rather it can be accessible via angular.CryptoJS reference.

### Public Functions of Secret Service

It provides public functions

* decipher
* encrypt
* getAnswer

#### decipher

#### encrypt

**calling pattern:**

secretService.decipher (encodedText);

secretService.encrypt (sourceText);

decipher function deciphers/decrypts the encoded string using CryptoJS AES decryption. encrypt does the opposite of decipher/ decrypt.

The universal decryption key (uniKey) is read from property angular.\_9 .

**uniKey:**

it is basically a disguised form of a long random predefined string of ASCII characters and of length 30. Here is my recipe of creating the uniKey from a random string.

**recipe to create uniKey**

i. Create a hard to guess long string of random characters.

e.g. "\_S6h1K!N8c$7FkM-W#92yTix$@X9sZ".

ii. Loop through each character and add a random multiple of 256 (since total number of ASCII characters is 256) to the characters ASCII code, thus creating the pseudo\_ascii number. The ASCII code of the original character can be retrieved anytime by pseudo\_ascii % 256.

iii. Finally the pseudo\_ascii for each character is concatenated together separated by hyphen (-) to form the “uniKey”. And it becomes something like,

7744863-14135635-13322038-2015080-16136753-19833163-12514593-17757518-4579128-20503651-7431460-6726711-16517190-5287019-14555213-6265901-17583703-4939299-14636857-8574514-19331705-23233876-11785833-4673144-13136164-10791488-16075864-24241465-11007347-23276122

#### getAnswer

Since the entire Knexaa app is primarily designed to be a static app (i.e. front-end only, no backend code), the answer to each question is hidden in the question json itself in a field called “key”. Find below the recipe to create answer\_key from original answer.

**recipe to create answer\_key**

# Home Module

# Library Module

This is to show the exam libraries (e.g. Code, Engineering etc) which are the root of various topics (e.g. C, C#, Java) or libraries (ECE, CSE etc) inside them. So, a library can contain other libraries or topics.

**Quick Analogy**

A **library** is like a **folder** (i.e can contain other libraries)

A **topic** is like a **file** (i.e. cannot contain other topics or libraries)

The route pattern is,

**/library/:libName**

The routeParam libName will contain the library name or index for the root of all libraries. The data mappings are as follows,

**/library/index** => Read the index.json under the main library folder

**/library/libName** => Read the index.json under that libName library folder

The mappings of exam-library name i.e. “libName” to “Actual path” are read from property angular.\_9.xlib [see 1.1 for details].

The work of each library page is mainly dictated by its index.json file.

## Library Index.JSON

Each library folder must have an index.json which lists all the sub libraries or topics it contains.

The **basic structure of index.json** looks like,

{

"groupLibName": "index",

"groupSubjectType": "library",

"list":

[

{

"name": "Coding"

},

{

"name": "Engineering"

},

{

"name": "Aptitude & Reasoning",

"subjectType": "topic"

}

]

}

**i. groupLibName =>** The name of the current library whose index.json is this one.

**ii. groupSubjectType =>** This is the 'subjectType’ of all those list entries where 'subjectType’ field is not explicitly mentioned. This field can be ‘library’ or ‘topic’ depending on what the majority of the children of this library is. If a library is mainly comprised of other libraries and has relatively few numbers / or ZERO topics, then there is no meaning of adding subject type to each entry in the list.

**iii. list =>** This is an array of library or topic jsons where each json contains

a. name: name of that particular library or topic

b. subjectType: [‘library’ / ‘topic’] needed only if differs from groupSubjectType

# Topic Module

This shows an exam topic page which contains various exams available under that topic. Topics are the most elementary units under library and *cannot contain subtopics*.

## Topic Index.JSON

The content of this page is dictated from the index.json file under this topic folder.

The **basic structure of a topic index.json** is as follows,

{

"examCodeExt": "c",

"mighty": true,

"availableExams": [

{

“name”: “C for Beginners”,

“examCode”: “c-11”,

“totalQ”: 100,

“Time”: 02:30:00,

“NoOfQToPass”: 80

},

{

“name”: “C for Intermediates”,

“examCode”: “c-12”,

“totalQ”: 100,

“Time”: 02:30:00,

“NoOfQToPass”: 80

},

{

“name”: “C for Experts”,

“examCode”: “c-13”,

“totalQ”: 100,

“Time”: 02:30:00,

“NoOfQToPass”: 80

}

]

}

**i. examCodeExt =>** This is the unique topic identifier for each topic and this code is appended to “examCode-” to create the **Exam Code Config** file under config folder for that particular exam. For example, “examCode-c.json” is the code config file for all “C” exams. This file contains the unique exam code for each exam. Exam codes are created as <examCodeExt>-<next\_Available\_Index>. Index 1 to 9 are reserved for Quick Exams.

**ii. mighty =>** Boolean. This field indicates whether a mighty.json file is present or not. The file “mighty.json” is nothing but a huge collection of Questions for each particular topic to enable “Quick Exams”(QE). So, whether QE is available depends on whether mighty.json is available.

**“Quick Exam”(QE)**s are random questions taken from a huge questions database named <examCodeExt>-“mighty.json”. QEs come in following flavors

a. Micro [Question: 10, Time: 00:15:00]

b. Mini [Question: 30, Time: 00:45:00]

c. Standard [Question: 60, Time: 01:30:00]

d. Macro [Question: 80, Time: 02:00:00]

e. Major [Question: 100, Time: 02:30:00]

**iii. availableExams =>** All the available fixed format exams.

# Data Definitions / Presentation / Structures

This section records how data is organized and maintained in Knexaa app.

## MCQ Questions

MCQ Questions are maintained in below json format

{

“q”: <the question text>,

“options”: <An array of Options>

[

“option that will be shown against A”,

“option for B”,

“option for C”,

...

],

“key”: <Encrypted-Answer-Key... See formula below in this doc>

}

### Formulas for Encrypted-Answer-Key

**Formula-1:**

The answer key should look like this,

XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX

Where, X = Alphanumeric character i.e. mixture of [0-9A-Z]

To understand, let’s first split by ‘-’ and give each part a name

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Folmula-ID | alphaString1 | alphaString2 | alphaString3 | alphaString4 | alphaString5 | alphaString6 | alphaString7 | alphaString8 |
| XX | XXXXX | XXXXX | XXXXX | XXXXX | XXXXX | XXXXX | XXXXX | XXXXX |

**Formula ID**

|  |  |  |
| --- | --- | --- |
| Formula ID | Meaning | Get ans from ans-key |
| A1 | MCQ Question with **single**answer | 1. var numString1= convertToNumString(alphaString1);  // similarly do for alphaString2, 3 and 4 to create numString2, 3 and 4  2. convertToNumStringfunction will convert any alpha char in that alphaNumeric string to number, by doing ASCII % 10 to get a number <=9  3. var firstNString = numString1 + numString2;  var secondNString = numString3 + numString4;  4. var alphaString = alphaString5 + alphaString6 + alphaString7 + alphaString8;  5. var ultimateIndex = parseInt(secondNString[firstNString[3]]) + parseInt(secondNString[firstNString[6]]) + parseInt(secondNString[firstNString[9]]);  //the index should not go beyond (alphaString.length-1)  6. ultimateIndex = ultimateIndex % (alphaString.length-1);  7. var ans = alphaString[ultimateIndex]; |
| A2 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 1, 4, 7 |
| A3 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 2, 5, 8 |
| A4 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 3, 4, 9 |
| A5 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 6, 8, 9 |
| A6 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 1, 3, 8 |
| A7 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 2, 7, 8 |
| A8 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 1, 2, 5 |
| A9 | MCQ Question with **single**answer | Same recipe as AAAA1, only on step 5, the fixed integers are 4, 5, 7 |

|  |  |  |
| --- | --- | --- |
| Formula ID | Meaning | Get ans from ans-key |
| B1 | MCQ Question with **Multiple** answers | 1. 4th Index of AlphaString1 will tell noOfCorrectOptions to be chosen  2. var shift = (noOfCorrectOptions <= 4) ? 1 : 0;  3. var alpha15 = alphaString2+ alphaString3 + alphaString4;  var alphaString = alphaString5 + alphaString6 + alphaString7 + alphaString8;  4. var correctAns = [];  var gap = 14 / noOfCorrectOptions;  4. for (var i=0; correctAns.length <noOfCorrectOptions; i= i+gap) {    correctAns.push( alphaString [ alpha15 [i + shift] ] );  shift \*= -1;  }  5. return correctAns; |
| B2 | MCQ Question with **Multiple** answers | Same as B1  Only in point 1. 3rd Index of AlphaString1 will tell noOfCorrectOptions to be chosen |
| B3 | MCQ Question with **Multiple** answers | Same as B1  Only in point 1. 1st Index of AlphaString1 will tell noOfCorrectOptions to be chosen |
| B4 | MCQ Question with **Multiple** answers | Same as B1  Only in point 1. 0th Index of AlphaString1 will tell noOfCorrectOptions to be chosen |
| B5 | MCQ Question with **Multiple** answers | Same as B1  Only in point 1. 2nd Index of AlphaString1 will tell noOfCorrectOptions to be chosen |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Node Tools