**Project Documentation: Code Renewal Services**  
  
**1. Introduction**   Code Renewal Services is a versatile Spring Boot project designed to handle periodic updates of various coding systems, including ICD, CPT, Medicines, Pharmacies, Allergies, and others. The project provides API-based initiations. It aims to streamline the process of reading and processing updated code files, synchronizing the changes with existing coding tables, and ensuring data integrity through verification and analysis.  
  
**2. Workflow Overview**  
   The following steps outline the workflow of the code renewal process:  
  
   Step 1: File Uploading  
   - This method retrieves(csv file) the updated coding system file from the S3 bucket.  
   - It reads the file's content and prepares it for further processing.  
  
   Step 2: File Processing  
   - The File Processing method takes the content of the file obtained in Step 1 and inserts it into a temporary table.  
   - The temporary table holds the updated codes until they are synchronized with the existing coding table.  
  
   Step 3: Analysis  
   - The Analysis method examines the additions, deletions, and updates present in the new file.  
   - It calculates statistics such as the total number of additions, deletions, and updates for reporting purposes.  
  
 Step 4: Verification  
   - Verification cross-checks the newly updated codes in the coding table against the content of the file once again.  
   - It ensures the accuracy and consistency of the updated codes.  
  
Step 5: Synching  
   - The Synching process updates the existing coding table with the changes found in the temporary table.  
   - It ensures that the coding table reflects the latest code updates.

Step 6: Complete  
   - The Complete method provides an overview of the entire code renewal process.  
   - It includes a summary of the changes made, statistics, and any relevant information about the success or failure of the process.  
  
**3. Project Modes**  
   Code Renewal Services offers following operation:  
  
   - **API-based Initiation:**  
     - In this mode, the project exposes APIs that allow external systems to trigger the code renewal process.  
     - External systems can send requests to the specific API endpoints, providing the necessary information to initiate the update for a particular coding system.  
     - The API-based initiation allows for flexibility and integration with other systems.  
  
**4. Abstract Class**: CodeRenewalAbstract  
   - The CodeRenewalAbstract class serves as a blueprint for each coding system's implementation.  
   - It defines the abstract methods that each coding system must implement to support the code renewal process.  
   - The abstract methods include:  
     - fileUploading()  
     - fileProcessing()  
     - analysis()

- verification()  
     - synching()  
     - complete()  
  
**5. Implementation for Each Coding System**  
   - For each coding system (ICD,CPT,Medicines,Allergies,Pharmacies etc.), a separate class should be created that extends the CodeRenewalAbstract class.  
   - Each class should provide the implementation for the abstract methods specific to that coding system.  
   - This allows for a modular and extensible design, enabling the addition of new coding systems in the future.  
  
**6. Conclusion**  
   The Code Renewal Services project offers flexibility and automation for managing code updates in various coding systems. Whether initiated through APIs or operated as a standalone automated process, the project streamlines

API’s List:

--------------

1.File Upload API  
----------------  
User Parameters  
1) Standard/DataSet - ICD10, CPT, SNOMED, Pharmacies, Medicines etc  
2) Release Version  
3) Release Date  
4) ZIP file to be uploadedFile

5)UserId

6)EffectiveFrom

7)EffectiveTo

2.Delete API:  
----------------  
1) For Unporcessed or Failure Files.

3.List API  
----------  
Fields to be given to UI  
1) Standard/Dataset  
2) Release Version  
3) Release Date  
4) FileName  
5) State (Uploaded/In Process/Processed)  
6) Temp Status (File Extraction, File Reading, Additions, Editing, Deleting, Verifying, Completed)  
7) Status (Success/Failure)  
8) Reason/Comments/\* Other Metadata Fields \*/

4.Process API  
------------  
1) Select from list ID (Metadata ID)  
2) Process Button only for Files which are in Uploaded state.----> Trigger your Process in Background and move the State to 'In Process'  
Background ---> 1) Zip File Extraction 2) Reading of Files 3) Analysis 4) Additions 5) Edits 6) Deletes. 7) Create Summary 8) Update Status  
Based on status -> 1) Revert or Resubmit 2) Correct and Update

5.Verify API  
-----------  
User Will verify the data based on Stats. Maintain verification step and results Using Verification Search API.

6.Update Status API  
-------------------------  
1) Once Verification is Done, User will update the Status to Completed. From this step onwards, the data will be visible in User Search API

S3 folder structure:

----------------------

Base Folder  
 |- Upload Folder  
    |-ICD10  
    |-SNOMED  
    |-ETC  
 |- InProcess Folder  
    |-ICD10  
    |-SNOMED  
    |-ETC  
 |- Processed Folder  
     |-ICD10  
        |- Date1  
            |- ICD10 Files  
        |- Date2  
            |- ICD10 Files  
     |- SNOMED  
        |- Date1  
            |- SNOMED Files  
        |- Date2  
            |- SNOMED Files

**Code Maintenance Steps**

**I.File Upload Steps:** **POST:https://devcsapi.atria.healthcare:8080/codemaintenance/fileupload?codeType=icd&releaseVersion=2022&releaseDate=2022-02-09&userId=2&effectiveFrom=2022-02-10&effectiveTo=2023-08-31**

**1.User Input:**

Collect the following information from the user:

Code type

Release version

Release date

User ID

Effective from date

Effective to date

Zip file containing the code

**2.Check Zip File Existence**:

Check if the provided zip file already exists in the S3 bucket.

If the file exists, show a message indicating that the file already exists in S3, and skip the rest of the process.

If the file doesn't exist, proceed to the next step.

**3.File Upload:**

Upload the provided zip file to the designated S3 bucket.

You can use a library or SDK for your programming language to interact with S3 for file upload.

**4.Update Processed Status:**

Once the file is successfully uploaded, update the processed status to 'uploaded' in system's database

Table: **coding\_standard\_files**

**II.File Process Steps:**

**POST:https://devcsapi.atria.healthcare:8080/codemaintenance/fileprocess?fileId=2&userId=2**

**1. Process Zip File:**

- Extract the contents of the uploaded zip file.

- Identify and read the appropriate text file from the extracted contents.

- Create a table based on the data extracted from the text file.

- Compare the newly created table with the existing table data.

- Generate verification data based on the comparison.

- Update the verification table with the generated verification data.

- Set the processed status of the file to 'pending\_verification'.

**2. Logging and Record Keeping:**

- Maintain a log of each step executed in a log table.

- Include timestamps, file IDs, and step descriptions in the log entries.

- Log any errors or exceptions encountered during the process.

**3. Entry Point:**

- Implement an entry point (e.g., a main function) to start the process.

- Specify the file ID for the file to be processed.

- Call the ' Process File' step with the provided file ID.

Tables: **1.coding\_standard\_files**

**2. icd10\_data\_verification**

**III. Retrieve Change Counts After Processing:**

**GET:** **https://devcsapi.atria.healthcare:8080/codemaintenance/changecounts?fileId=1&userId=3**

- The response from the server will contain the counts of records that were added, updated, and deleted during the processing.

- Capture and utilize these counts for further analysis or reporting.

**Table:code\_change\_counts**

**IV.ICD Verification Search Results :**

**GET:** **https://devcsapi.atria.healthcare:8080/codemaintenance/verification/icd?fileId=1&searchTerm&status&pageSize=10&pageNumber=0**

fileId: The ID of the file to which the codes belong.

searchTerm: The search term to filter codes by code and descriptions (short, medium, long). Leave empty if not searching.

status: The status of the codes to filter by(All,Added,Updated,Deleted). Leave empty if not filtering by status.

pageSize: The number of codes to retrieve per page (e.g., 10).

pageNumber: The page number to retrieve (e.g., 0 for the first page).

- The response from the server will contain a list of codes based on the provided search and pagination parameters.

Table: **icd10\_data\_verification**

**V.Verification Log:** **POST:https://devcsapi.atria.healthcare:8080/codemaintenance/codeverification?codeset=icd&fileId=1&userId=2**

-Verifying the codes as per user verified and we are maintaining the verified codes in the verification log table

Table: **code\_verification\_log**

**VI.Synching:POST:** [**https://devcsapi.atria.healthcare:8080/codemaintenance/filesynch?fileId=2&userId=4**](https://devcsapi.atria.healthcare:8080/codemaintenance/filesynch?fileId=2&userId=4)

**-** A**fter verification we need to sync the added,updated,deleted records into the existing table.**

**Table:icdnew**

**VII.Post Sync Search Results:GET:** [**https://devcsapi.atria.healthcare:8080/codemaintenance/postsyncresults/icd?fileId=1&status=Updated&pageSize=10&pageNumber=1**](https://devcsapi.atria.healthcare:8080/codemaintenance/postsyncresults/icd?fileId=1&status=Updated&pageSize=10&pageNumber=1)

After Syncing we have to search the records based on status(Added,Updated,Deleted**)**

**Table Schemas:**

**1.Coding\_standard\_files**

**CREATE TABLE `coding\_standard\_files` (**

**`id` int NOT NULL AUTO\_INCREMENT,**

**`file\_name` varchar(500) DEFAULT NULL,**

**`file\_path` varchar(500) DEFAULT NULL,**

**`code\_standard` varchar(45) DEFAULT NULL,**

**`inserted\_date` timestamp NULL DEFAULT CURRENT\_TIMESTAMP,**

**`user\_id` int DEFAULT NULL,**

**`source` varchar(45) DEFAULT NULL,**

**`processed\_status` varchar(45) DEFAULT NULL,**

**`status` varchar(45) DEFAULT NULL,**

**`release\_date` date DEFAULT NULL,**

**`current\_status` varchar(45) DEFAULT NULL,**

**`release\_version` varchar(45) DEFAULT NULL,**

**`comments` varchar(255) DEFAULT NULL,**

**`modified\_date` timestamp NULL DEFAULT NULL,**

**`modified\_user\_id` int DEFAULT NULL,**

**`active` int DEFAULT '1',**

**`effective\_from` varchar(50) DEFAULT NULL,**

**`effective\_to` varchar(50) DEFAULT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci**

**2.Coding Maintenance Log**

**CREATE TABLE `coding\_maintenance\_log` (**

**`id` int NOT NULL AUTO\_INCREMENT,**

**`file\_id` int DEFAULT NULL,**

**`event\_type` varchar(100) DEFAULT NULL,**

**`event\_desc` varchar(250) DEFAULT NULL,**

**`user\_id` int DEFAULT NULL,**

**`inserted\_date` timestamp NULL DEFAULT CURRENT\_TIMESTAMP,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci**

**3.Code Change Counts**

**CREATE TABLE `code\_change\_counts` (**

**`id` int NOT NULL AUTO\_INCREMENT,**

**`file\_id` int DEFAULT NULL,**

**`added\_records` int DEFAULT NULL,**

**`updated\_records` int DEFAULT NULL,**

**`deleted\_records` varchar(45) DEFAULT NULL,**

**`status` varchar(45) DEFAULT NULL,**

**`user\_id` int DEFAULT NULL,**

**`inserted\_date` timestamp NULL DEFAULT CURRENT\_TIMESTAMP,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci**

4.**Code Verification Log**

CREATE TABLE `code\_verification\_log` (

`id` int NOT NULL AUTO\_INCREMENT,

`codeset` varchar(45) DEFAULT NULL,

`code` varchar(45) DEFAULT NULL,

`file\_id` int DEFAULT NULL,

`user\_id` int DEFAULT NULL,

`inserted\_date` timestamp NULL DEFAULT CURRENT\_TIMESTAMP,

`notes` varchar(500) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

**5.icd10 data verification**

CREATE TABLE `icd10\_data\_verification` (

`id` int NOT NULL AUTO\_INCREMENT,

`icd\_order` varchar(5) DEFAULT NULL,

`icd\_code` varchar(10) DEFAULT NULL,

`icd\_id` int DEFAULT NULL,

`type` varchar(1) DEFAULT NULL,

`short\_desc` varchar(255) DEFAULT NULL,

`medium\_desc` varchar(512) DEFAULT NULL,

`long\_desc` varchar(1024) DEFAULT NULL,

`file\_id` int DEFAULT NULL,

`inserted\_date` timestamp NULL DEFAULT CURRENT\_TIMESTAMP,

`status` varchar(45) DEFAULT NULL,

`verification\_status` tinyint DEFAULT '0',

PRIMARY KEY (`id`),

KEY `idx\_icd\_code` (`icd\_code`)

) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

**6.Coding Asynch task status**

CREATE TABLE `coding\_asynch\_tasks\_status` (

`id` int NOT NULL AUTO\_INCREMENT,

`file\_id` int NOT NULL,

`task\_type` varchar(150) NOT NULL,

`current\_status\_desc` varchar(255) DEFAULT NULL,

`completion\_status` varchar(100) DEFAULT NULL,

`completion\_percent` int NOT NULL,

`user\_id` int NOT NULL,

`inserted\_date` timestamp NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

**7.icd standard versions**

CREATE TABLE `icd\_standard\_versions` (

`id` int NOT NULL AUTO\_INCREMENT,

`icd\_order` varchar(45) DEFAULT NULL,

`icd\_code` varchar(45) DEFAULT NULL,

`icd\_id` int DEFAULT NULL,

`type` char(1) DEFAULT NULL,

`short\_desc` varchar(255) DEFAULT NULL,

`medium\_desc` varchar(255) DEFAULT NULL,

`long\_desc` varchar(512) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `icd\_code` (`icd\_code`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci