- **Material Management System is an ongoing project, where we are backend API developers here. This phase (3rd Phase) is probably the last one. The previous 2 phases was successfully delivered in time and the client is using them.**
- The project will be used for a construction firm. In this business, two parties – Donators and Receivers are involved, based on some matching criteria.
- Matching between Donators and Receivers is accomplished through a complex algorithm, which was implemented in the previous phase. The algorithm considers the four facts: filling purpose (Permanent/Temporary), schedule start & end dates, material volume and project location. These four criteria reveal some score by the algorithm, which are stored in four score fields in the 'matched_results' table.
- The system is developed with NodeJS 14 with MSSQL Database.
- The sheet 'Field Name Explanation' of the attached spreadsheet describes all the required tables' description (that's the standard procedure to document a database)
- *Installation & Execution Guide:*
    - Download 'MaterialManagement.sql ' and import it into your local MSSQL database.
    - Download **SourceRepo** and execute in the command prompt: *npm install.*
    - Update your database configuration in the '.env' file
    - Browse the table 'authorized_users' where to discover some credentials to be used
    - Now execute *npm start*. This will initiate nodemon to execute the project.
    - Hit the browser to visit **http://localhost:3000/docs**. You'll see the Swagger documentation with the developed APIs until now. Open **api/swagger/swagger.yaml** and you'll see all the APIs details specification.
    - Phase 2 has been deployed at https://mmaprojectapi.azurewebsites.net/docs.
    - Now hit **POST /oauth/token** and use the default credential to login. If your database is running properly, you'll be able to get the *access_token*. Copy it, scroll up a bit to get the authorize button on the right. Click it and type 'Bearer ACCESS_TOKEN', then hit *Authorize* button. This process will enable you to access the other APIs. Let me provide you some example data to run some APIs below:
        - You can use the obtained 'used_id' from the login response to run ***GET*** /appUser/{user_id}. FYI, the data will be fetched through the external CLIENT API (old version).
        - The two POST APIs under the tag **Supplementary Document File** are used to upload files (for the table 'supplimentaries') to Azure Blob Storage and using the same file retrieved from Azure server to delete. However, you need to use the new APIs, detailed in the PDF document to replace Azure Blob Storage.
        - The APIs under the tag **Manage & View Requests (Backfill/Disposal)** can be tested as below:
            1. **GET** /projectByDepartment/{department_key}: use the department keys 17810 or 14820 (again data will be fetched through the external CLIENT API).
            2. **POST** /createRequest: change the values of department_key. Project_address (based on the obtained address from the above GET

API), request_type, filling_purpose, material volume, available_volume(should be equal material volume) and schedule dates:

- {
- "user_id": "KNC01",
- "department_key": "17810",
- "project_address": "Kai Tak, Kowloon City District, Hong Kong",
- "request_type": "Backfill",
- "filling_purpose": "Temporary",
- "material_type": "Soil",
- "material_quality": "",
- "material_volume": 850,
- "material_unit": "m3",
- "matched_volume": 0,
- "available_volume": 850,
- "schedule_start_date": "2021-09-10",
- "schedule_end_date": "2021-09-30",
- "schedule_status": "Confirmed",
- "remarks": "Remarks from KNC01",
- "supplementary_document_names": "",
- "supplementary_documents": "",
- "contact_names": "",
- "contact_phones": ""
- }

3. **GET** /listUserRequests/{user_id}/{request_type} and /listGlobalRequests/{user_id}/{request_type}: user 'WLW01' for user_id
4. **GET** /requestDetails/{request_id}/{user_id}: pick some values from database
5. **PUT** /updateRequest: You can use the similar object as that of **createRequest** with 'id' on top
6. **DELETE** /deleteRequest/{request_id}/{user_id}/{department_key}: I'LL RECOMMEND NOT TO EXECUTE IT, BECAUSE SOME IMPORTANT DATA MIGHT BE LOST FOR YOUR DEVELOPMENT

- The APIs under the tag **Manage & View Backfill/Disposal Requests with Matches** can be tested as below (important to understand your action points):
  1. **GET** /listUserRequestsWithMatches/{user_id}/{request_type}: use 'HWC04' for user_id (do you see how matched data come per a specific owner request?).
  2. **POST** /confirmMatches: by observing the information as well as the sample data in the database, if you can understand that how to confirm a match (Our Side Confirm, Other Side Confirm, Both Confirmed), then you can perform some matches (an example given below)
     - {
     - "user_id": " WLW01 ",

        - " own_request_id ": 4094,
        - "matched_requests": [
        - 4088, 4090
        -]
        - }
    3. **PUT** /manageDocuments: When a match is confirmed by the both sides, then it is required submit some documents. When system detects that a match is confirmed by both sides, then it automatically inserts some rows in the 'documents' table (search data by the matched_result_id=10058). The field 'compulsory_document' can hold multiple file-names, separating by coma(,). For the simplicity of the development, the POST APIs under the tag **Supplementary Document File** are used for this purpose, where documents are first uploaded to fetch the file-names, to be used in this PUT API.
    4. **GET** /matchDetails/{user_id}/{own_request_id}/{matched_request_id}: use data 'HWC04', '4091' and '4085' for **User ID**, **Own Request ID** and **Matched Request ID** respectively.
    5. **POST** /cancelMatch: I'LL RECOMMEND NOT TO EXECUTE IT, BECAUSE SOME IMPORTANT DATA MIGHT BE LOST FOR YOUR DEVELOPMENT

- Before diving into the actual requirements, some basic need to be discussed:
    - The field 'departmen_key' under the table 'requests' is sometimes also treated as project-id or project-code throughout the phases. As an example, In the attached PDF file, you'll see the images which have IDs like **A2368, B5563.**
    - From the table *requests:*
        - request_type(1): Donator (Disposal)
        - request_type(2): Receiver(Backfill)
        - filling_purpose(1): Permanent
        - filling_purpose(2): Temporary
    - The project is all about *creating* and *matching* **disposal/backfill** requests:
        - Basic data is saved into 'requests' table
        - A request's other information, like supplementary documents, contact people and material data (action point) are saved into 'supplementaries', 'contacts' and 'materials' tables respectively.
        - When a **disposal** request is created by a donator, then the system attempts to find some matched **backfill** requests. Again, when a **backfill** request is created by a receiver, then the system attempts to find some matched **disposals**.
        - The request, for which the system attempts to find matches, is called ***own request***. And the matches are called ***matched requests***
        - The table 'matched_results' has the foreign-keys 'own_request_id' and 'matched_request_id', referenced with 'id' field of 'requests' table. So, for example, 'own_request_id' for the request of **4091** can have matchings like **4085, 4087, 4092, 4093** within the field 'matched_request_id'.

        - The table ‘matched_results’ has another important field ‘status’ with four possible values as below:
            1. **0 ->** Neither owner nor the matched parties confirm the match
            2. **1 ->** Owner confirmed only
            3. **2 ->** Matched party confirmed only
            4. **3 ->** both owner & matched side confirmed a matched
        - There another important table, which is ‘matched_summaries’. It is connected with the ‘matched_results’ table with the foreign-key ‘matched_result_id’. By the table-name, it is understandable that it stores the special values, for which the system made a match for a particular request.
    - Other details may be provided if needed …
- In the attached spreadsheet, you’ll see the sheet ‘(Old) Client API - Project & Us’, which is the reference of **Client API**. This was used to fetch information like a single project-details, project-involved people. If you open the **.env** file, you’ll find the API URL is set under the key ‘CLIENT_API_ROOT’. Your first task will be to modify the CLIENT API root based on the sheet ‘(New) Client API - Project & Us’.
- With the information above, please proceed to implement the required APIs for **Part 1 & 2.**

**** Deadline:**

- **The APIs of Part 1 need to be accomplished on or before 28th Nov’21 ☺**
- **The APIs of Part 1 need to be accomplished by 3rd Dec’21.**