



# Snap Script Implementation Details

## Team Selector

Mentor: Yogesh Gangwar

Intern : Arya Wankhede

Buddy: Pragati Yadav



# Project Introduction

**Snap-Script** – A utility for UL Countries (US and Canada) to automate missing product system images script generation.

Today, missing product system images require manual database scripting, which is time-consuming and error-prone.

Snap-Script provides developers a quick glance at missing images through an intuitive UI and allows them to generate the necessary database scripts with a single click — saving time and boosting productivity.

## **Goal/Deliverables :**

- Allow developers to view all missing product system images for UL countries with a single click
- Automate the generation of database scripts for missing images through a single-click action

# Business Impact

The solution will be integrated into the Firestop Selector's in-house tools ecosystem, optimizing internal workflows.

By shortening development time, it will directly contribute to accelerating the time-to-market for related products and updates.



# Pre-requisite:

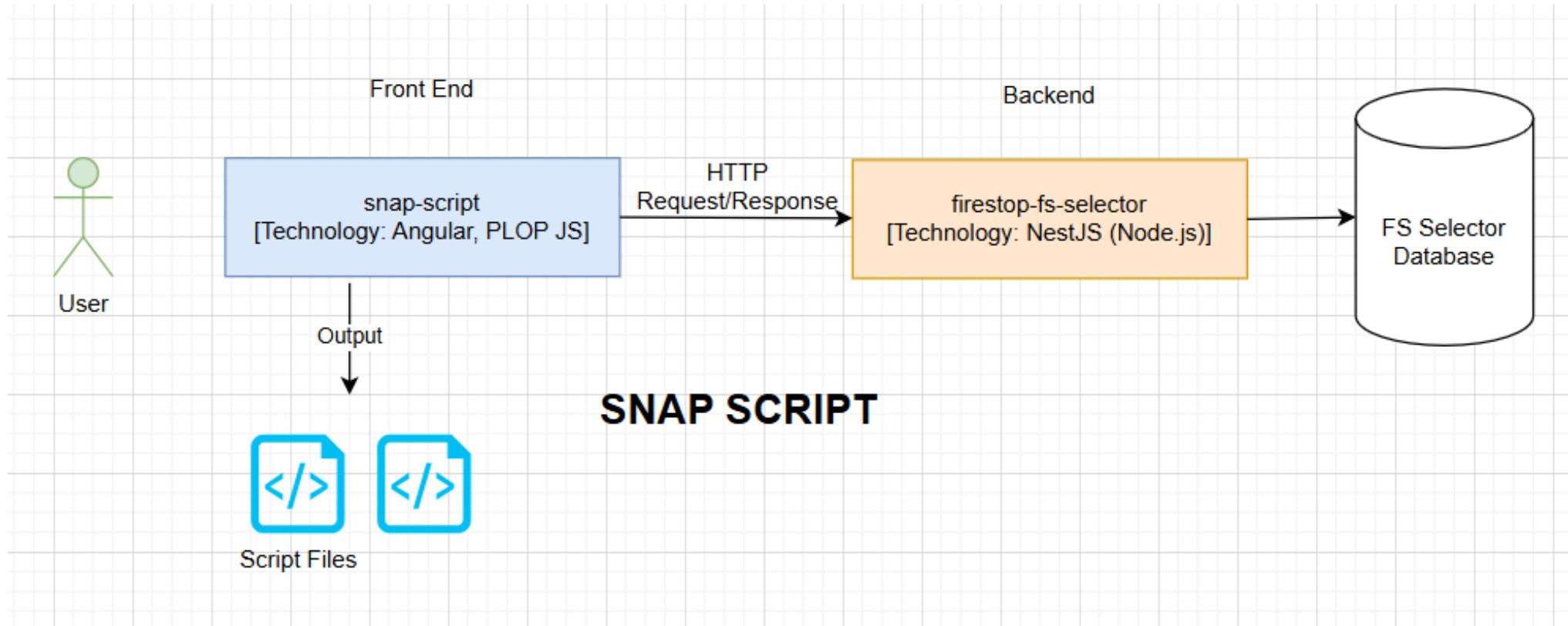
## **Technology Stack:**

Angular, Nest js, Plop, VS Code IDE , Jest Framework, Jira , GitLab, Postgres database, Docker

## **Existing ETL Process**

<https://hilti.atlassian.net/wiki/spaces/BFS/pages/1704100926/ETL+Run+--+Quick+Steps>


# Implementation Approach



# Implementation Approach Backend

## Frontend UI:

Create new standalone **snap-script** angular project. And create components to provide UX as shown in below mock-up.

 SnapScript

Country

All

▼

Fetch Data

MISSING IMAGES TYPICALS

<input checked="" type="checkbox"/>	Select all	S.No.	Typical Name	Typical ID	Country
<input checked="" type="checkbox"/>		1	W-J-1245	3000	US
<input checked="" type="checkbox"/>		2	F-C-3034	3001	US
<input checked="" type="checkbox"/>		3	C-AJ-2123	3000	US
<input checked="" type="checkbox"/>		4	W-L-1192	3002	Canada

Generate Scripts

# Implementation Approach Backend

## Frontend UI Functionality:

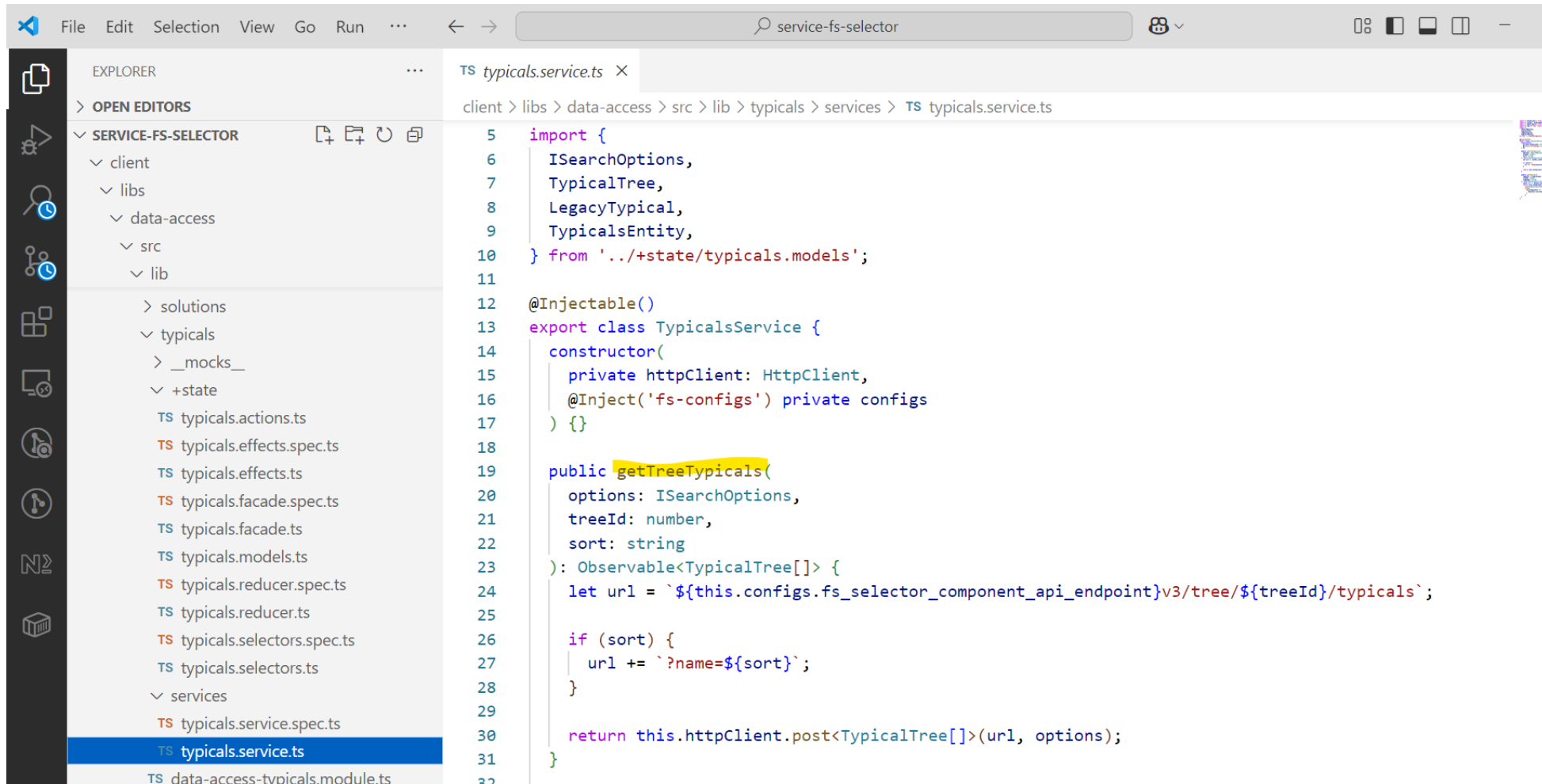
1. The UI **should have a dropdown** to select the country (US, Canada, or All) and a button to fetch data for the selected option.
2. It **should display document data** in a grid with the following columns: **S.No., Typical Name, Typical ID, Market ID, and Type**.
3. The user **should be able to select all documents or specific ones** based on their requirement.
4. After selection, the user **should be able to click on the "Generate Scripts" button** to generate the corresponding database script.
5. Data base script should generate in individual file for each selection.
6. Application should retain the state of Documents for which script generation successful.



# Implementation Approach Backend

## Frontend UI:

Create new service in **snap-script** angular project by taking reference of **TypicalsService** in **service-fs-selector**. Also use ngrx store to persist the data and execute actions on store.



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left and the Editor window on the right. The Explorer sidebar shows the project structure for 'SERVICE-FS-SELECTOR', with the file 'typicals.service.ts' selected under the 'services' directory. The Editor window displays the code for 'typicals.service.ts'.

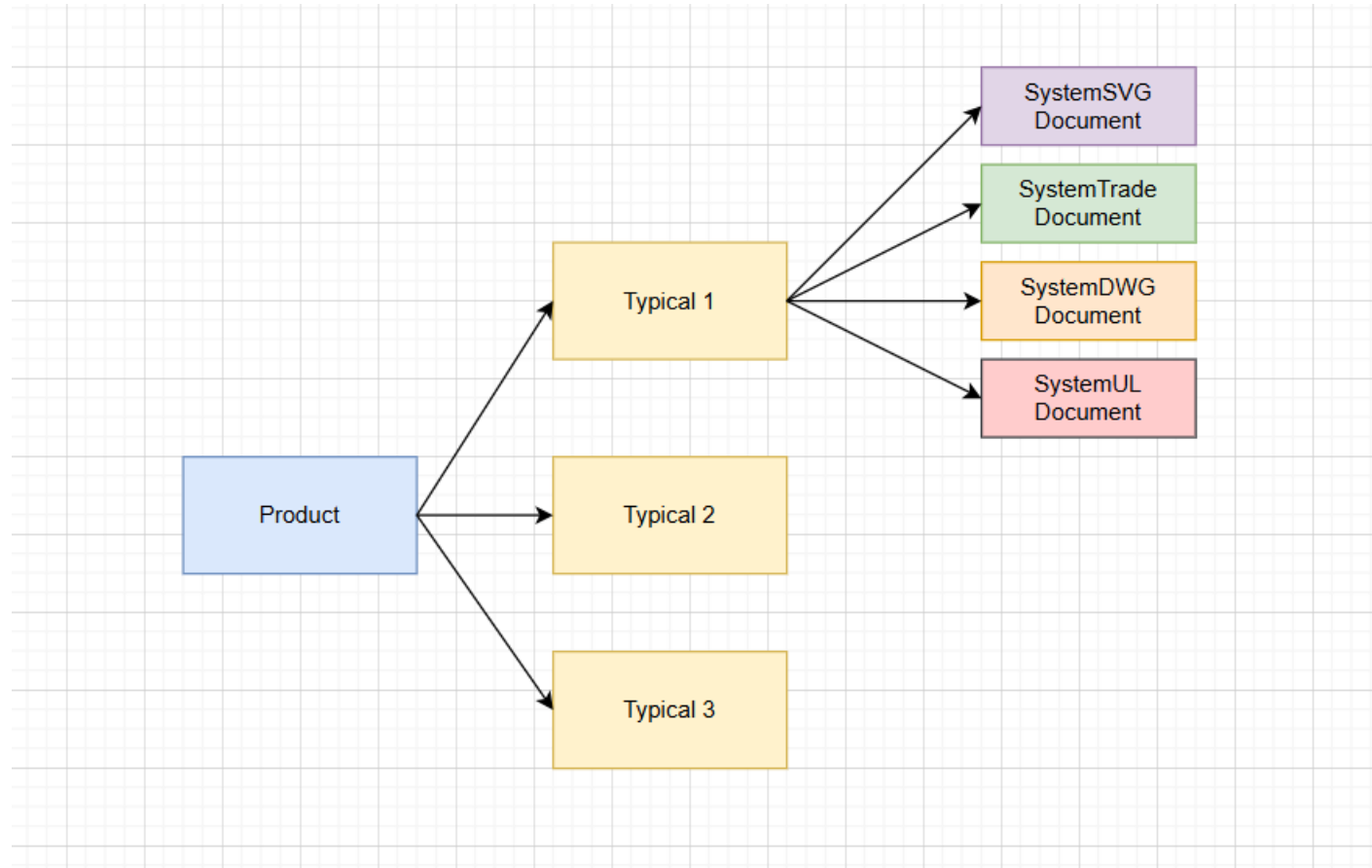
```
5 import {
6   ISearchOptions,
7   TypicalTree,
8   LegacyTypical,
9   TypicalsEntity,
10 } from '../+state/typicals.models';
11
12 @Injectable()
13 export class TypicalsService {
14   constructor(
15     private httpClient: HttpClient,
16     @Inject('fs-configs') private configs
17   ) {}
18
19   public getTreeTypicals(
20     options: ISearchOptions,
21     treeId: number,
22     sort: string
23   ): Observable<TypicalTree[]> {
24     let url = `${this.configs.fs_selector_component_api_endpoint}v3/tree/${treeId}/typicals`;
25
26     if (sort) {
27       url += `?name=${sort}`;
28     }
29
30     return this.httpClient.post<TypicalTree[]>(url, options);
31   }
32 }
```



# Implementation Approach

## Documents Linkage:

This is how multiple types Documents are linked to Typical, which finally linked to Product



# Implementation Approach Backend

## Backend Database:

**STEP 1]** Open **Typicals** table in FS-Selector database in PostgreSQL. Get **Id's** from **Typicals** table for given **MarketId**

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane shows the database structure, with 'Typicals' table selected under 'Tables (12)'. The 'Query Editor' pane shows the following SQL query:

```
1 SELECT "Id", "Name", "ApprovalNo", "ApprovalSystem", "MarketId", "ExternalId", "IsActive"
2 FROM public."Typicals" where "MarketId" = '2';
```

The 'Data Output' pane displays the results of the query in a table format:

	Id [PK] integer	Name character varying (255)	ApprovalNo character varying (100)	ApprovalSystem character varying (5)	MarketId integer	ExternalId character varying	IsActive boolean
1	3000	W-J-1245	W-J-1245	UL	2	PRD_SYSTEM_2982296	true
2	3001	C-AJ-4054	C-AJ-4054	UL	2	PRD_SYSTEM_2285997	true
3	3002	F-C-3074	F-C-3074	UL	2	PRD_SYSTEM_2286240	true
4	3003	W-J-8048	W-J-8048	UL	2	PRD_SYSTEM_2286412	true
5	3004	W-L-3335	W-L-3335	UL	2	PRD_SYSTEM_2286548	true
6	3005	C-AJ-3318	C-AJ-3318	UL	2	PRD_SYSTEM_2285991	true
7	3006	W-L-2509	W-L-2509	UL	2	PRD_SYSTEM_2286510	true
8	3007	HW-D-1104	HW-D-1104	UL	2	PRD_SYSTEM_2713628	true
9	3008	W-L-5345	W-L-5345	UL	2	PRD_SYSTEM_8847742	true
10	3009	HI/BPF 120-18	HI/BPF 120-18	UL	2	PRD_SYSTEM_10932879	true

# Implementation Approach Backend

## Backend Database:

**STEP 2]** Open **Typical\_Documents** table.

Get **DocumentId**'s from table for given **TypicalId**

The screenshot shows the pgAdmin 4 web interface. On the left, the 'Browser' pane displays a tree view of database objects. The 'Typical\_Documents' table is selected under the 'Tables (12)' category. The main pane shows the 'Query Editor' with a SQL query: `SELECT * FROM public."Typical_Documents" where "TypicalId" = 3000;`. Below the query editor, the 'Data Output' tab is active, displaying a table with 4 rows and 4 columns: Id [PK] integer, TypicalId integer, and DocumentId integer. The results show four rows of data where TypicalId is 3000 and DocumentId values are 22704, 22705, 22706, and 22707.

	Id [PK] integer	TypicalId integer	DocumentId integer
1	23294	3000	22704
2	23295	3000	22705
3	23296	3000	22706
4	23297	3000	22707

# Implementation Approach Backend

## Backend Database:

**STEP 3]** Open **Documents** table.

Get **DocumentId**'s from table for given **TypicalId**

The screenshot shows the pgAdmin 4 web interface. On the left, the 'Browser' pane shows the database structure, with 'Typical\_Documents' selected under 'Tables (12)'. The 'Query Editor' pane shows a SQL query with two lines: 'SELECT \* FROM public."Typical\_Documents" where "TypicalId" = 3000;' and 'SELECT \* FROM public."Documents" where "Id" in (22704, 22705, 22706, 22707);'. The 'Data Output' pane shows the results of the query, which are the rows from the 'Documents' table where 'Id' is in the specified list. The results are displayed in a table with columns: Id, ExternalId, Type, Name, and ExternalUrl.

Id	ExternalId	Type	Name	ExternalUrl
1	22704	SystemSVG	WJ1245c.svg	hna/hna/WJ1245c.svg
2	22705	SystemTrade	(Trade) W-J-1245	https://productdata.hilti.com/APQ_HC_RAW/ASSET_DOC_LOC_2982312.pdf
3	22706	SystemDWG	(DWG) W-J-1245	https://productdata.hilti.com/APQ_HC_RAW/ASSET_DOC_LOC_3059656.dwg
4	22707	SystemUL	(UL) W-J-1245	https://productdata.hilti.com/APQ_HC_RAW/ASSET_DOC_LOC_3059660.pdf

# Implementation Approach Backend

## Backend Database:

**STEP 4]** In **Documents** table.

For given **Id**'s collection, get record for which **Type** is 'SystemSVG'

**NOTE: If no records exist, it means we need to generate script for this.**

The screenshot shows the pgAdmin 4 web interface. On the left, the 'Browser' pane displays a tree view of database objects. Under 'Tables (12)', the 'Typical\_Documents' table is selected. The 'Columns (3)' section shows 'Id', 'TypicalId', and 'DocumentId'. The main pane shows the 'Query Editor' with the following SQL query:

```
1 SELECT * FROM public."Typical_Documents" where "TypicalId" = 3000;  
2 SELECT * FROM public."Documents" where "Id" in (22704, 22705, 22706, 22707) and "Type" = 'SystemSVG';  
3  
4
```

Below the query editor, the 'Data Output' tab is active, displaying a table with the following data:

	<b>Id</b> [PK] integer	<b>ExternalId</b> uuid	<b>Type</b> character varying (50)	<b>Name</b> character varying (500)	<b>ExternalUrl</b> text
1	22704	3b7d117a-66a2-4d37-9ef4-f5aa73ded74d	SystemSVG	WJ1245c.svg	hna/hna/WJ1245c.svg

# Implementation Approach Backend

## Backend Database:

Write query to fetch documents data where **ExternalUrl** is null or empty based on input **MarketId**

**NOTE:** For UL region, **Type** value should be 'SystemSVG'

```
SELECT "Id", "Name", "ApprovalNo", "ApprovalSystem", "MarketId", "ExternalId",  
"IsActive" FROM public."Typicals" where "Id"= '4950';
```

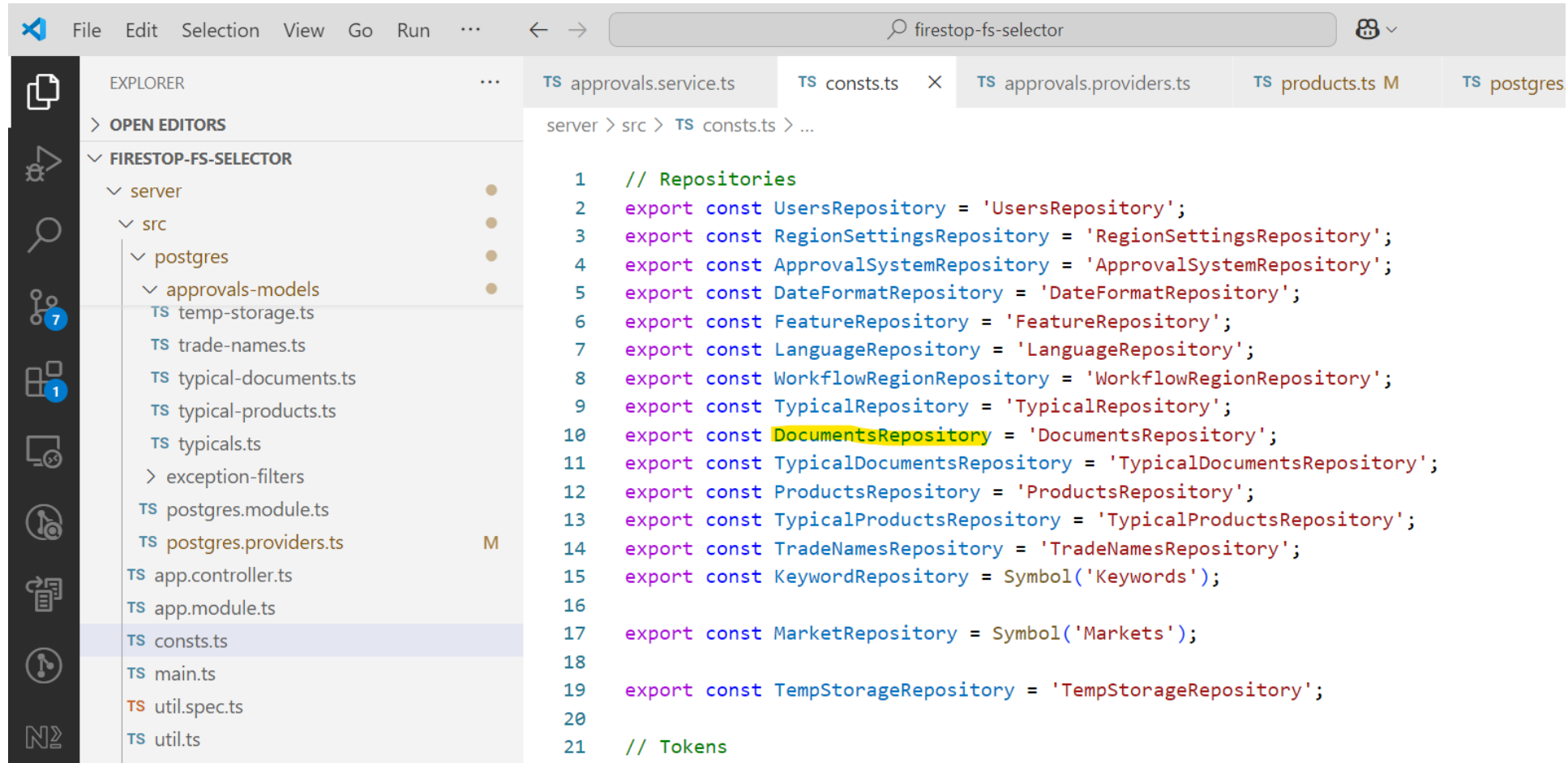
```
SELECT * FROM public."Typical_Documents" where "TypicalId"= '4950';
```

```
SELECT * FROM public."Documents" where "Id" in (31741,31742, 31743, 31744,  
34803);
```

# Implementation Approach Backend

## Backend Service:

Add service in **firestop-fs-selector** project using **DocumentsRepository**. Take reference of this repository and add new service **DocumentsService** and **DocumentsController**.



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the project structure for 'FIRESTOP-FS-SELECTOR'. The 'server' directory contains a 'src' folder, which in turn contains a 'postgres' folder. Inside 'postgres', there is an 'approvals-models' folder containing several TypeScript files, including 'TS const.ts' which is currently selected. The main editor area shows the content of 'TS const.ts', which lists various repository exports. The 'DocumentsRepository' entry on line 10 is highlighted in yellow.

```
1 // Repositories
2 export const UsersRepository = 'UsersRepository';
3 export const RegionSettingsRepository = 'RegionSettingsRepository';
4 export const ApprovalSystemRepository = 'ApprovalSystemRepository';
5 export const DateFormatRepository = 'DateFormatRepository';
6 export const FeatureRepository = 'FeatureRepository';
7 export const LanguageRepository = 'LanguageRepository';
8 export const WorkflowRegionRepository = 'WorkflowRegionRepository';
9 export const TypicalRepository = 'TypicalRepository';
10 export const DocumentsRepository = 'DocumentsRepository';
11 export const TypicalDocumentsRepository = 'TypicalDocumentsRepository';
12 export const ProductsRepository = 'ProductsRepository';
13 export const TypicalProductsRepository = 'TypicalProductsRepository';
14 export const TradeNamesRepository = 'TradeNamesRepository';
15 export const KeywordRepository = Symbol('Keywords');
16
17 export const MarketRepository = Symbol('Markets');
18
19 export const TempStorageRepository = 'TempStorageRepository';
20
21 // Tokens
```



# Implementation Approach Backend

## Backend Service:

Add service in **firestop-fs-selector** project using **DocumentsRepository**. Take reference of this repository and add new service **DocumentsService** and **DocumentsController**.

**Endpoint:** ``${this.configs.fs_selector_api_endpoint}documents/${treeId}/missingurl`;`

## Documents/:treeId

```
@Controller('documents/:treeId')
@UseInterceptors(HcpTrackingInterceptor)
export class DocumentsController {
    constructor(private readonly documentsService: DocumentsService) {}

    @ApiBearerAuth()
    @Get('/missingurl')
    @UsePipes(ValidationPipe)
    @UseFilters(new ValidationExceptionFilter(), new ForeignKeyConstraintExceptionFilter())
    public async getMissingUrlDocuments(
        @Param('treeId', ConvertTreeIdToMarketIdPipe) marketId: number) {
        return this.documentsService.getMissingUrlDocuments(marketId);
    }
}
```

# Implementation Approach Backend

## Backend Service:

Add service in **firestop-fs-selector** project

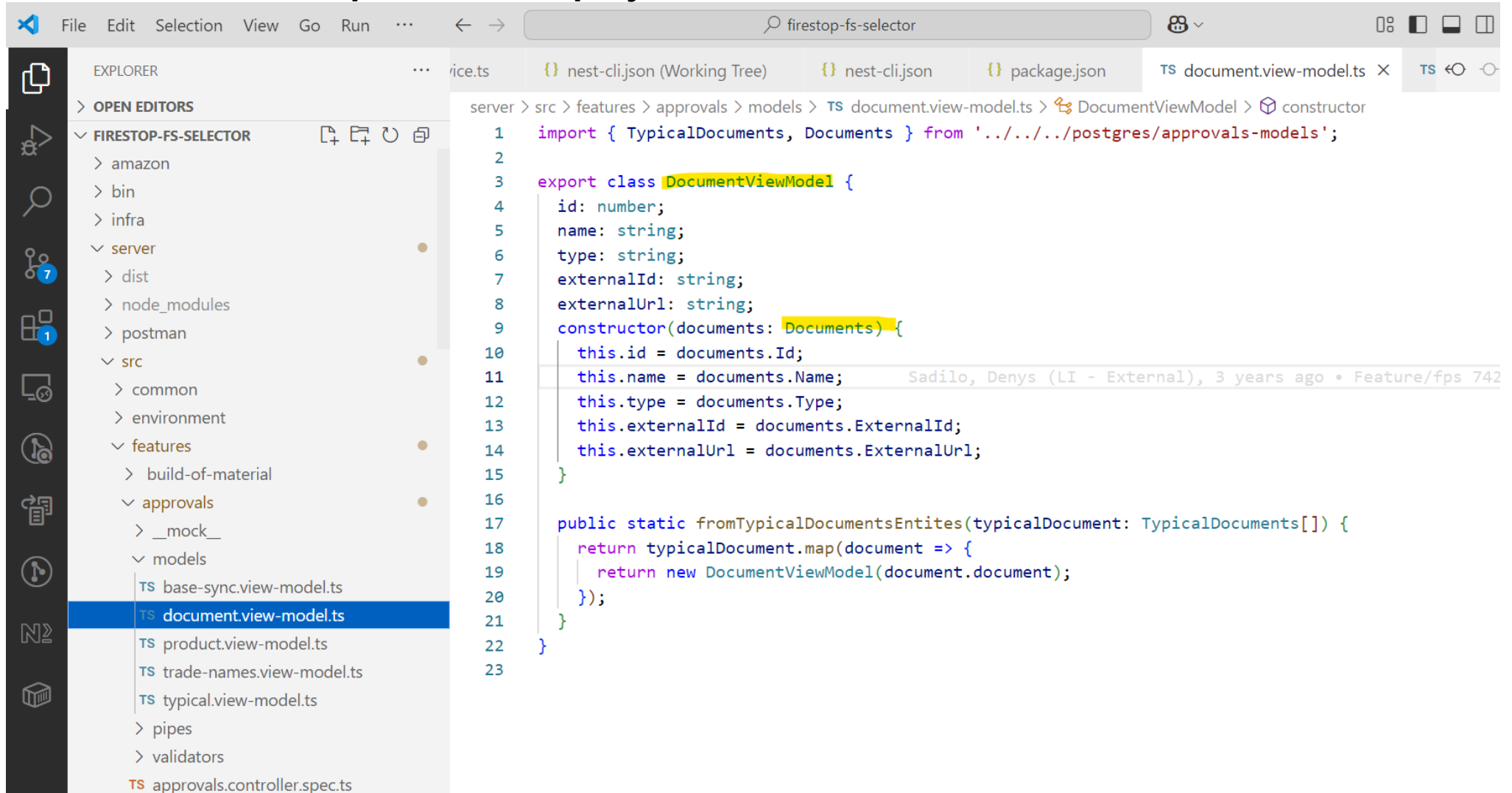
The screenshot shows the Visual Studio Code interface with the **firestop-fs-selector** project open. The Explorer sidebar on the left shows the project structure, with the **approvals-models** folder expanded and **documents.ts** selected. The main editor displays the TypeScript code for **documents.ts**, which defines a **Documents** class extending **Model** from **sequelize-typescript**. The code includes imports for database-related types and defines several database columns and relationships.

```
server > src > postgres > approvals-models > TS documents.ts > Documents
1  import { Column, DataType, Model, Table, PrimaryKey, BelongsTo, ForeignKey } from 'sequelize-typescript';
2  import { Markets } from '../markets';
3
4  @Table
5  export class Documents extends Model<Documents> {
6    @PrimaryKey
7    @Column(DataType.INTEGER)
8    Id: number;
9
10   @Column(DataType.STRING(500))
11   Name: string;
12
13   @Column(DataType.STRING(50))
14   Type: string;
15
16   @Column(DataType.UUID)
17   ExternalId: string;
18
19   @Column(DataType.TEXT)
20   ExternalUrl: string;
21
22   @Column(DataType.TEXT)
23   AssetLang: string;
24
25   @ForeignKey(() => Markets)
26   @Column(DataType.INTEGER)
27   MarketId: number;
28
29   @BelongsTo(() => Markets)
30   Market: Markets;
```

# Implementation Approach Backend

## Backend Service:

Add service in **firestop-fs-selector** project



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure for 'firestop-fs-selector'. The 'server' folder is expanded, showing subfolders like 'dist', 'node\_modules', 'postman', 'src', 'common', 'environment', 'features', 'approvals', and 'models'. The 'models' folder is further expanded, listing several TypeScript files, with 'document.view-model.ts' selected. The main editor area displays the content of 'document.view-model.ts'. The file path in the breadcrumb is 'server > src > features > approvals > models > TS document.view-model.ts > DocumentViewModel > constructor'. The code defines a class 'DocumentViewModel' with properties 'id', 'name', 'type', 'externalId', and 'externalUrl'. It includes a constructor that takes a 'Documents' object and assigns its properties to the instance. Additionally, there is a static method 'fromTypicalDocumentsEntites' that takes an array of 'TypicalDocuments' and returns an array of 'DocumentViewModel' objects.

```
1  import { TypicalDocuments, Documents } from '../../../postgres/approvals-models';
2
3  export class DocumentViewModel {
4      id: number;
5      name: string;
6      type: string;
7      externalId: string;
8      externalUrl: string;
9      constructor(documents: Documents) {
10         this.id = documents.Id;
11         this.name = documents.Name;
12         this.type = documents.Type;
13         this.externalId = documents.ExternalId;
14         this.externalUrl = documents.ExternalUrl;
15     }
16
17     public static fromTypicalDocumentsEntites(typicalDocument: TypicalDocuments[]) {
18         return typicalDocument.map(document => {
19             return new DocumentViewModel(document.document);
20         });
21     }
22 }
23
```

# Implementation Approach Output Consumption

## Backend Service:

Below sql script should be generated via Snap Script UI that will act as input for firestop-fs-selector

### In Documents table:

**Id** should be **max id + 1**

**ExternalId** should be **guid**

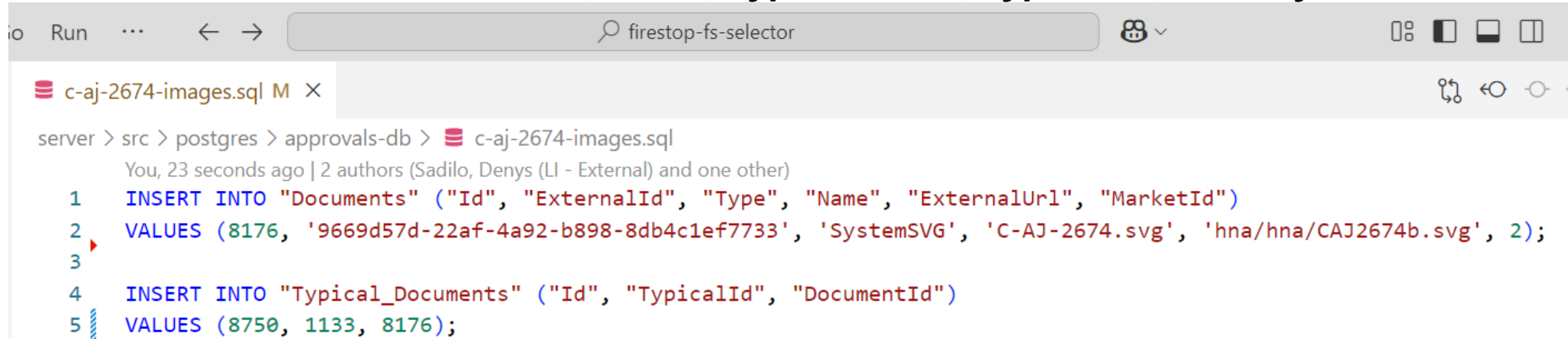
**Type** should be **'SystemSVG'**

**Name** should be **Typical name**

**ExternalUrl** should be **Typical name** without **-** and additional **b.svg** suffix

### In Typical Documents table

**Id** should be **max id + 1** **Select All, S.No., Typical Name, Typical ID, Country**



```
server > src > postgres > approvals-db > c-aj-2674-images.sql
You, 23 seconds ago | 2 authors (Sadilo, Denys (LI - External) and one other)
1  INSERT INTO "Documents" ("Id", "ExternalId", "Type", "Name", "ExternalUrl", "MarketId")
2  VALUES (8176, '9669d57d-22af-4a92-b898-8db4c1ef7733', 'SystemSVG', 'C-AJ-2674.svg', 'hna/hna/CAJ2674b.svg', 2);
3
4  INSERT INTO "Typical_Documents" ("Id", "TypicalId", "DocumentId")
5  VALUES (8750, 1133, 8176);
```

# Thank you

**YOGESH GANGWAR**

Hilti Aktiengesellschaft  
Feldkircherstrasse 100  
9494 Schaan, Liechtenstein

yogesh.gangwar@hilti.com  
www.hilti.group