Phone Number Detection

1. Introduction

- PhoneNumberDetection is a web application developed in C# using ASP.NET Core framework that allows users to input text containing phone numbers and detects and identifies the phone numbers based on various formats.

2. Architecture Overview

- The application follows a Model-View-Controller (MVC) architectural pattern, where:
- **Model:** Represents the data and business logic of the application. In this application, the **`PhoneNumberModel`** class represents the model for phone number data.
- **View:** Represents the user interface components. Views are implemented using Razor syntax (.cshtml files) for dynamic rendering of HTML content.
- **Controller:** Handles user input, processes requests, and provides responses. Controllers are responsible for invoking the appropriate business logic and selecting the appropriate view to render.

3. Components

3.1 Model

- **PhoneNumberModel:** Represents the data model for phone numbers. Contains properties for input text, converted text, and validation result.

3.2 Controller

- **PhoneNumberController:** Handles user requests related to phone number detection. Implements actions to display the input form and process input data. Utilizes the **`PhoneNumberModel`** for data transfer between views and controllers.

3.3 View

- **Index.cshtml:** The main view for the application. Contains a form for inputting text and displaying detected phone numbers along with their formats and validation results.

4. Dependencies

- **ASP.NET Core:** Web framework for building web applications with C#.
- **xUnit:** Testing framework for unit testing.

- **Razor:** View engine for creating dynamic web pages.

5. Development Environment

- **IDE:** Visual Studio or Visual Studio Code.
- **Framework:** .NET Core SDK.
- **Dependencies Management:** NuGet Package Manager.
- **Version Control:** Git/GitHub.

6. Testing

- **Unit Tests:** Unit tests are implemented using xUnit framework to ensure the correctness of business logic and controller actions.

7. Future Enhancements

- **Improved Validation:** Enhance validation logic to handle more complex phone number formats and edge cases.
- **Internationalization:** Add support for multiple languages and locales.
- **UI/UX Enhancements:** Improve the user interface for better usability and accessibility.

8. Deployment

- **Hosting:** The application can be deployed to any web server capable of hosting ASP.NET Core applications.
- **Continuous Integration/Continuous Deployment (CI/CD):** Implement CI/CD pipelines for automated testing and deployment.

9. Conclusion

- PhoneNumberDetection is a simple yet effective web application for detecting and identifying phone numbers from text input. It provides a robust architecture and extensible design to accommodate future enhancements and scalability.

1. SCPLite DB Setup

1.1 In table "tenant" in the SCPLite DB, update the field "nj_location" with the path of "Nextgen.Job. Exe" in the BAT server.

For. e.g - D:\Xpression-Perf\Xpression.Job\V1\NextGen.Job.exe

update tenant set [nj_location] = 'D:\Xpression-Perf\Xpression.Job\V1\NextGen.Job.exe' where [name] = 'xpression_qa_wine_spirits_njbl'

2. Tenant DB Setup

2.1 In table "enterprise_setting", update key value "NJBLExecutionType"
NJBLExecutionType – 1 (parallel) , 2 (time-based) , 0 (default – non-NJBL)
By default, for a tenant, value will be 0 which is non-NJBL.

For. e.g -

update enterprise_setting set [value] = '1' where [name] = 'NJBLExecutionType'

Tables -

njbl_batch_trigger_data – Contains metadata for batch details for NJBL and map the latest job_id to it.

njbl_batch_job_archive – Contains all the historical data of batch_id-job_id mapping to be traced if required

njbl_location – At start of every NJBL, locations are loaded batchwise into this table

njbl_sourcingnetwork – At start of every NJBL, njbl_location specific SNs are loaded batchwise into this table

event_type – New event types -

1. njbl_start - To trigger NJBLs from XPression.Event application
2. njbl_post_process – To wrap up post NJBL processes -

A) Triggering "IBP Baseline Workflow"
B) Truncating pre_staging tables
C) Truncating njbl_batch_trigger_data table for time-based NJBL type
D) Updating job_id to NULL in njbl_batch_trigger_data table for paralle NJBL type
E) Removing Redis cache for that tenant

Pre-Requistes -

1. NJBLExecutionType – This flag should be set to "1" - parallel OR "2" - time-based for NJBL customers (Default - "0" for non-NJBL)
2. GetNJBLData –This step frequency should be set to "1" for NJBL customers

DevOps Tool Process -

1. Files get uploaded in S3 bucket at location "PreInputFiles"
2. Scheduled task runs which checks existence of trigger files.

Stored Procedures-

1. usp_insert_batch_trigger_data – For time-based NJBL type, his SP is called to insert data into "njbl_batch_trigger_data"
2. usp_add_njbl_event_log – Adds "njbl_start" event for that particular "batch_name" passed from DevOps tool.
3. usp_check_njbl_completed – This SP is called from DevOps task scheduler at the end of all NJBLs (at a particular time). If all NJBLs are completed (successfully, manually stopped or failed), then this SP is called which
4. usp_get_nj_batchid_by_name – Get batch_id from batch_name passed to the NJ.exe
5. usp_get_nj_step_status_by_jobid -
6. usp_get_njbl_status – To load NJBL process details grid on UI
7. usp_njbl_post_process – This is called from XPression.Event for "njbl_post_process" event and this SP does following - Triggering "IBP Baseline Workflow", Truncating pre_staging tables, Truncating njbl_batch_trigger_data table for time-based NJBL type, Updating job_id to NULL in njbl_batch_trigger_data table for paralle NJBL type
8. usp_njByLocation – This SP is called for each NJBL to load data into "njbl_location" and "njbl_sourcingnetwork" based on the "batch_id" passed
9. usp_postnjByLocation –This SP is executed at end of NJBL and deletes the completed SNs from "njbl_sourcingnetwork" and updates locations as "Successful" or "Failed" in table "njbl_location"
10. Rollout SPs – batch_id wise data insert/ update
11. IBP SPs - batch_id wise data insert/ update

Corner Cases:

1. For parallel NJBL Execution type -

A) If a batch does not arrive in S3 for that particular day, then mark that "batch_id" record as "is_inactive" to skip from checking.

B) If batch_id with sequence "1" does not arrive for that day (rarest of rare case), then update subsequent batch sequences accordingly. Means, sequence 2 batch becomes sequence 1 and sequence 3 batch becomes sequence 2 and so on. This has to be updated from backend as we only know if a batch has not arrivedat the last moment (real time) and not before hand.

C) If a batch fails and "Manually Stopped", "Failed", "Completed" status is not inserted into "pricing_job_history" due to SQL connection issue or n/w issue (rarest of rare case), then that entry has to be inserted manually into "pricing_job_history" table

For e.g - If batch_name "E" fails, then execute below code (Replace "E" with required appropriate batch_name)

```
DECLARE @batch_name varchar(10) = 'E'
DECLARE @job_id Varchar(200) = ''
DECLARE @job_seq_id INT = 0

SELECT @job_id = job_id FROM njbl_batch_trigger_data WHERE batch_name = @batch_name
SELECT top 1 @job_seq_id = job_seq_id FROM pricing_job_history WHERE id = @job_id

select @job_id, @job_seq_id

IF (@job_id != '' AND @job_seq_id > 0)
BEGIN
        INSERT INTO pricing_job_history(id, step, date_time, status, job_seq_id)
        VALUES (@job_id,'ManuallyJobStopped', getdate(),'Completed', @job_seq_id)
END
```

Difference between Regular NJ and NJBL:

	Regular	**NJBL**
NJBLExecutionType	0	1 or 2
Manual NJ run command	Nextgen.Job.exe tenant_name	Nextgen.Job.exe tenant_name batch_name
NJ Restart	Yes (manual only – command - "Nextgen.Job.exe tenant_name restart")	Not developed yet
Prestaging tables truncate	In NJ itself	After all NJBLs are completed, "njbl_post_process" event
Dashboard refresh	Refresh event (will get refreshed after 5 min)	Direct call to SP, no event entry
NJ Restart	Yes	No
Period End NJ	Regular	Need to keep NJBLExecutionType = 0 before Nj starts till NJ completes and then update it back to previous entry. It should run as full NJ as per FRD.