### ASP.NET MVC to Angular Migration Using Groq/Gemini and Flask

**1. Introduction**

This document provides a detailed design overview for the migration of legacy ASP.NET MVC applications to an Angular-based architecture. The migration process includes utilizing Python and Flask to dynamically generate metadata for Angular components using the Groq/Gemini API.

**Objective**

To enable the seamless transition of legacy ASP.NET MVC applications by dynamically generating JSON metadata from configuration files and leveraging Angular for dynamic UI rendering.

Prototype/Implementations Details - <https://github.com/kanishkarthik/dotnet-to-angular-migration>

**2. System Architecture**

**2.1 High-Level Architecture**

1. **Flask Application (Python)**:
   * Accepts user inputs (e.g., country code, payment method).
   * Reads .cs configuration files from Asp.net MVC Application (Source App).
   * Interacts with the Groq/Gemini API to generate JSON metadata.
   * Stores metadata in the Angular application’s assets folder.
   * Below are AI & LLM models are used

|  |  |
| --- | --- |
| Groq | Llama 3.1 8B Instant **Llama 3.3 70B Versatile** DeepSeek R1 Distill Llama 70B  Llama 70b 8192 |
| Groq Ingest (RAG) | Llama 3.1 8B Instant **Llama 3.3 70B Versatile** DeepSeek R1 Distill Llama 70B  Llama 70b 8192 |
| Gemini | **Gemini 1.5 Pro**  **Gemini 2.0 Flash**  Gemini 2.0 Flash-Lite Preview |

1. **Groq/Gemini API**:
   * Processes .cs file content.
   * Converts configurations into JSON metadata for dynamic Angular UI rendering.
2. **Angular Application (Target App)**:
   * Consumes JSON metadata to dynamically render forms and UI components.

**2.2 Component Diagram**

[User]

|

v

[Flask App]

|-- Read Configurations (.cs files)

|-- Interact with Groq/Gemini API --> [Groq/Gemini API]

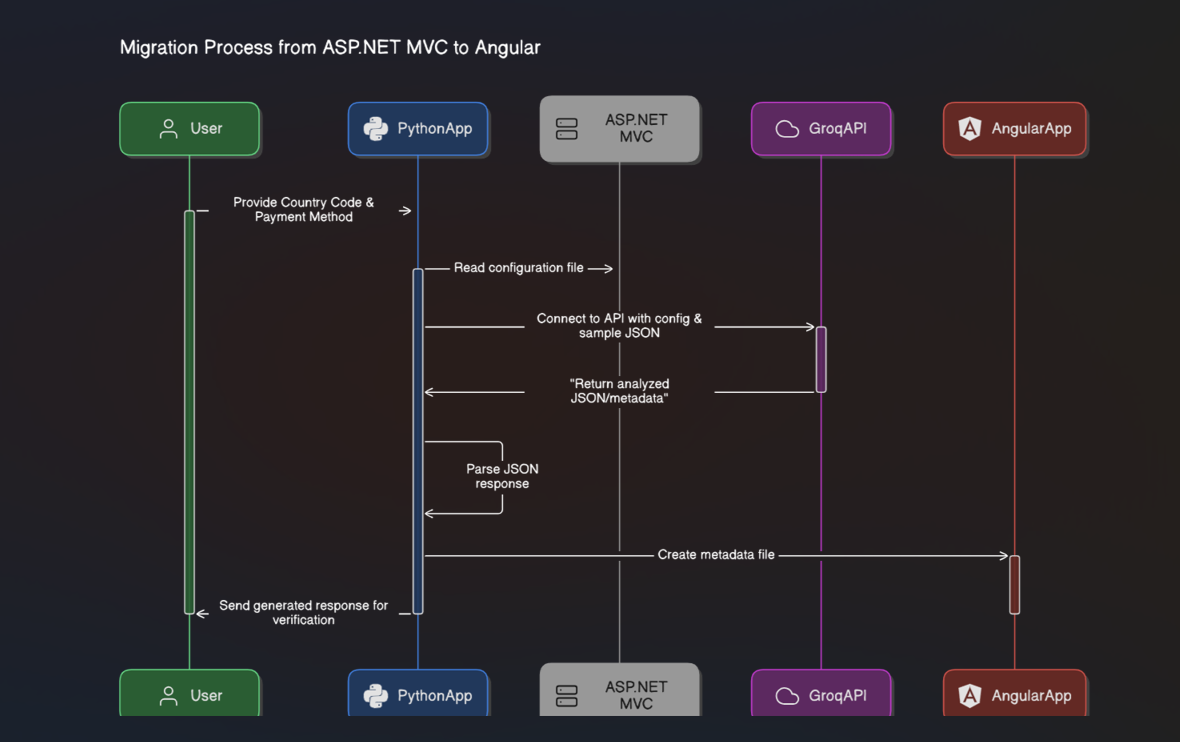
|-- Store JSON Metadata --> [Angular Metadata Folder]

v

[Angular App]

--> Render Dynamic UI

**3. Detailed Design**

****

**3.1 Python(Flask) Application**

**Key Functionalities**

1. **Input Handling**:
   * Accepts country\_code and payment\_method from the user.
   * Validates inputs to ensure required fields are present.
2. **Configuration Mapping**:
   * Reads configurations.json to locate the corresponding .cs file based on the user input.
   * Groq Ingest – Scans the entire ASP.NET MVC codebase, indexes relevant configuration files using vector embeddings, and applies transformations to generate structured JSON metadata for Angular UI rendering.
3. **Metadata Generation**:
   * Reads the .cs file content.
   * Sends the content to the Groq/Gemini API for analysis and metadata generation.
   * Parses the Groq/Gemini API’s response and validates the JSON.
4. **Metadata Storage**:
   * Writes the generated JSON metadata to the AngularApp/src/assets/metadata directory with a filename based on country\_code\_payment\_method.json.

**Flask Routes**

| **Route** | **Description** |
| --- | --- |
| / | Displays a form for user input. |
| /generate-metadata | Processes inputs, generates metadata, and stores it. |

**3.2 Groq/Gemini API Integration**

**Purpose**

* Converts ASP.NET MVC .cs configuration files into JSON metadata suitable for Angular.

**Metadata Format**

{

  "formId": "US\_BKT",

  "country": "US",

  "paymentMethod": "BKT",

  "sections": [

    {

      "id": "paymentDetails",

      "name": "Payment Details",

      "fields": [

        {

          "id": "tranRefNo",

          "name": "Transaction Reference",

          "type": "textbox",

          "required": true,

          "pattern": "[A-Z0-9]+",

          "maxLength": 10,

          "validation": {

            "messages": {

              "required": "Transaction reference is required",

              "pattern": "Only uppercase letters and numbers allowed"

            }

          }

        }

      ]

    }

  ]

}

**3.3 Angular Application**

**Dynamic UI Rendering**

1. **Metadata Consumption**:
   * Reads JSON metadata from the assets/metadata folder.
   * Dynamically generates forms, fields, and UI components based on the metadata.
2. **Integration**:
   * Metadata determines the layout, field types, and validation rules for the UI.

**Advantages**

* Decouples UI from hardcoded configurations.
* Allows easy updates for new fields or configurations.

**4. Workflow**

**4.1 End-to-End Flow**

1. **Input Stage**:
   * User enters country\_code and payment\_method in the Flask app.
2. **Configuration File Processing**:
   * Flask locates the corresponding .cs file.
   * Reads the file content and sends it to the Groq API.
3. **Metadata Generation**:
   * Groq/Gemini API analyses the .cs file and generates JSON metadata.
4. **Metadata Storage**:
   * Flask writes the JSON metadata to the Angular app’s assets folder.
5. **Dynamic Rendering**:
   * Angular reads the metadata and dynamically renders the UI.

**5. Error Handling**

**5.1 Flask Application**

* **Invalid Inputs**:
  + Returns error messages if country\_code or payment\_method is missing.
* **Missing Configuration Files**:
  + Raises an error if the .cs file is not found.
* **Groq/Gemini API Errors**:
  + Logs and returns errors from the Groq/Gemini API (e.g., invalid JSON response).

**5.2 Angular Application**

* Handles missing or malformed metadata gracefully by showing appropriate error messages.

**6. Implemented Enhancements and Security Improvements**

1. **Real-Time Metadata Updates**:
   * Automate the refresh of Angular components when new metadata is generated.
2. **Enhanced Security**:
   * Store API keys and sensitive data in environment variables or secure vaults.
3. **Comprehensive Logging**:
   * Implement detailed logs for tracking Flask and Groq API interactions.

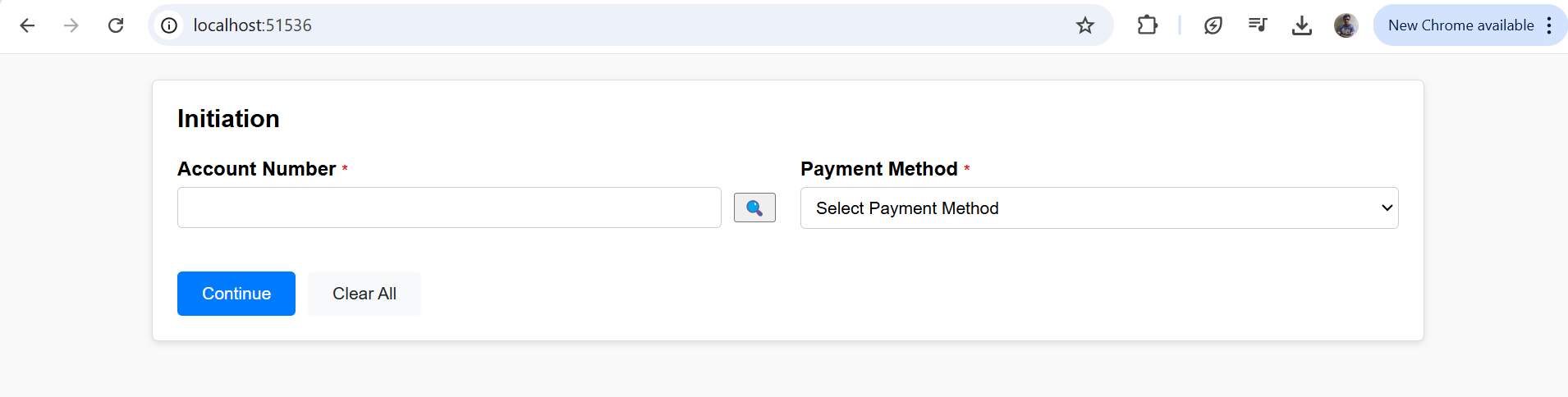
**7. Key Benefits of the Solution**

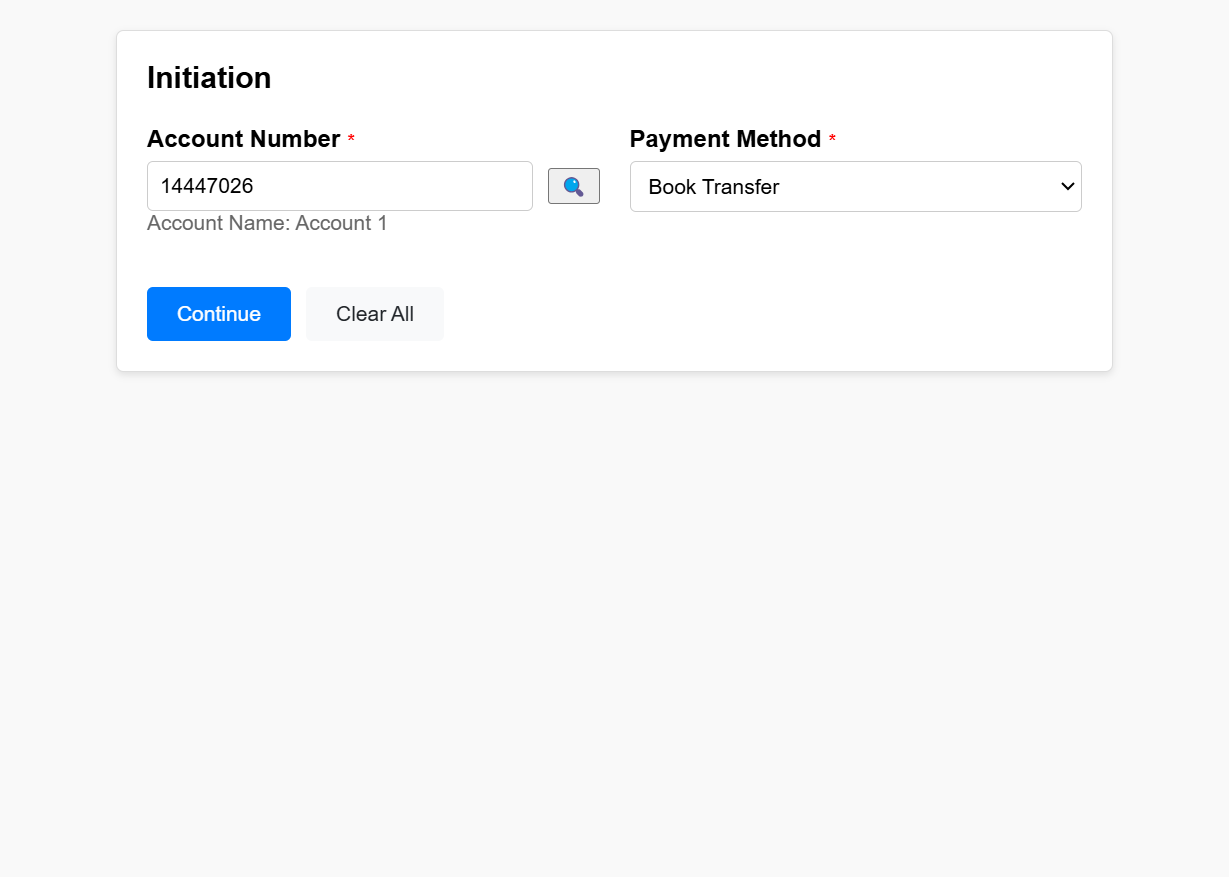
* **Dynamic UI Rendering**: Eliminates the need for hardcoded forms in Angular.
* **Scalability**: Easily supports new configurations and fields.
* **Efficiency**: Automates the metadata generation process.
* **Maintainability**: Decouples the front-end and back-end logic.

**8. Application Screen Shot**

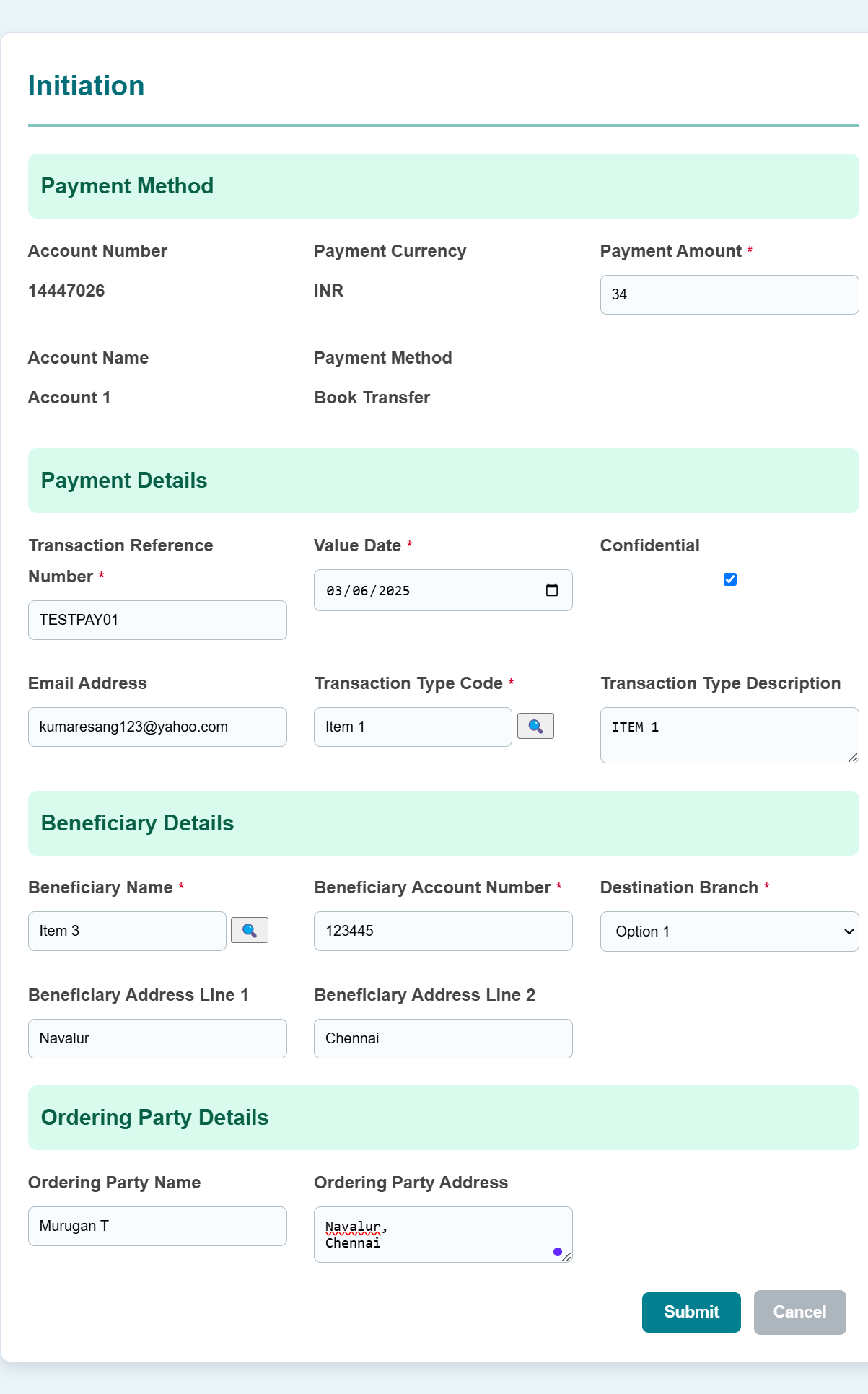
**Asp.net MVC Application (Source App)**

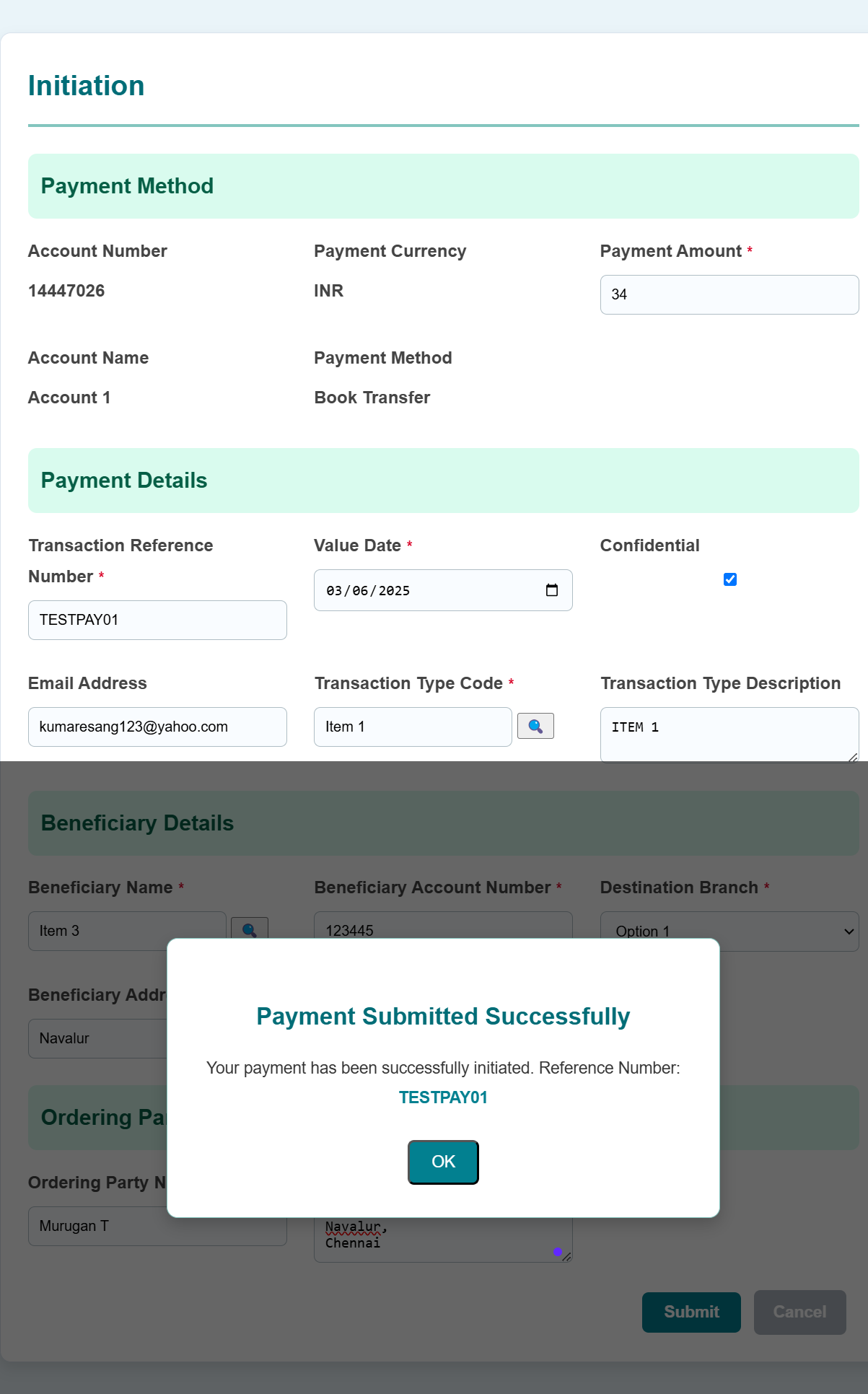
* **Select Account Number & Payment Method**

****

****

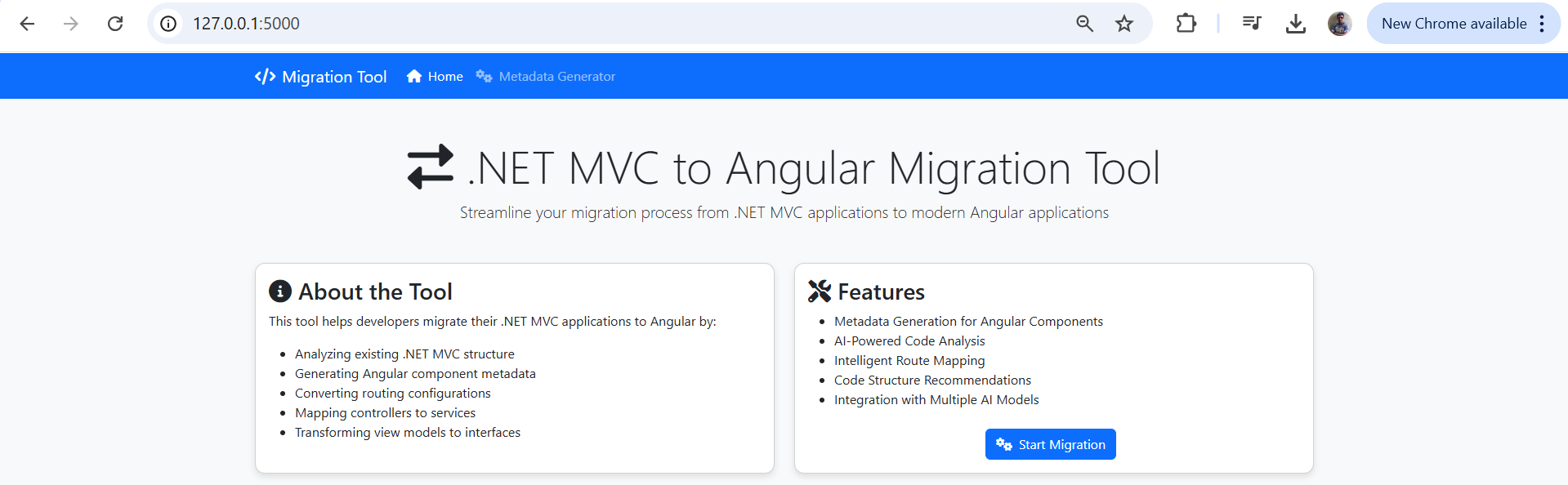
* **Click continue after selected account and payment method to get into payment initiation page to initiate a payment**

****

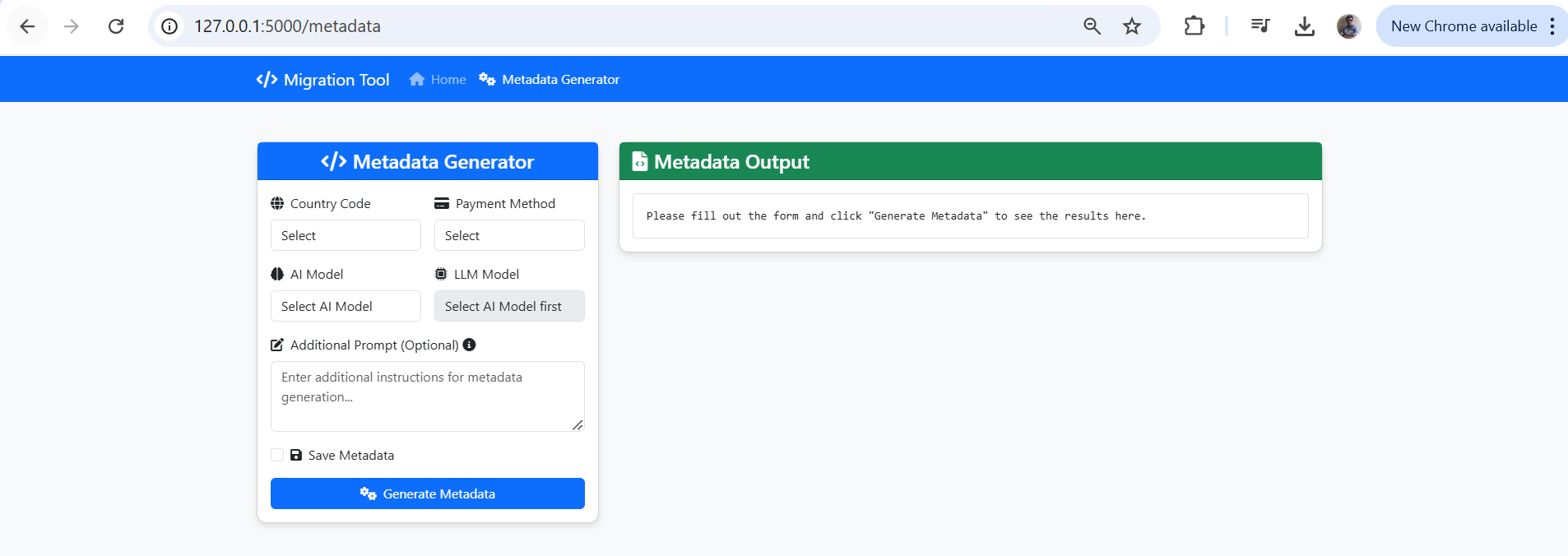
****

**Python (Mediator App – Metadata Generator using Gen AI)**

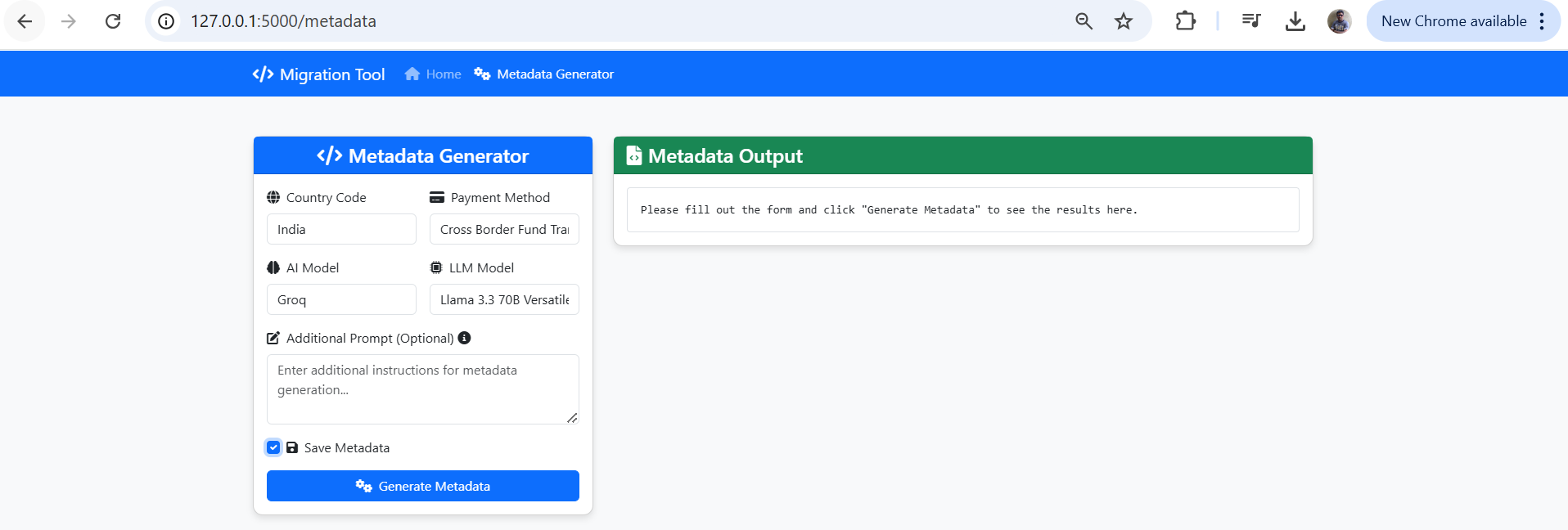
* **Landing Page**

****

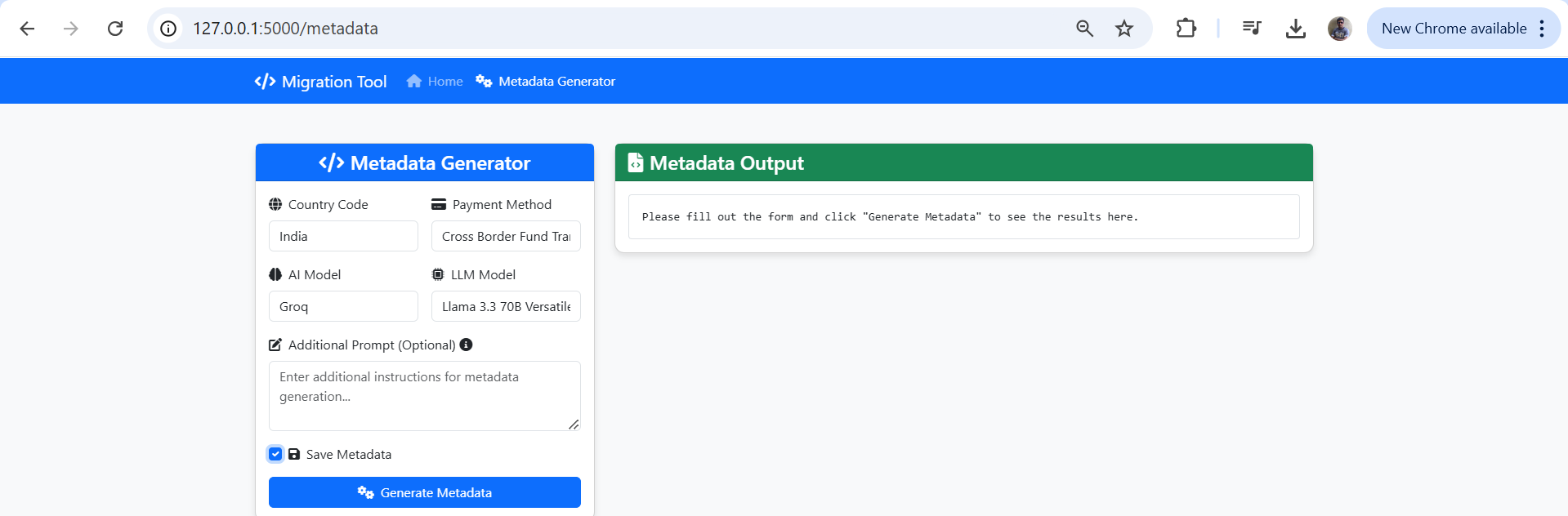
* **Click “Start Migration” button or “Metadata Generator” navigation link**

****

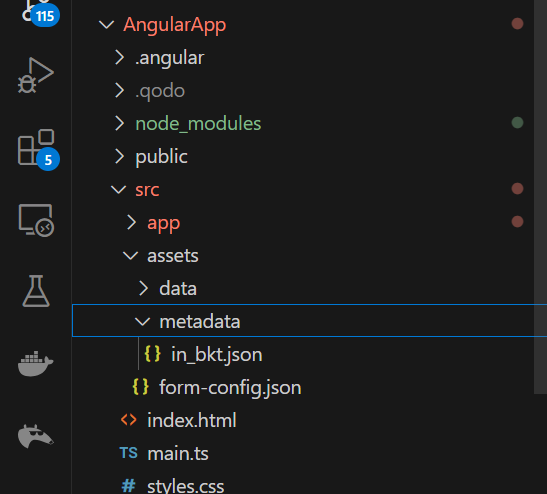
* **Select the respective country, payment method, and AI(Groq/Gemini) & LL Model which want to generate json/metadata for angular application.**

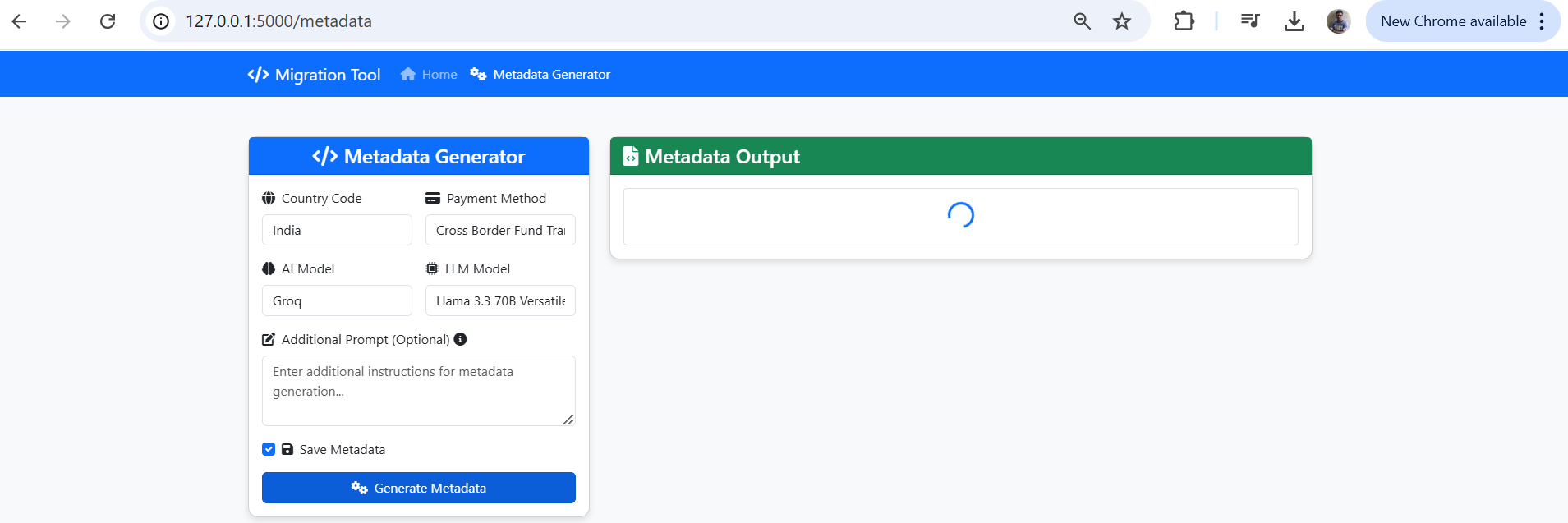
****

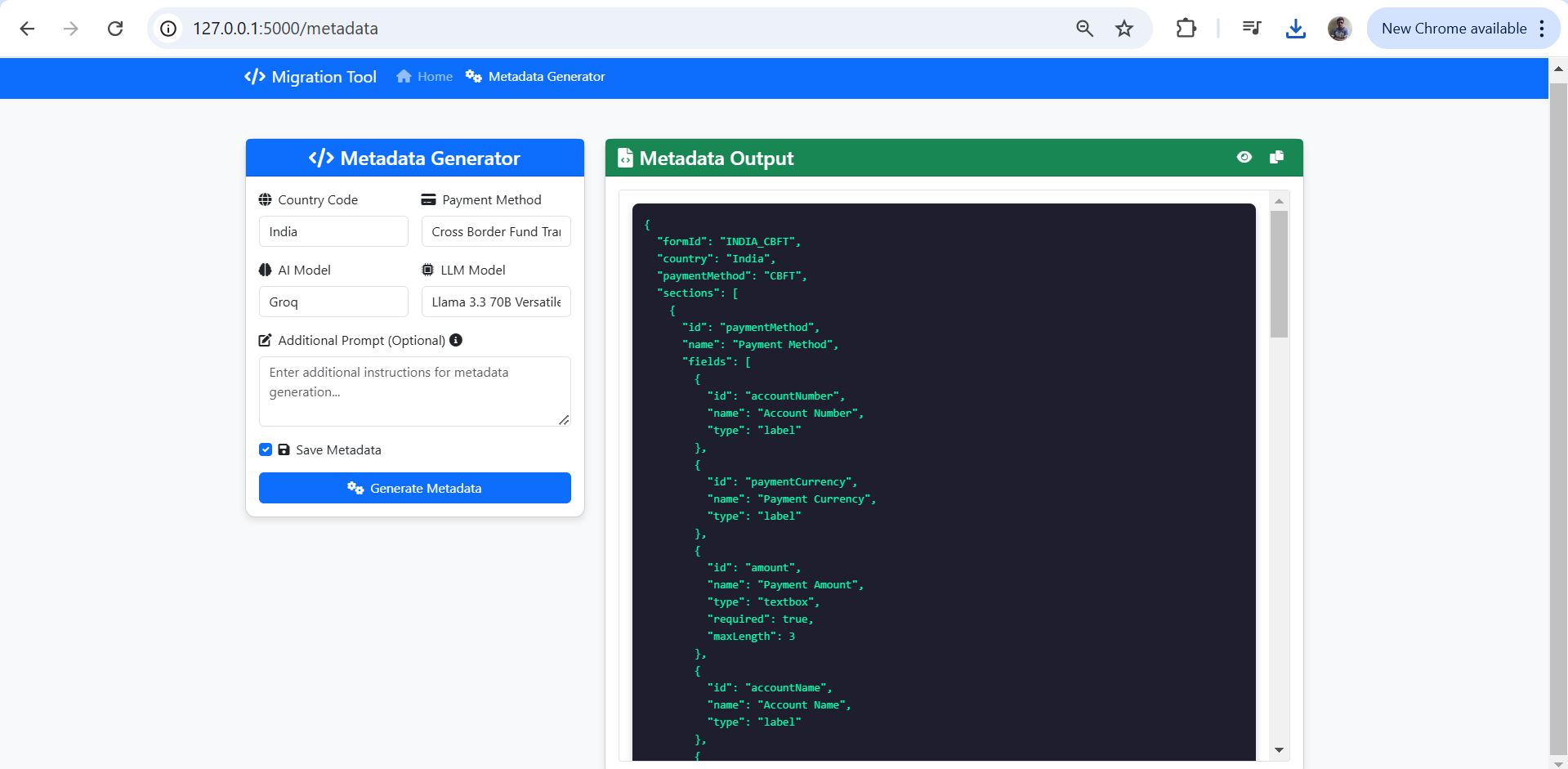
* **Click “Generate Metadata” to send API to python which internally connect to respective LLM Model to analyse the asp.net code and produce the metadata/json**

****

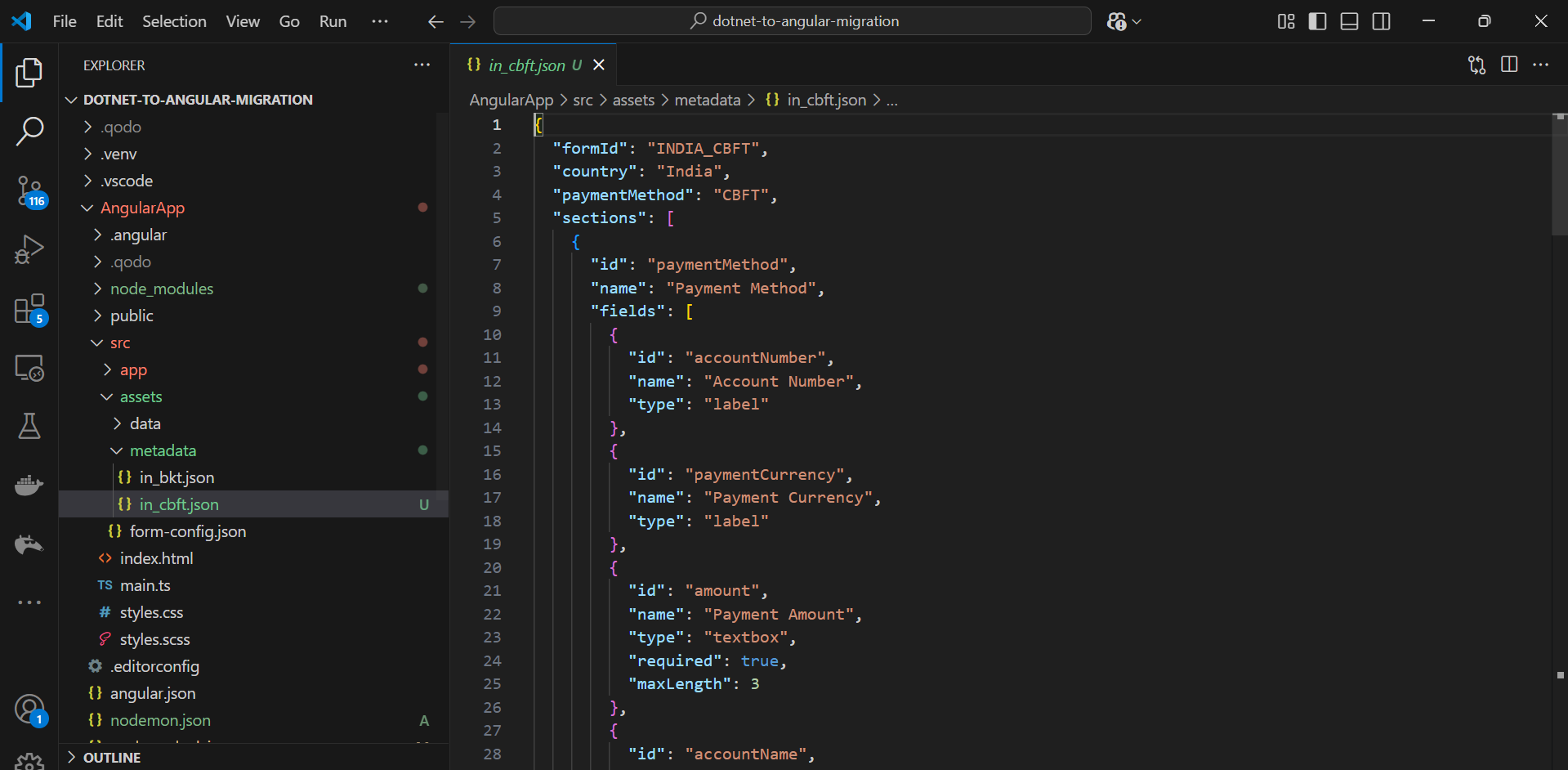
**Before**

****

****

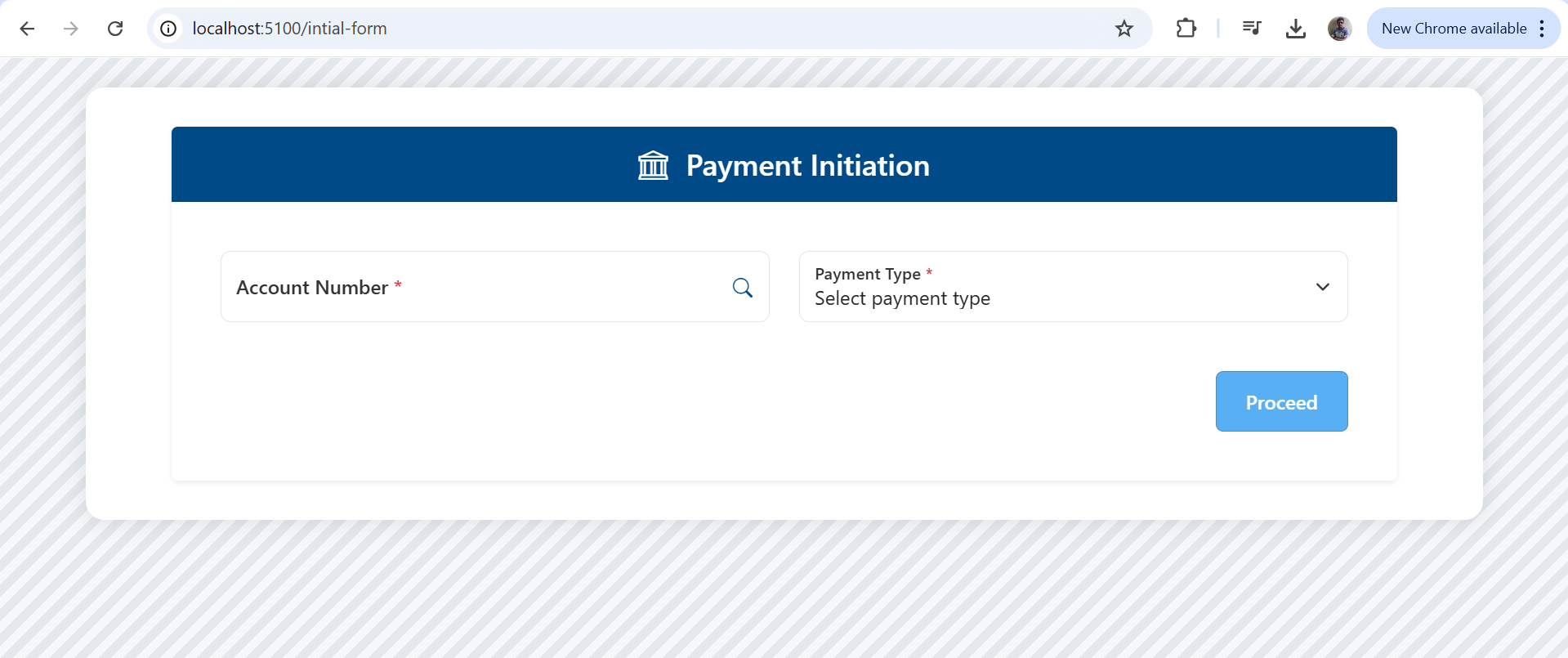
****

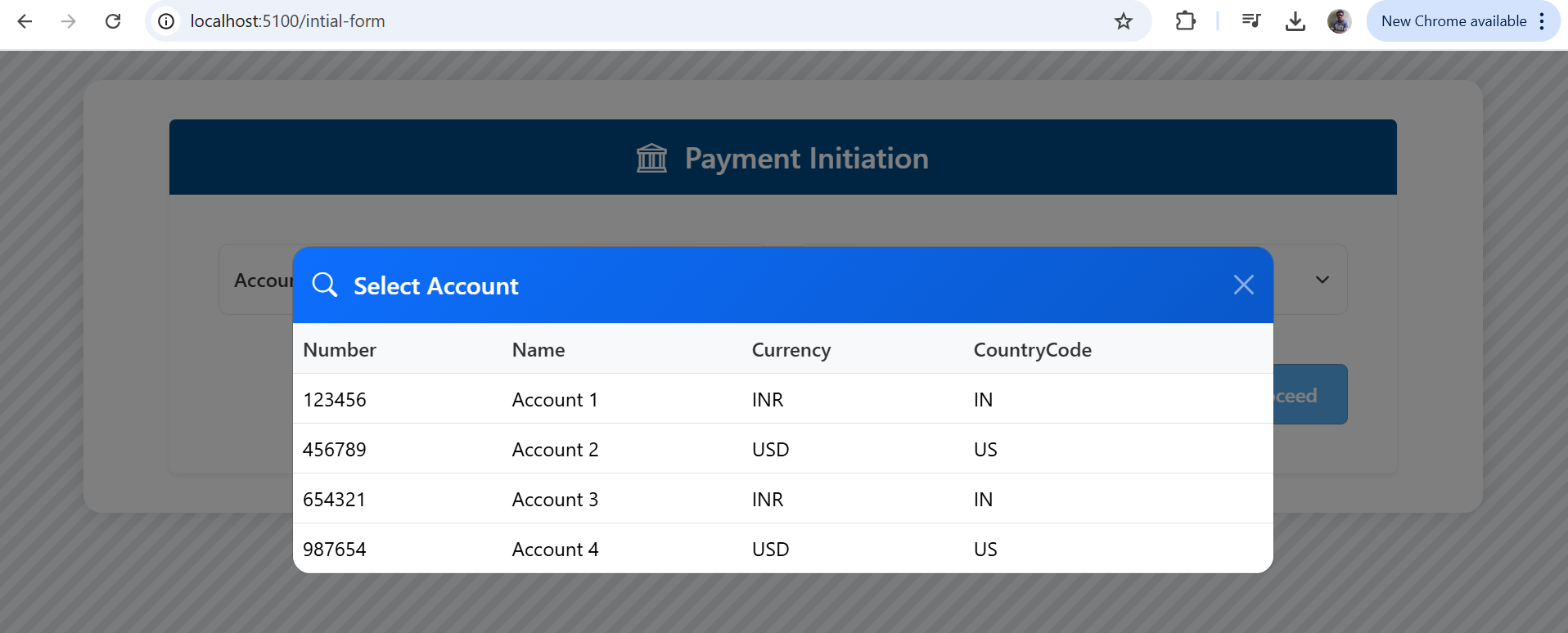
**After ( in\_cbft.json file automatically created)**

****

**Angular Application (Target App)**

* **Landing Page**

****

****

* **Click continue to get into payment initiation page**