**Automated Routing**

**1. Overview**

**The Automated Request Routing Microservice acts as a dynamic orchestration layer between two backend subsystems:**

1. **Prediction Service – Provides weather predictions based on user inputs and geolocation.**
2. **Recommendation Service – Suggests optimal destinations based on user preferences.**

**This microservice determines at runtime which subsystem to invoke, validates inputs, and ensures responses follow a consistent structured format.**

**2. Core Responsibilities**

* **Accept HTTP requests containing user input in the body and optional headers.**
* **Validate input fields: desiredWeather triggers recommendation; absence triggers prediction.**
* **Route requests to the appropriate subsystem based on input.**
* **Return responses with HTTP status codes, structured JSON, and input validation metadata.**
* **Handle errors gracefully, including invalid input (400) and internal failures (500).**

**3. API Endpoints**

**POST /automateRequest**

* **Accepts a request to determine routing.**
* **Request Body Example:**

**{**

**"inputsInfo": {**

**"country": "Egypt",**

**"desiredWeather": "Sunny",**

**"date": "2025-10-05"**

**}**

**}**

* **Optional Header: x-forwarded-for for IP-based geolocation if country is not provided.**

**4. Routing Logic**

1. **If inputsInfo.desiredWeather exists → forward request to Recommendation Service.**
2. **Otherwise → forward request to Prediction Service, using extracted IP if necessary.**
3. **Invalid input → return 400 Bad Request.**
4. **Internal failures → return 500 Internal Server Error.**

**5. Example Responses**

**Recommendation Response:**

**{**

**"recommedations": [**

**{ "country": "Egypt", "city": "Cairo", "reason": "Sunny and mild" },**

**{ "country": "Spain", "city": "Barcelona", "reason": "Matching desired weather" }**

**],**

**"inputs": {**

**"country": "Egypt is valid",**

**"desiredWeather": "Sunny is valid"**

**}**

**}**

**Prediction Response:**

**{**

**"predicted": [**

**{ "date": "2025-10-05", "temperature": 29, "humidity": 45 },**

**{ "date": "2025-10-06", "temperature": 27, "humidity": 50 }**

**],**

**"inputs": {**

**"country": "Egypt is valid",**

**"region": "Cairo is valid",**

**"city": "Cairo is valid",**

**"date": "2025-10-05 is valid"**

**}**

**}**

**Invalid Request Example:**

**{**

**"error": "Not valid request",**

**"message": "The user did not provide country and/or desired weather or they were invalid",**

**"details": "Please check the validity of the inserted data"**

**}**

**6. Use Cases**

| **Use Case** | **Input** | **Expected Output** |
| --- | --- | --- |
| **Basic Input (Prediction)** | **{"inputsInfo": {"country": "Egypt", "city": "Cairo"}}** | **Predicted weather for Cairo** |
| **Preference-based Input (Recommendation)** | **{"inputsInfo": {"country": "Spain", "desiredWeather": "Sunny"}}** | **Recommended sunny cities** |
| **Edge Case (Invalid Input)** | **{"inputsInfo": {"country": "Atlantis", "desiredWeather": "Stormy"}}** | **Error 400 Bad Request** |

**7. System Design & Flow**

1. **Request Validation**
   * **Checks existence of desiredWeather → routes to Recommendation Service.**
   * **Otherwise → routes to Prediction Service using IP-derived location.**
   * **Returns detailed input validation metadata.**
2. **Error Handling**
   * **400 Bad Request → Input missing or invalid.**
   * **500 Internal Server Error → System failure in underlying service.**
3. **Response Structure**
   * **Standardized JSON for all success and failure cases.**
   * **Includes inputs report for transparency and traceability.**

**8. Example Implementation (Node.js)**

**automated.js**

**const { prediction } = require('./prediction')**

**const { recommended } = require('./recommend')**

**async function automateRequest(req, res){**

**try{**

**const { inputsInfo } = req.body**

**let response**

**if(inputsInfo.desiredWeather){**

**response = await recommended(inputsInfo)**

**} else {**

**const ip = req.headers['x-forwarded-for'] || req.connection.remoteAddress**

**response = await prediction(inputsInfo, ip)**

**}**

**return res.status(response.status).json(response.data)**

**} catch(err) {**

**return res.status(500).json({ message: err.message })**

**}**

**}**

**module.exports = { automateRequest }**

**prediction.js**

**const { predict } = require('../predict/predict')**

**async function prediction(inputsInfo, ip){**

**try{**

**const predictSystem = new predict(ip)**

**const { predictData, inputsReport } = await predictSystem.prediction(inputsInfo)**

**return {**

**status: 200,**

**data: { predicted: predictData, inputs: inputsReport }**

**}**

**} catch(err){**

**return {**

**status: 500,**

**data: { predicted: null, inputs: null, message: 'Input data could not be processed' }**

**}**

**}**

**}**

**module.exports = { prediction }**

**recommend.js**