**Policy Management System Design, Implementation and Testing**

Source code: <https://github.com/ericguousaca/PolicyManagementSystemSolution>

1. **System Requirement**

Please refer to document **PolicyManagementSystem\_Requirement.docx** for System requirements

1. **System Design and Analyst**
   1. According the system requirements, designed below micro service architect to develop the Policy Management System:



* 1. Below tools and technologies will be used for development:

|  |
| --- |
| Asp.Net Core Web API (5.0 C#) |
| Sql Server Database |
| MongoDB |
| Rabbit MQ |
| Ocelot API Gateway |
| Angular |
| Kendo Angular UI |
| Visual Studio 2019 |
| Visual Studio Code |
|  |

* 1. Below design pattern will be used:

|  |
| --- |
| Dependency injection (All asp.net core Web Api will use the built-in dependency injection features to implement the controls, services and repositories etc. so that they can be easy maintenance and perform the unit testing by Mock technologies). |
| Observation Pattern (The angular app will adapt the Observation pattern in service to communicate with server side micro services) |
| Builder Pattern (Use the builder pattern to build the Policy Detail information. The Policy detail are combined a couple of different sections, for example, main body section, user types section, policy type section and maturity amount section, and in the real world, each section may be complicated and have different logic to build, so can use Builder pattern to create different builder class and object to build them in the different ways section by section). |
|  |
|  |

* 1. Below Security OWASP will be used:

|  |
| --- |
| 1: Data Access: Use of the Entity Framework Core to prevent the SQL injection. |
| 2: Nuget: Use Nuget to keep all of your packages up to date. |
| 3: Use allow-list validation on all user supplied inputs (Use built in Model Validation and created Custom Model Validation etc.) |
| 4: Added sufficient Logging & Monitoring features in code (Used Asp.Net core dependency injection to inject the ILogger object.) |

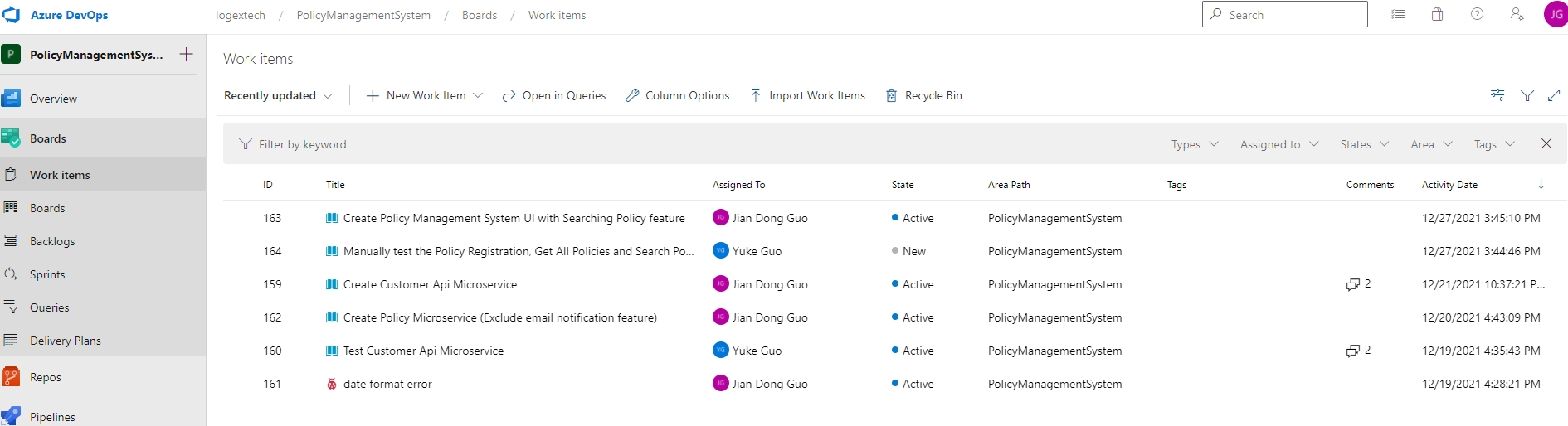
* 1. Below testing tools and methodologies will be used for testing:

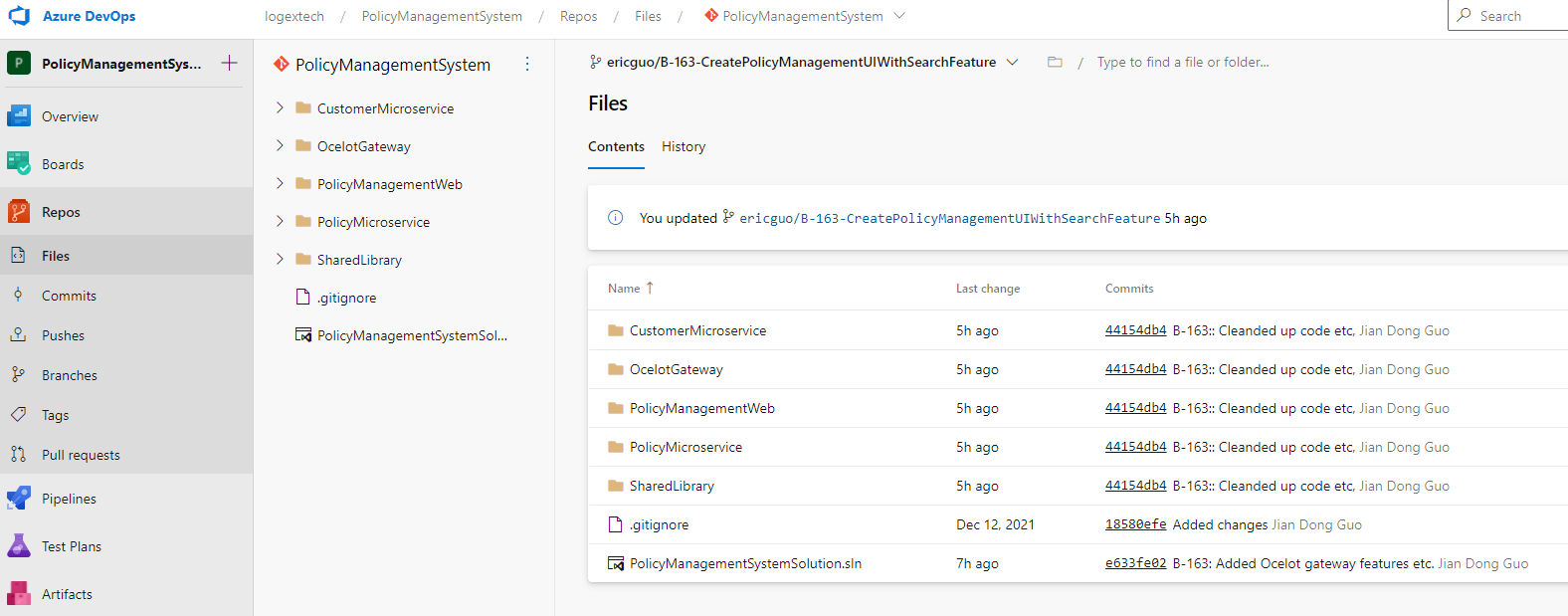
|  |
| --- |
| NUnit with Moq |
| Angular Unit Testing |
| Cypress for end to end testing |
| Postman |
| Manual Test Cases |
|  |

* 1. Below tool will be used for project management:

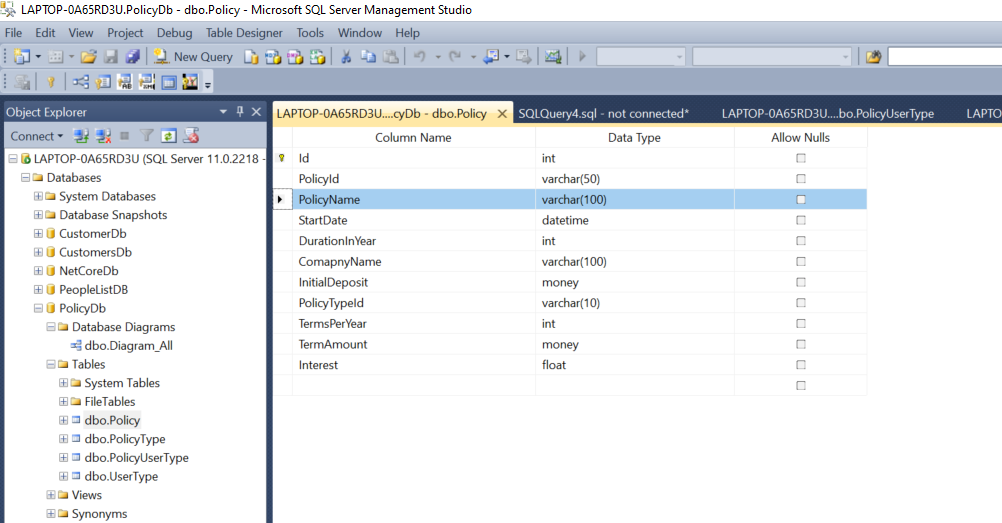
Microsoft Azure DevOps

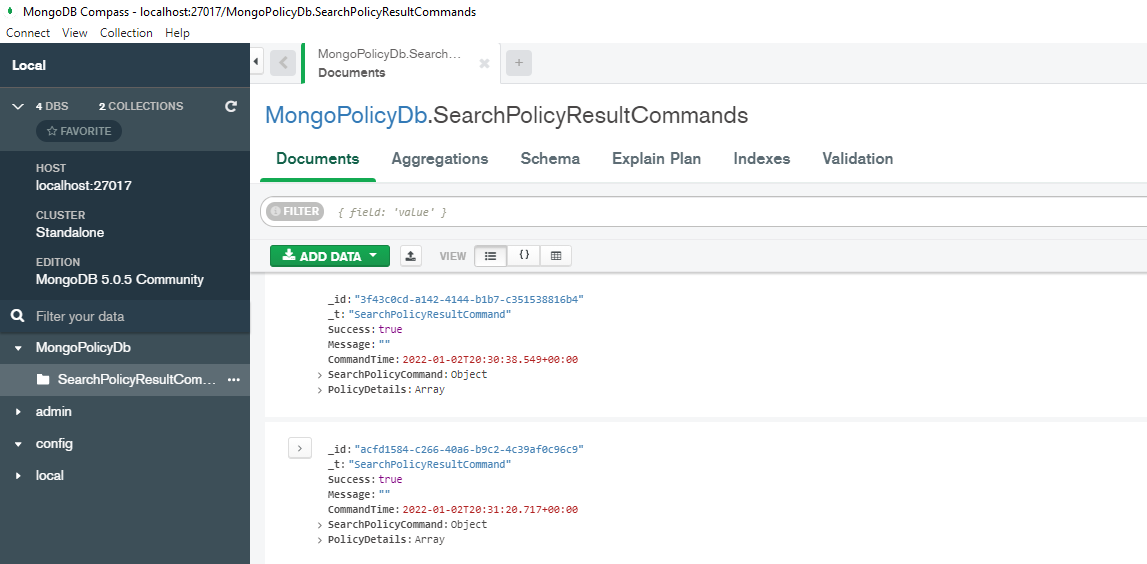
1. System Implementation
   1. Setup Azure DevOps environment for source code management and Agile project management:



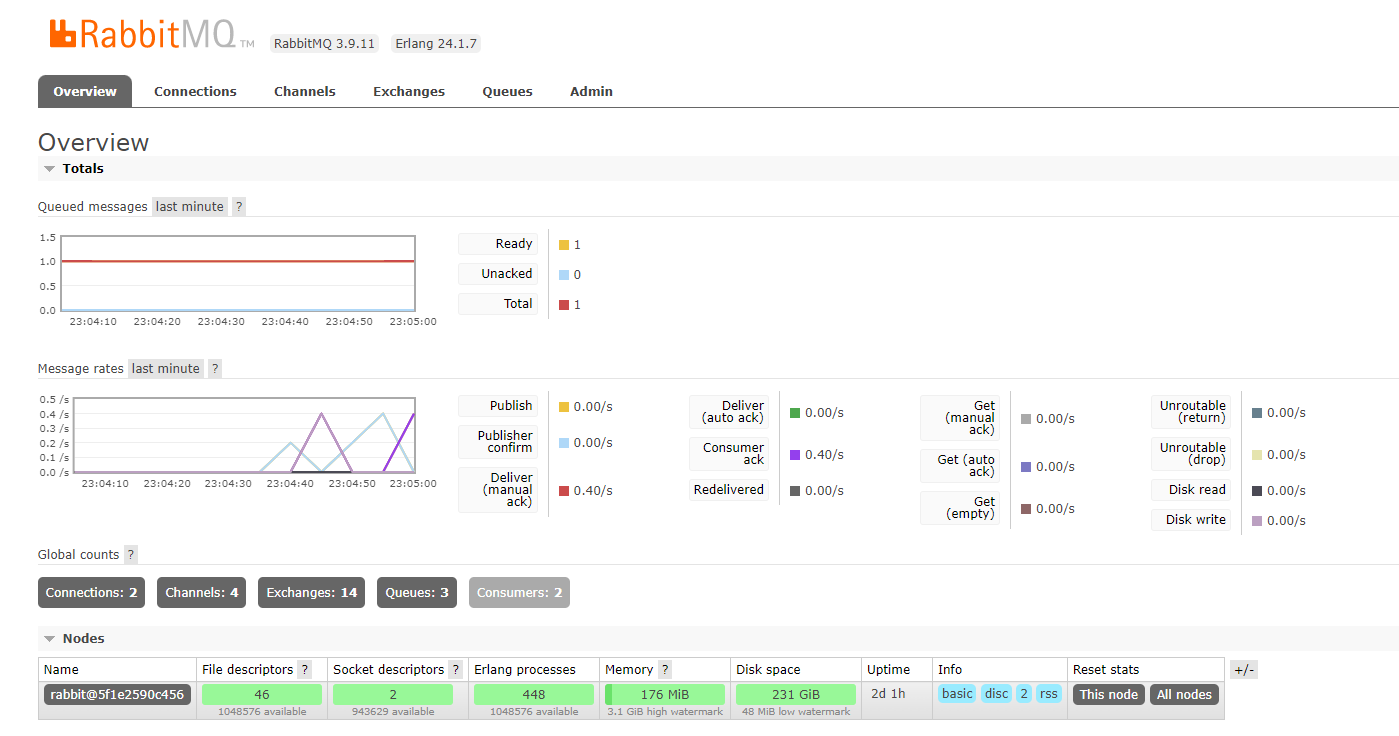


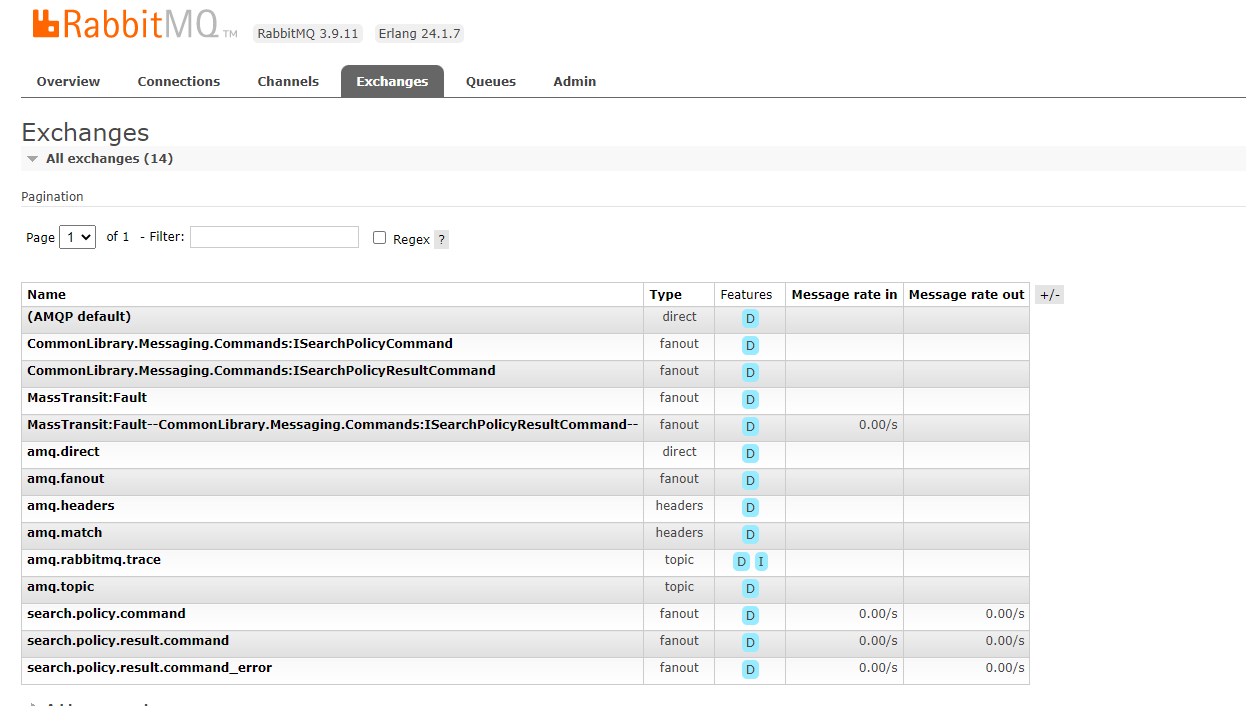
* 1. Install Sql Server and MongoDB locally:





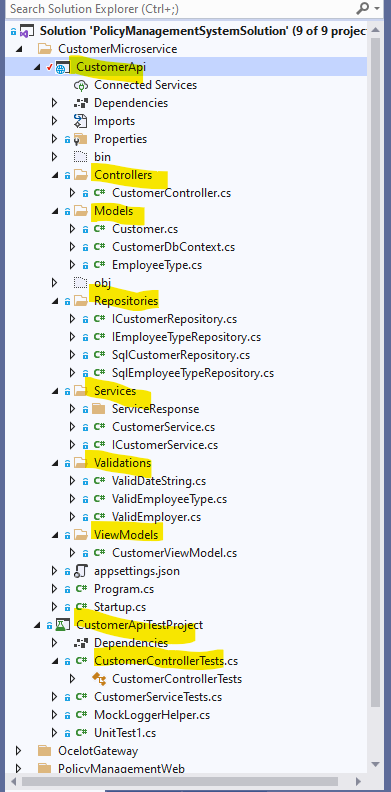
* 1. Install windows Docker, set it up to run Rabbit MQ container:



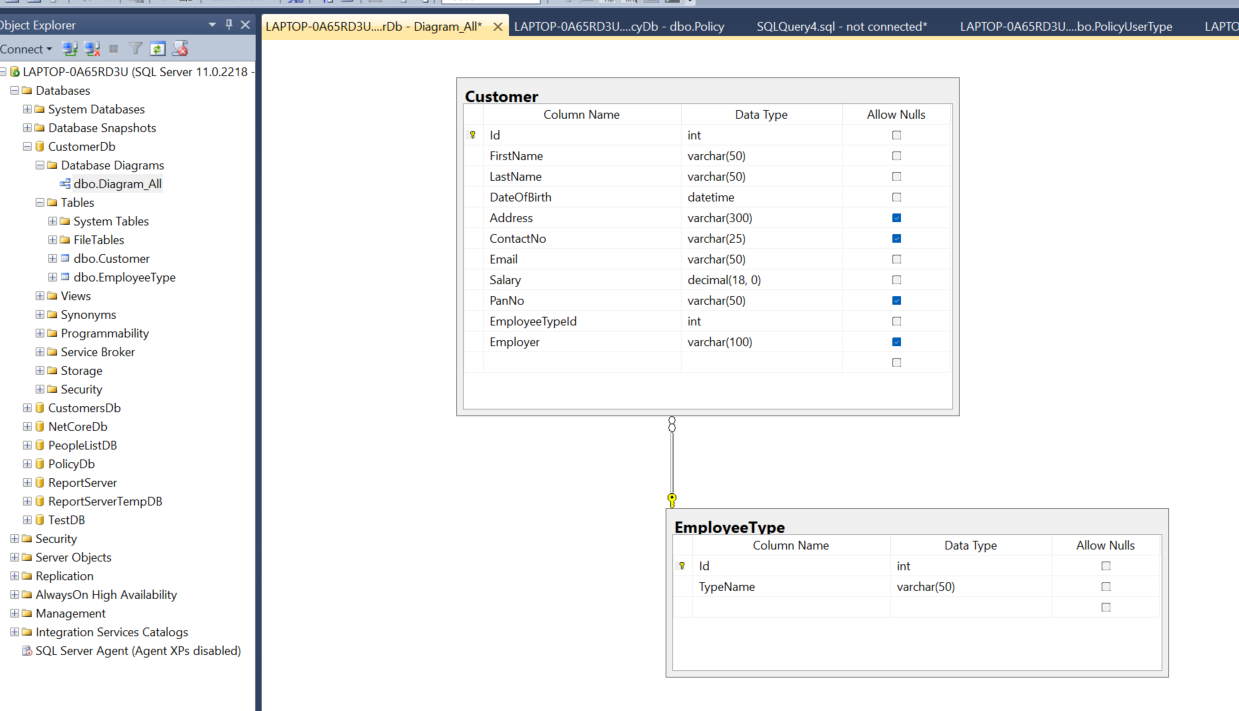


* 1. Implement the Customer Micro Service:
* Created below Customer Api project in Visual Studio 2019:

The micro service has the feature to add Customer into MS SQL Server DB.



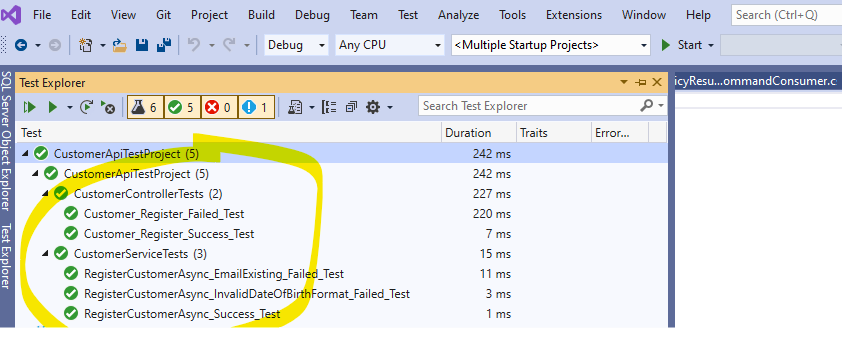
* Created SQL Server DB as below:



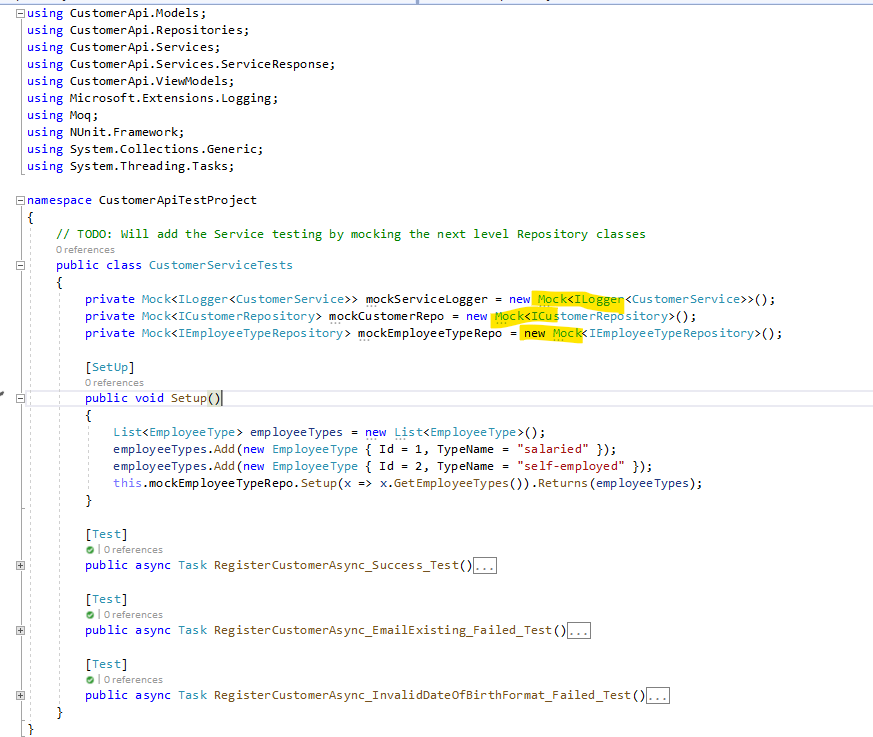
* Used below technologies to implement the project:

|  |
| --- |
| Dependency injection |
| Repository Pattern |
| Entity Framework Core |
| Asp.net built in Logging to log information/error in console |
| NUnit test with Moq |
| SQL Server DB |
|  |

* Unit Testing by using NUnit + Moq
  1. Implemented below Unit Testing for Customer API Micro Service:



* 1. Below is the sample Unit Testing code:

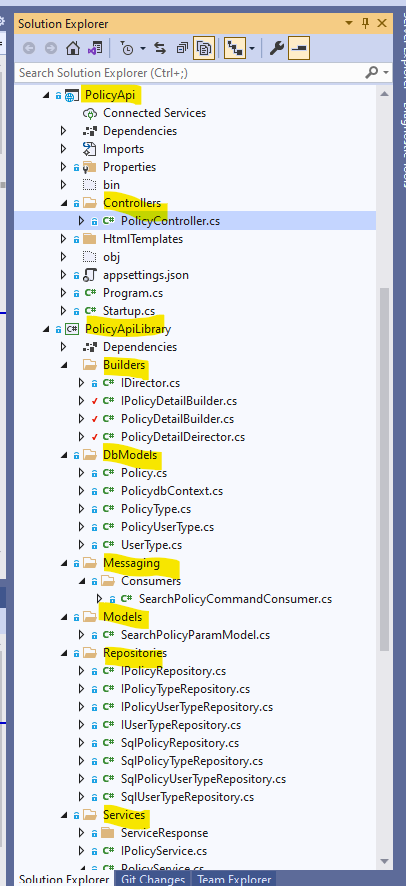


* 1. Implement the Policy Micro Service:
* Created below Policy API project in Visual Studio 2019:

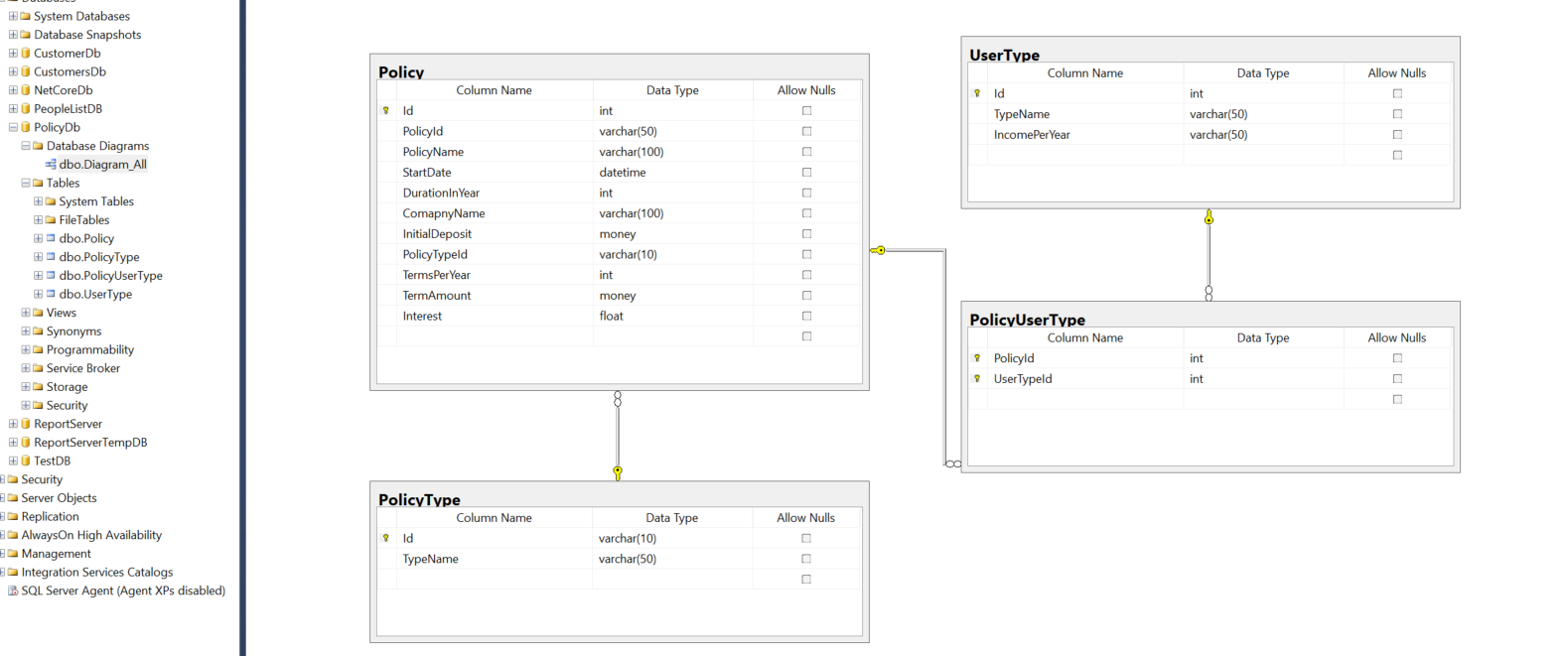
Since the API Project is complicated than Customer API so separate the Library into individual PolicyApiLibrary project.

The Policy API has below features:

* 1. Registers policy
  2. Gets all policy
  3. Searches policy
  4. Consumes the Rabbit MQ Search Policy Command, searches Policy and then sends the search result to Rabbit MQ



* Created below Database in Sql Server:



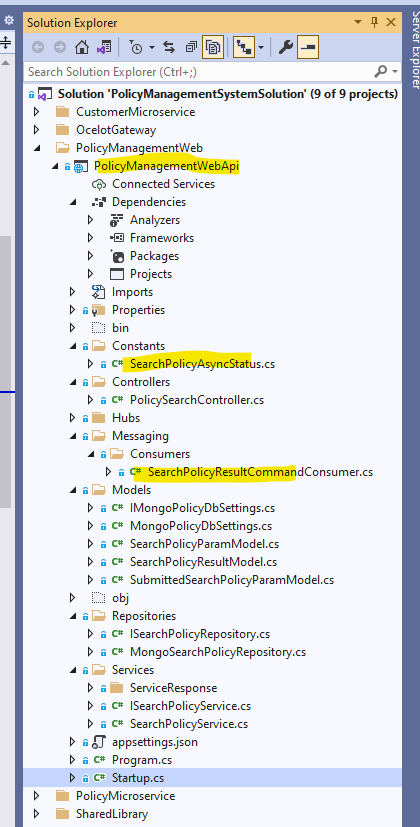
* Used below technologies to implement the project:

|  |
| --- |
| Dependency injection Pattern |
| Repository Pattern |
| Builder Pattern |
| Entity Framework Core |
| Asp.net built in Logging to log information/error in console |
| MassTransit and MassTransit.AspNet.Core |
| NUnit test with Moq |
| SQL Server DB |

* 1. Implement the Policy Management Web API Microservice
* Implement the PolicyManagementWebApi Micro Service in Visual Studio 2019:

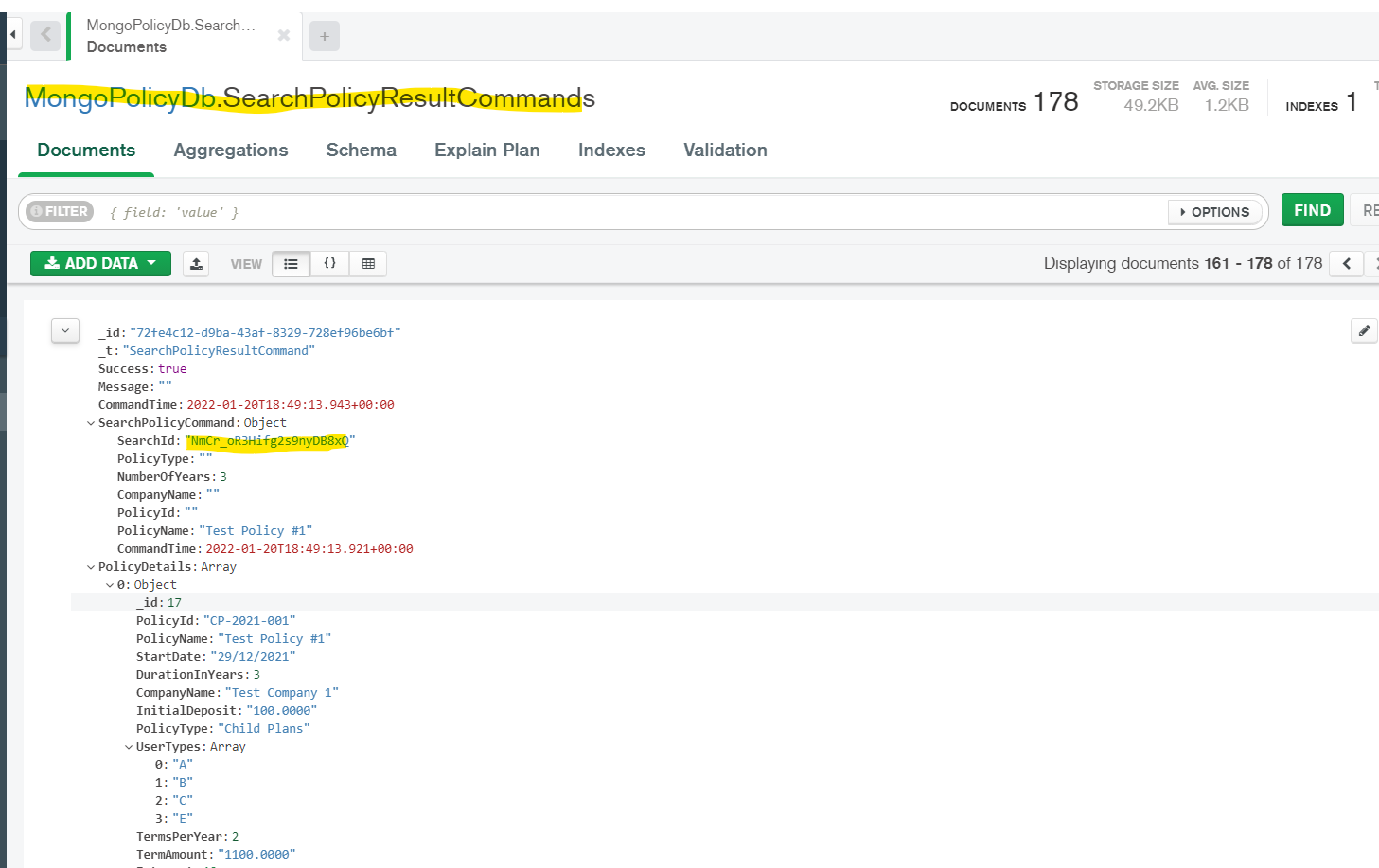
The project is the glue API project between front end app and back end API service and has below features:

* 1. Client app (Angular app) can call the service to submit the Search Policy Command to Event Bus (Rabbit MQ).
  2. Can consume the Search Policy Result Command from Event Bus (Rabbit MQ) and store in MongoDB.
  3. Keep the persistent connection with client by using SiganalR (it has built in WebSocket features inside the package) technology.
  4. SignalR will communicate with client for each status changes in Async Policy Search feature.
  5. Client app (Angular app) also can call the service to get Search Policy result from MongoDB by providing the unique Search Id.





* Created below MongoDB database and collection to store Search Policy Results from Event Bus (Rabbit MQ):



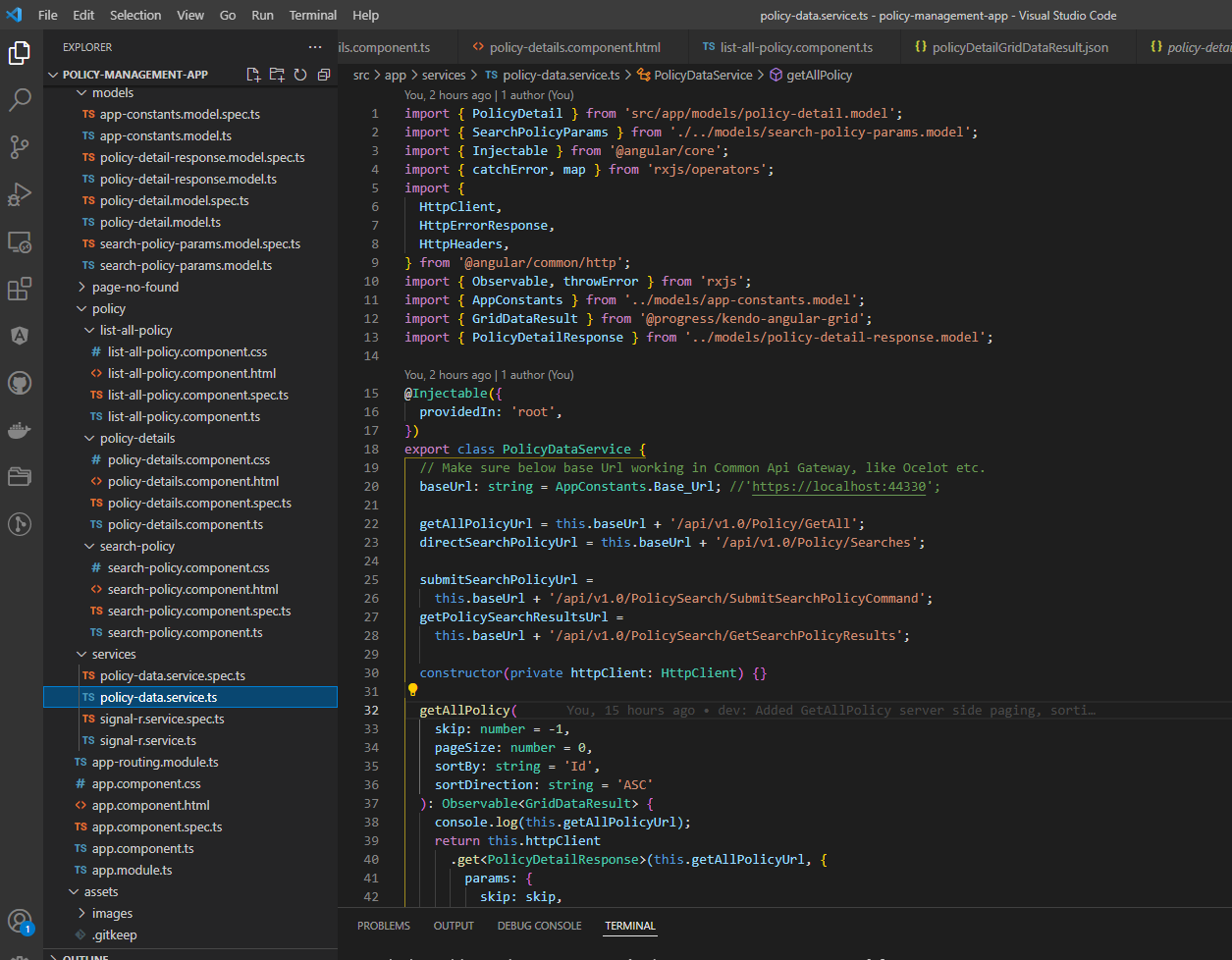
* Used below technologies to implement the project.

|  |
| --- |
| Dependency injection |
| Repository Pattern |
| Entity Framework Core |
| Asp.net built in Logging to log data in console |
| MassTransit and MassTransit.AspNet.Core |
| MongoDB |
| SignalR for persistent communication with Client App (Async Policy Search) |

* 1. Implement the Angular App
* Used Visual Studio Code to implement below client **policy-management-app** (under folder: PolicyManagementSystem\PolicyManagementWeb\policy-management-app) by using Angular technology

The Angular app has below features:

* 1. Use Kendo Angular UI Gird to show all policy directly by calling Policy API Micro Service with server side pagination, sorting features
  2. Can search policy directly by calling Policy API Micro Service and show in Kendo Angular UI grid with Client side pagination and sorting features
  3. Can call the PolicyManagementWebApi Micro Service to submit Search Policy command (including search parameters and unique Search Id in the command object) to Event Bus (Rabbit MQ).
  4. Importing SignalR Angular model so that can keep persistent communication with PolicyManagementWebApi Micro Service and update the status and search results automatically in Kendo Angulat UI Grid with client side pagination and sorting features.
  5. Can also load search policy results by calling the PolicyManagementWebApi Micro Service to query MongoDB with the Unique Search Id and show the results in Kendo Angulat UI Grid with client side pagination and sorting features.



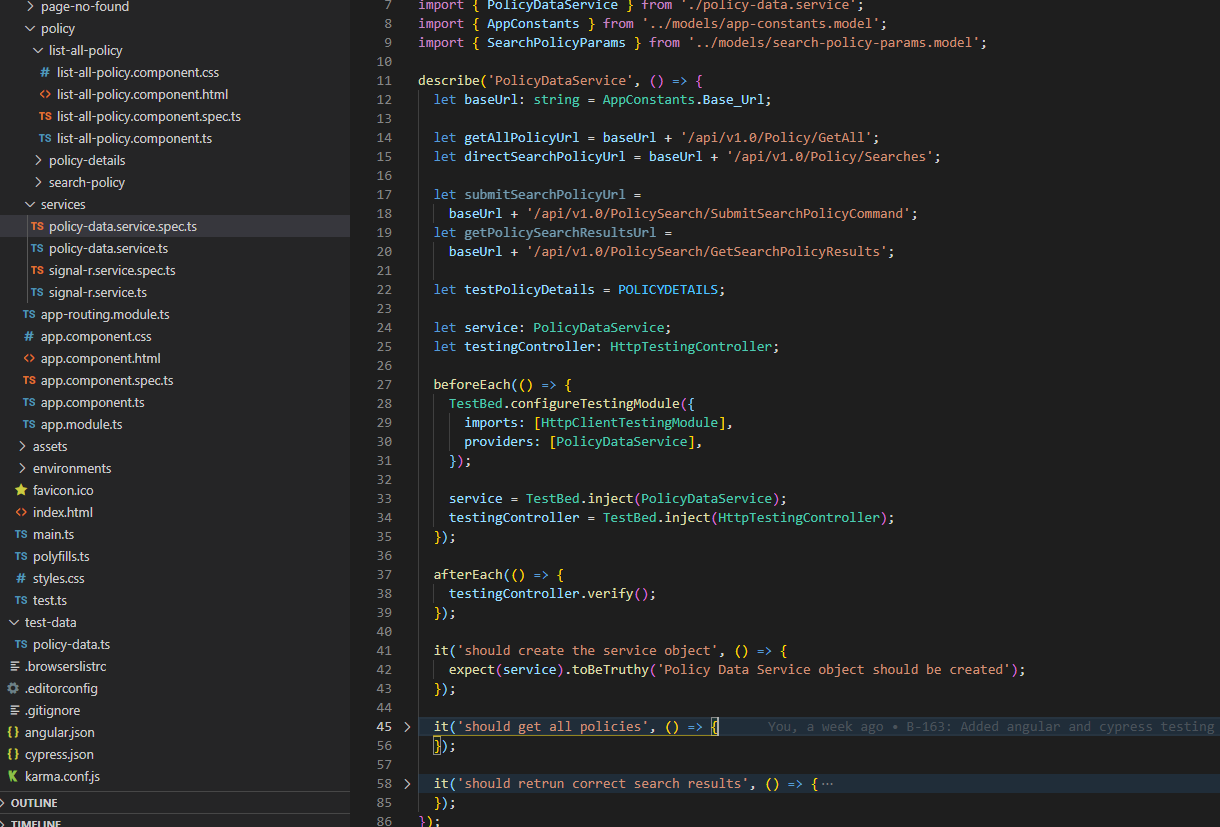
* Used below technologies to implement the project:

|  |
| --- |
| Angular |
| Bootstrap 5 |
| Kendo Angular UI |
| Dependency injection |
| Observer pattern |
| Reactive forms |
| Angular Routing |
| Type Script |
| RxJS |
| SignalR Package (To communicate with Policy Management API for Async Policy Search) |
| HttpClient |
| Karma and Jasmine (Angular Unit Testing) |
| Cypress (End to end testing) |

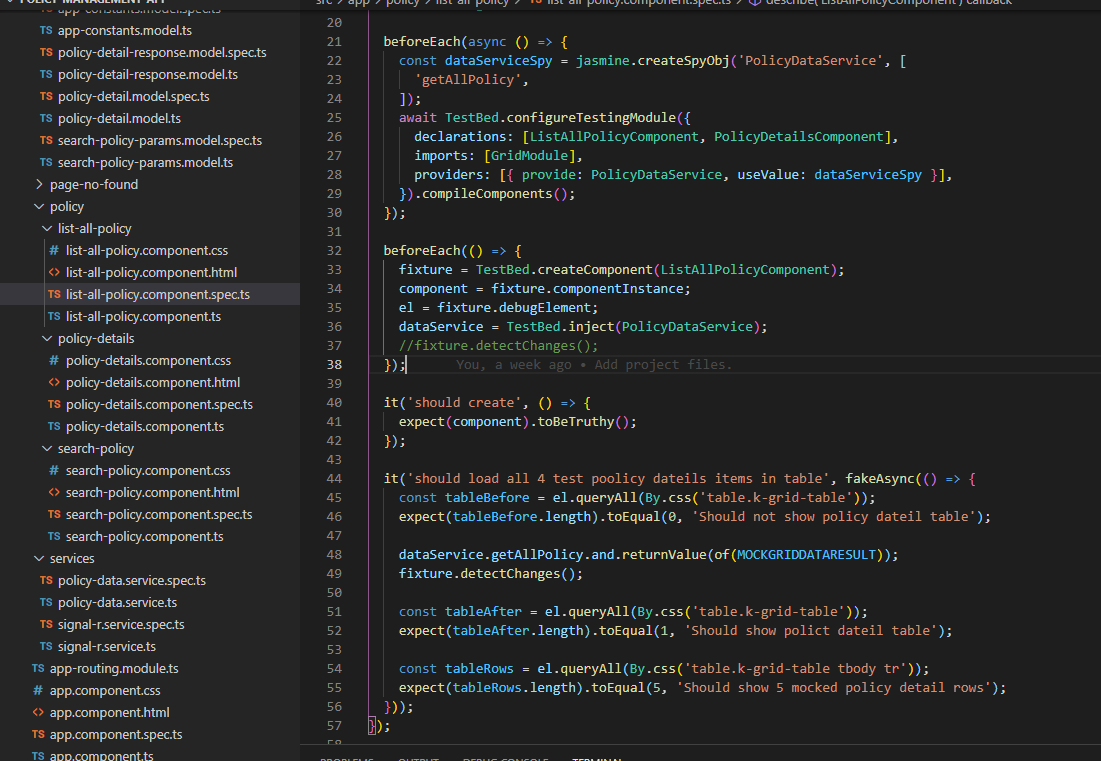
* Angular Unit Testing:

Implemented unit testing on some of services and components by Karma and Jasmine as below:

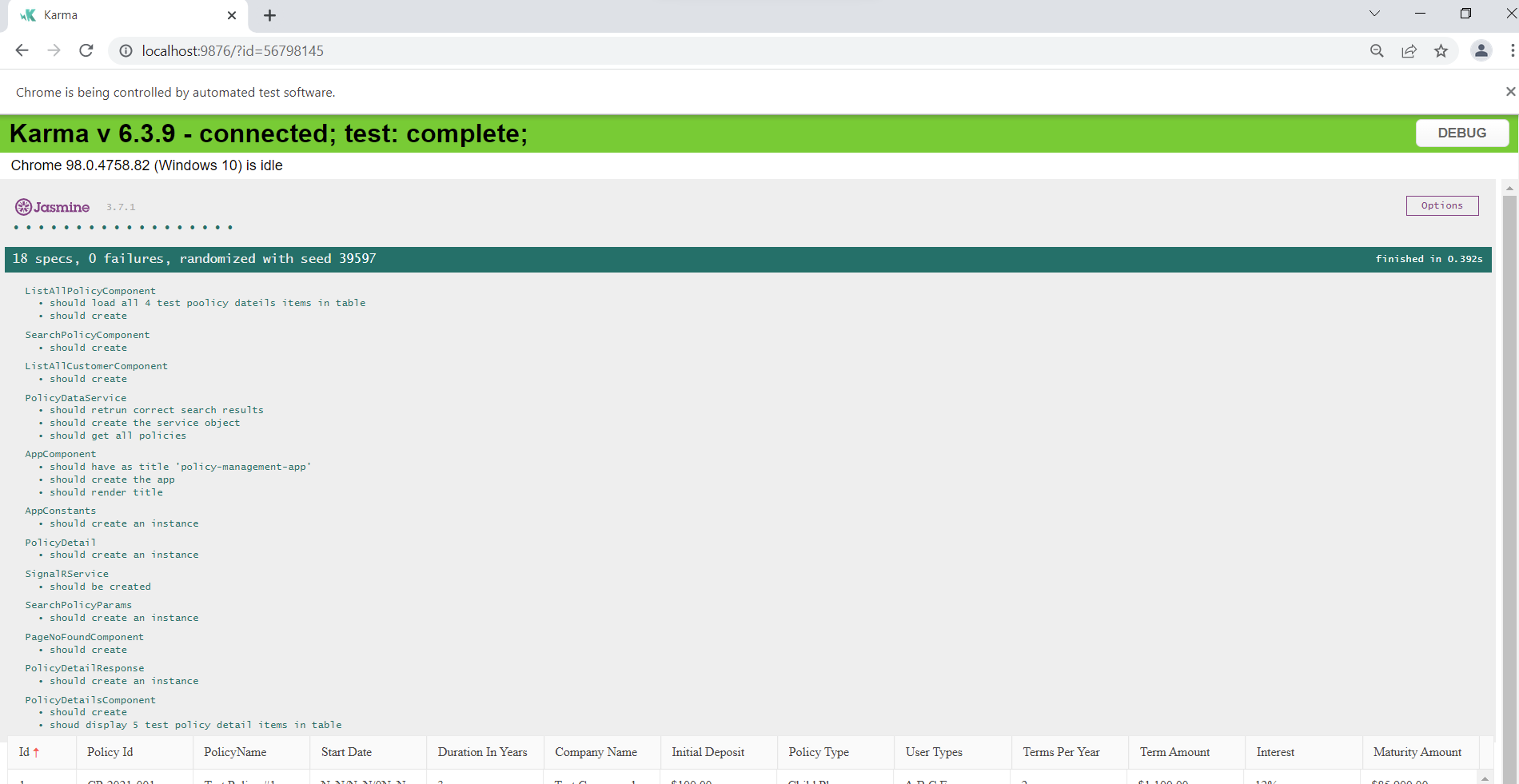
1. Unit Testing on PolicyDataService by Mock Data:



1. Unit Testing on ListAllPolicyComponent by Mocked Service, Data on Kendo Angular UI Grid:

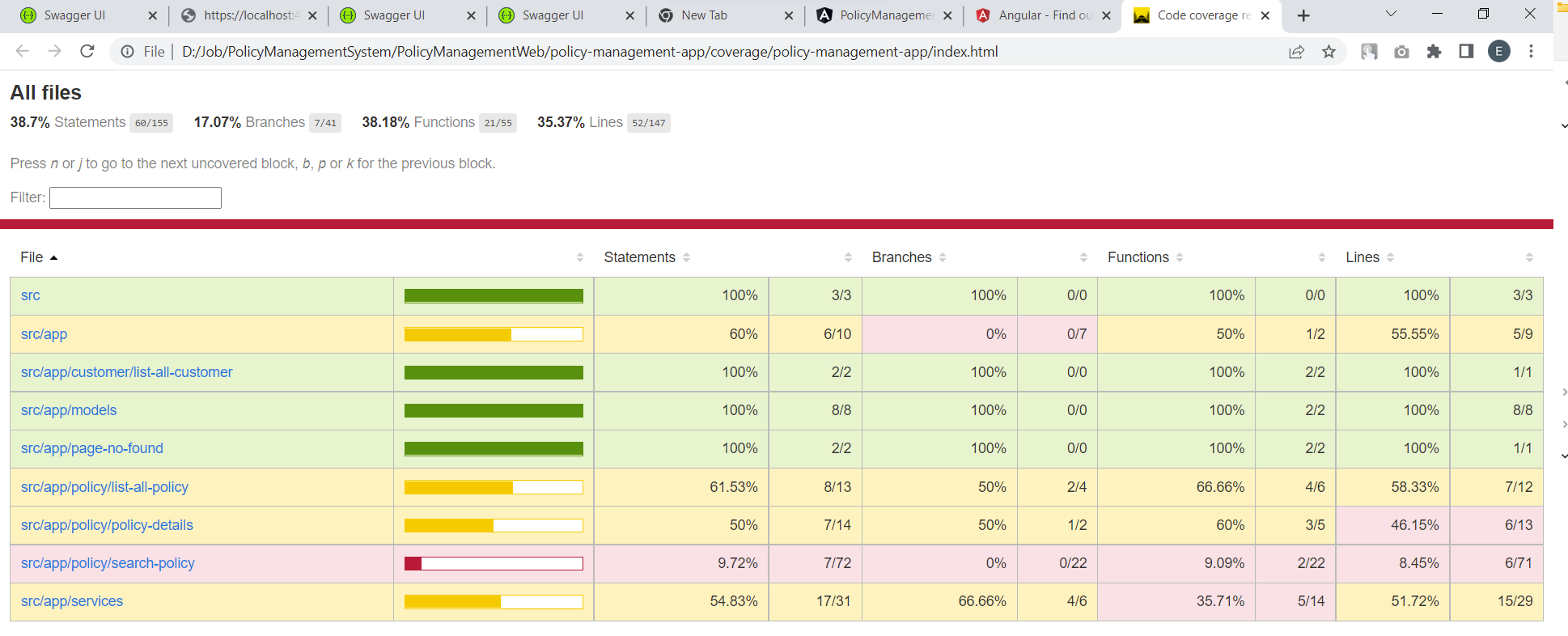


1. Unit Testing Output:



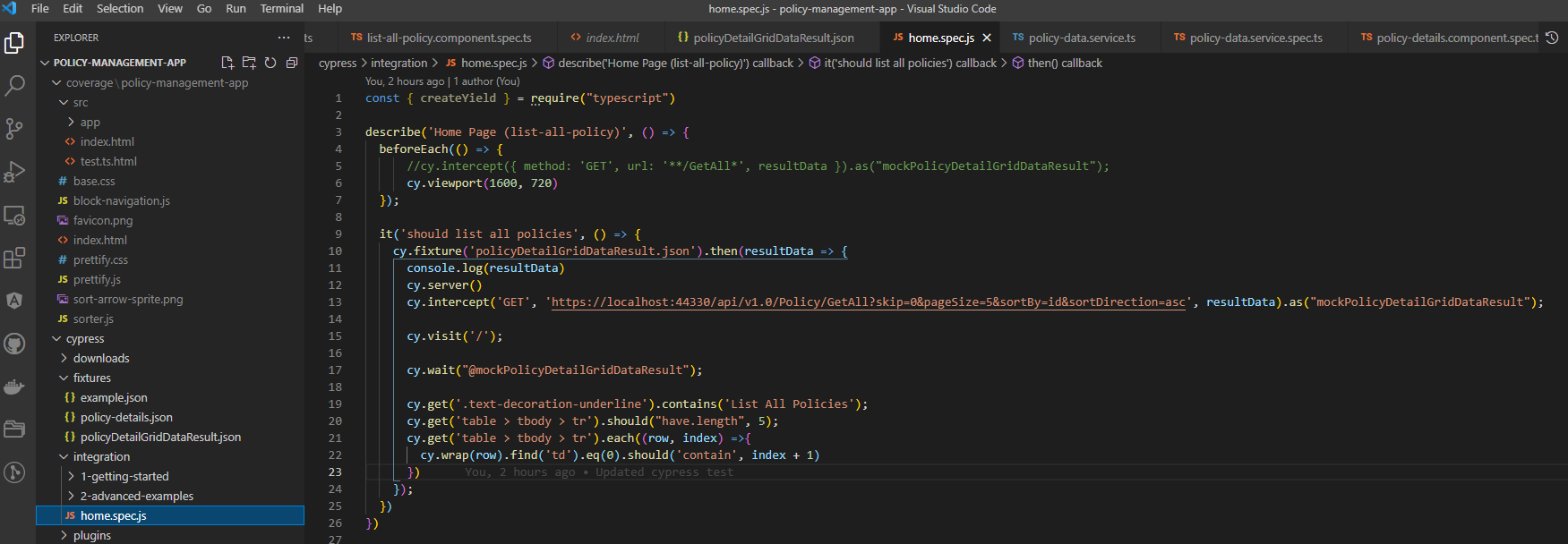
1. Below is the code coverage of Unit Testing:

ng test --no-watch --code-coverage

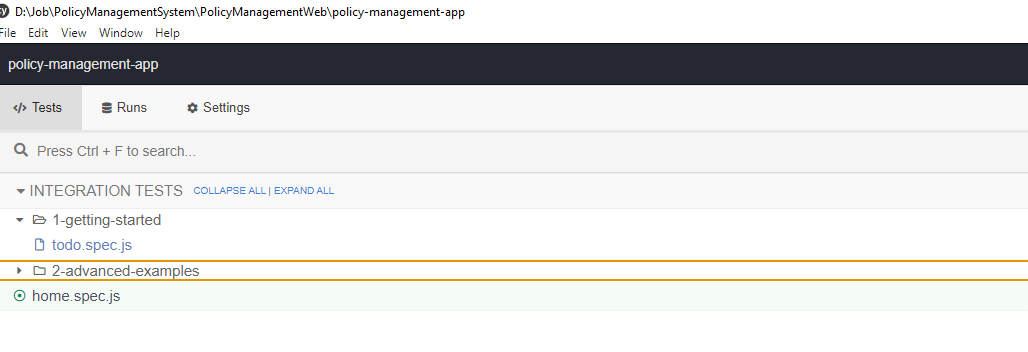


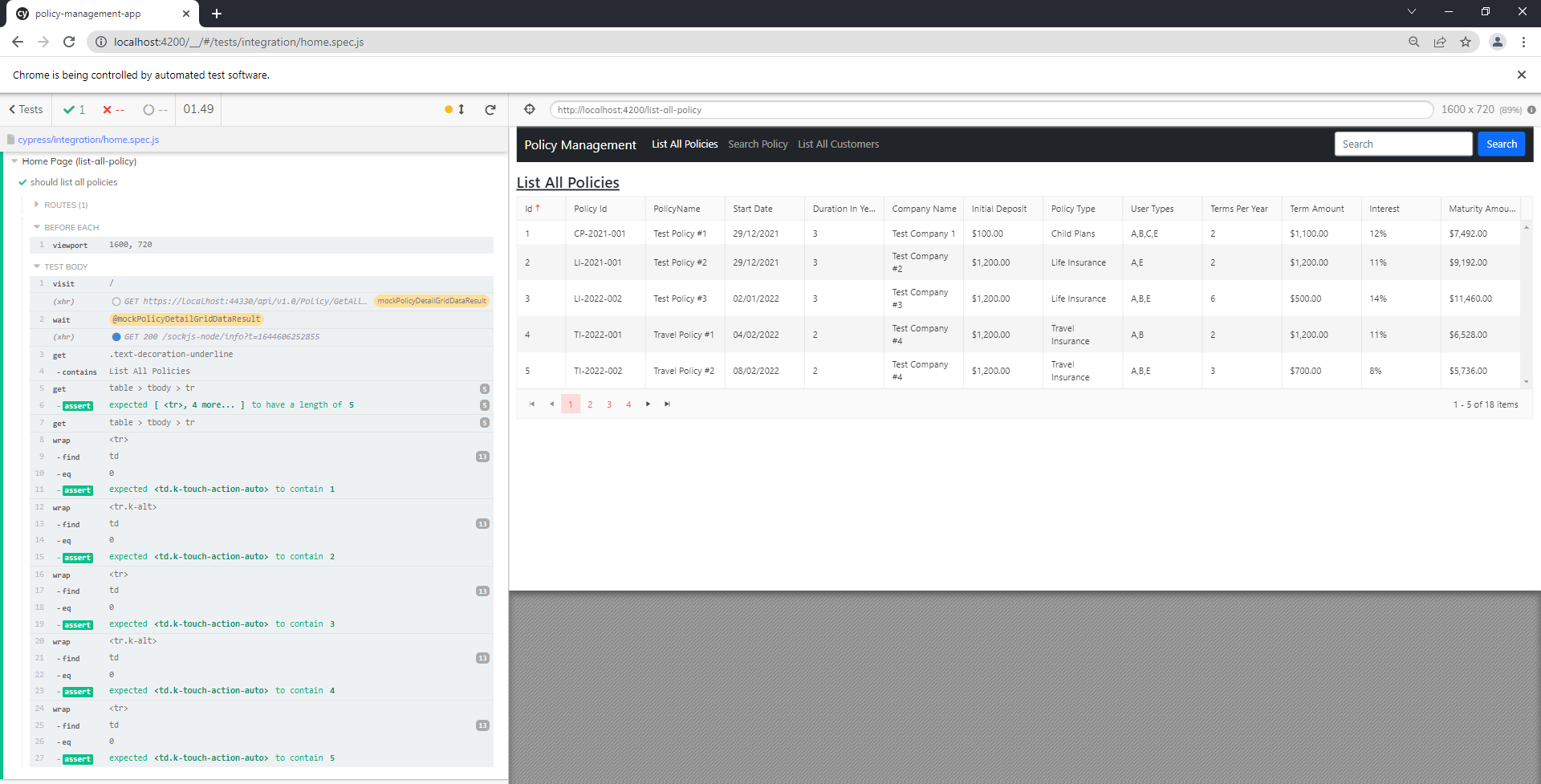
* Angular End to End testing by Cypress:

Implemented simple end to end testing by using Cypress:

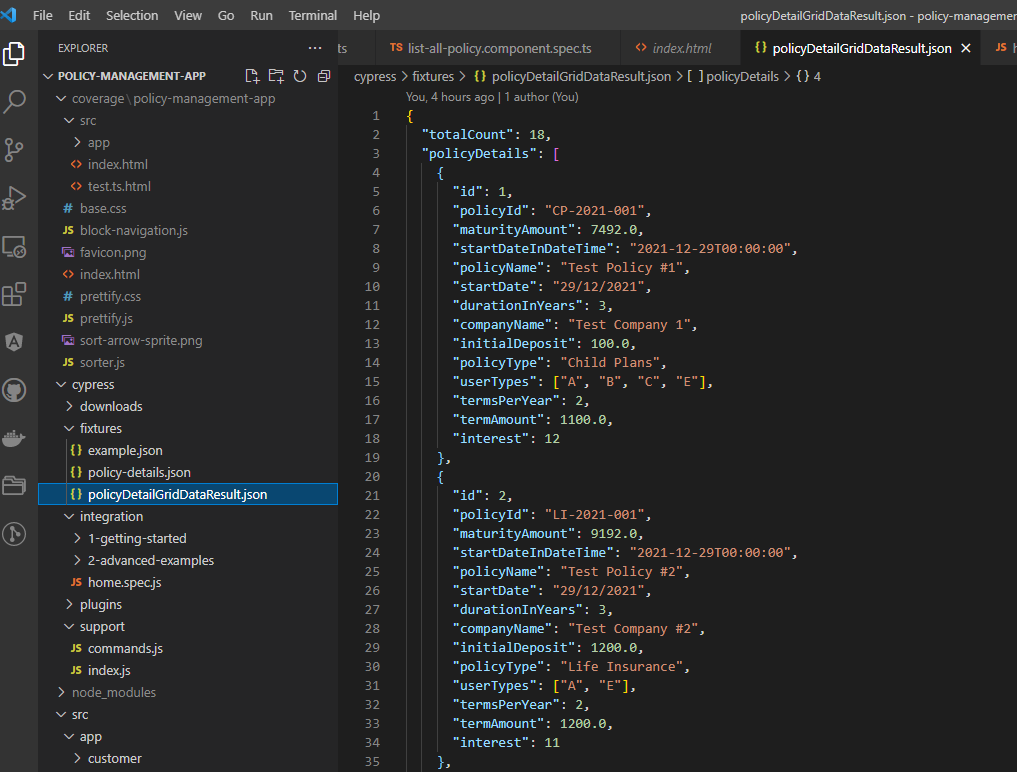


And below is the screenshot of Cypress testing on home page that loads the Mocked Policy data (Id is from 1 to 5) in the end to end testing:



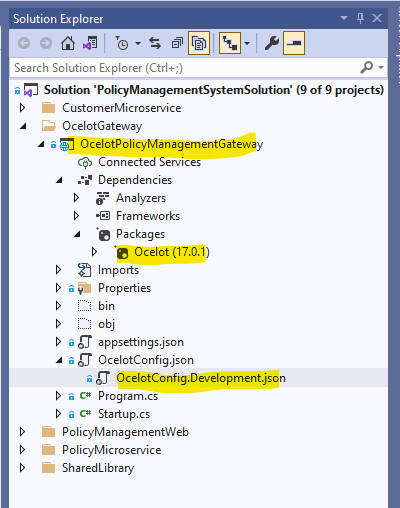


Mocked data for above Cypress testing:

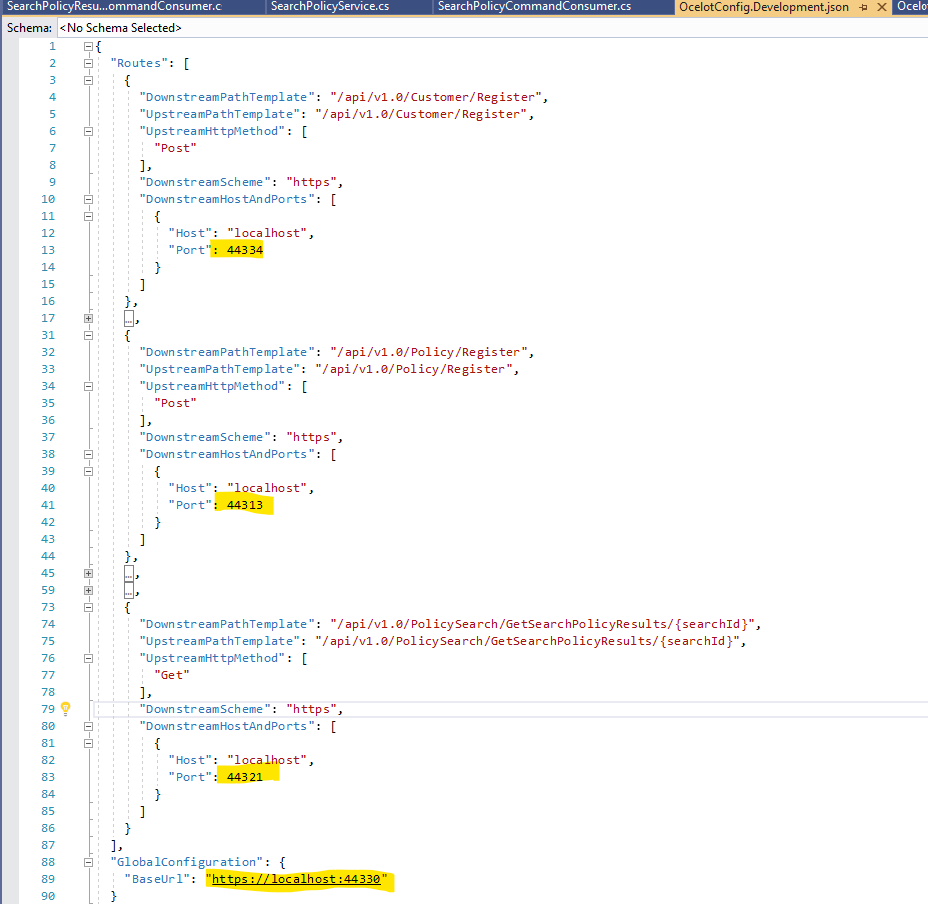


* 1. Implement the Ocelot API Gateway App
* Implemented the OcelotPolicyManagementGateway project in Visual Studio 2019.

The project can route the API endpoints by the common gateway URL.



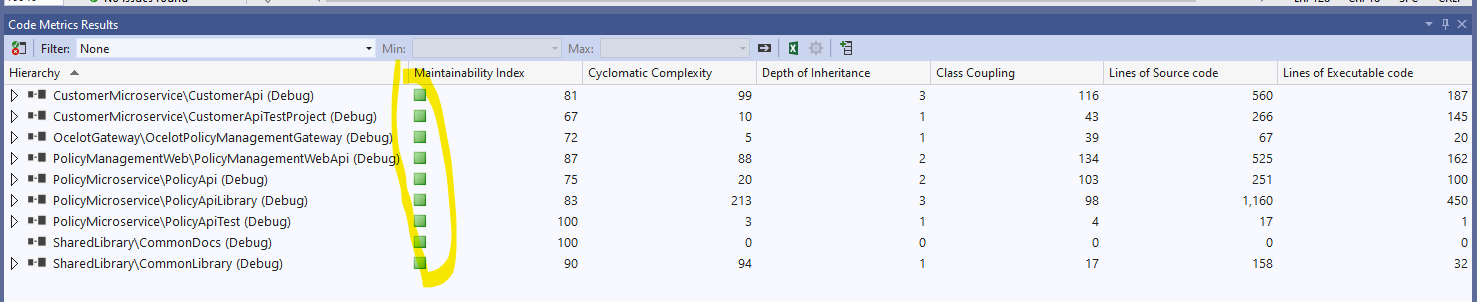
* Ocelot configure file defined below routes for common API Gateway.



* Used blow technologies to implement the project.

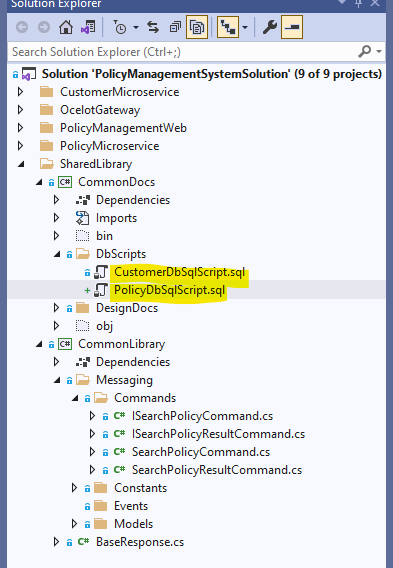
|  |
| --- |
| Asp.Net Code |
| Latest Ocelot package (17.0.1 support .Net Code 5.0) |
|  |
|  |

* 1. Keep the high quality code by using suitable design pattern, coding style and used Calculate Code Metrics tool to monitor code quality:

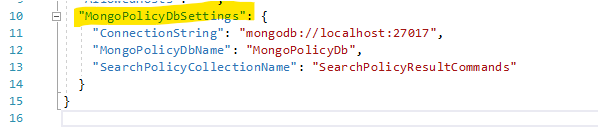


1. System Setup
   1. Database setup.

* Please find below 2 SQL server scripts from below folder and set up the CustomerDb and PolicyDb in MS SQL Server



* Please refer to below setting to set up the MongoDB local or in Docker:



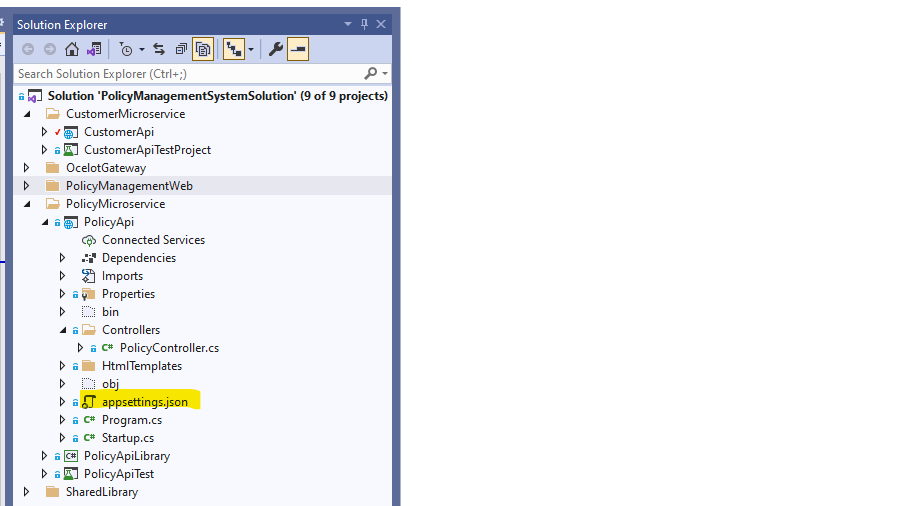
* 1. Rabbit MQ Docker setup

Please refer to below command to run Rabbit MQ from Docker:

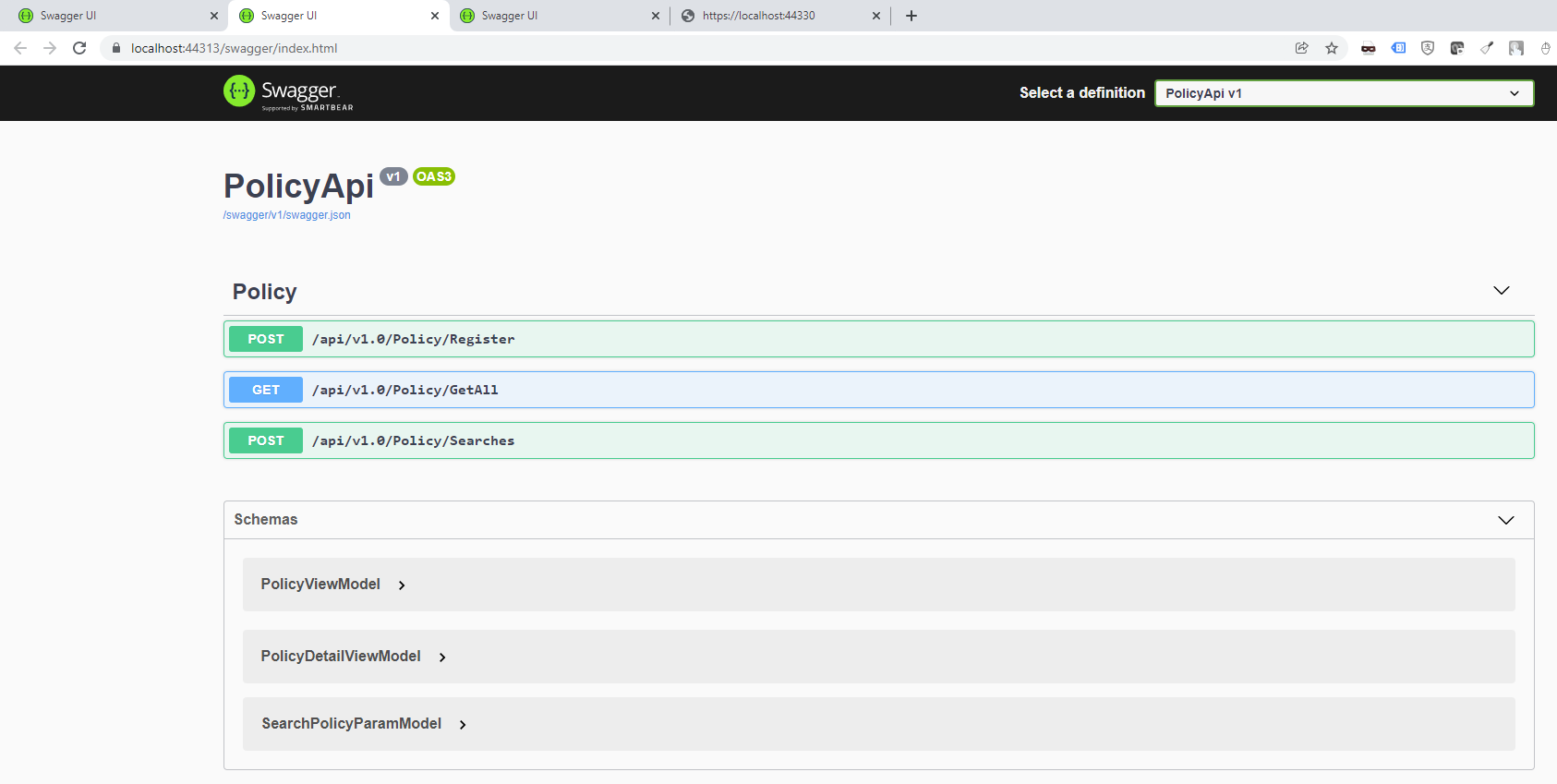
*docker run -d --name rabbitmq-policy -p 5672:5672 -p 15672:15672 rabbitmq:3-management*

* 1. Asp.Net Core Application setup

Please open the solution in Visual Studio 2019 and change all appsettings.json and Ocelot configuration file according your local settings, compile the code and run all services by Ctrl+F5:



Shall see the blow 3 Swagger pages and one Ocelot page opened:



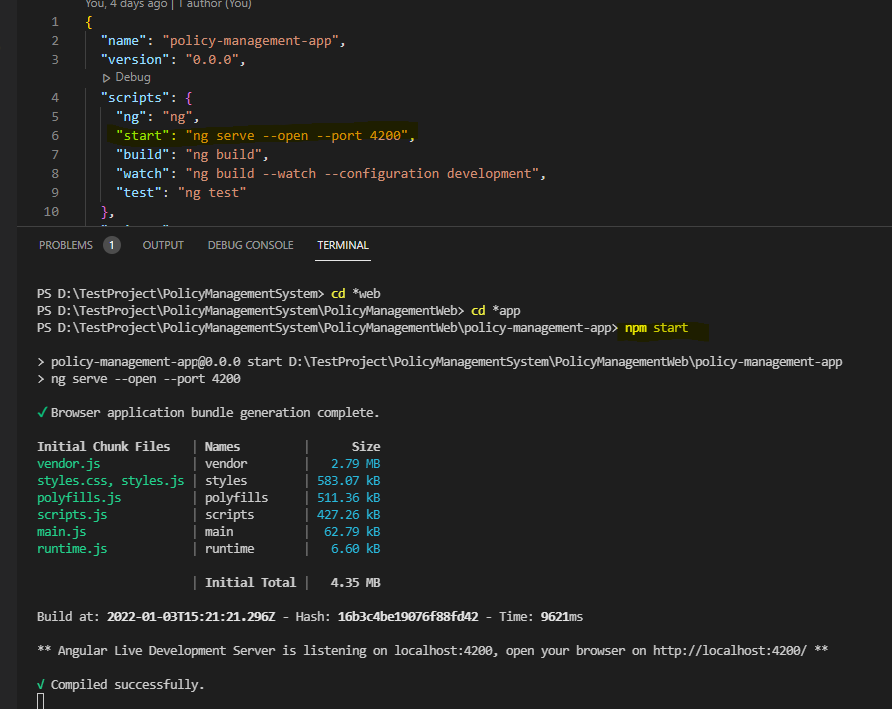
* 1. Set up Angular app:
* Please use Visual Studio Code to open the policy-management-app under below folder in solution:

*PolicyManagementSystem\PolicyManagementWeb\policy-management-app*

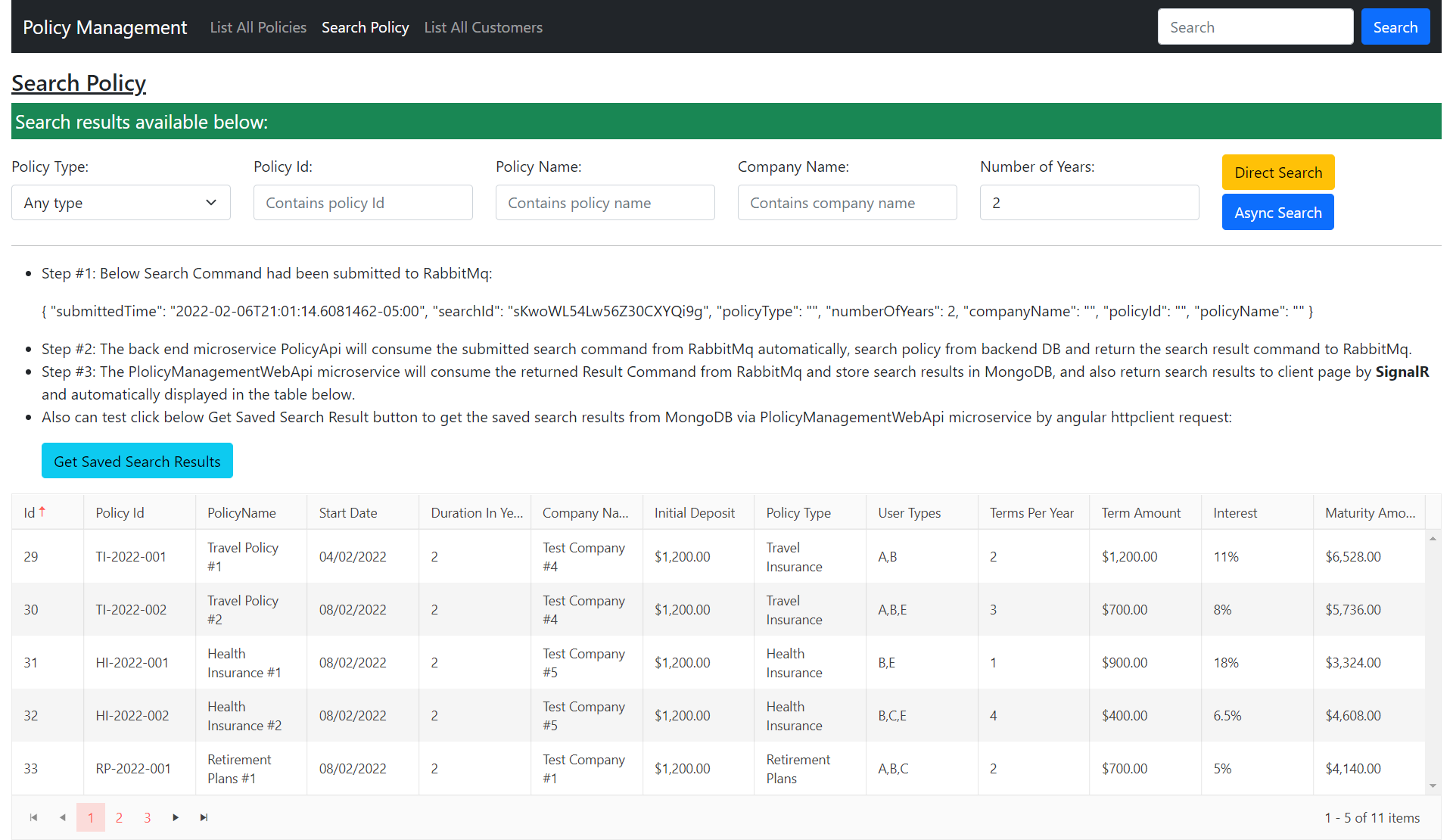
* Update the Ocelot base url from below service file:



* Run npm install to install all dependencies modules and run npm start to open the angular app:

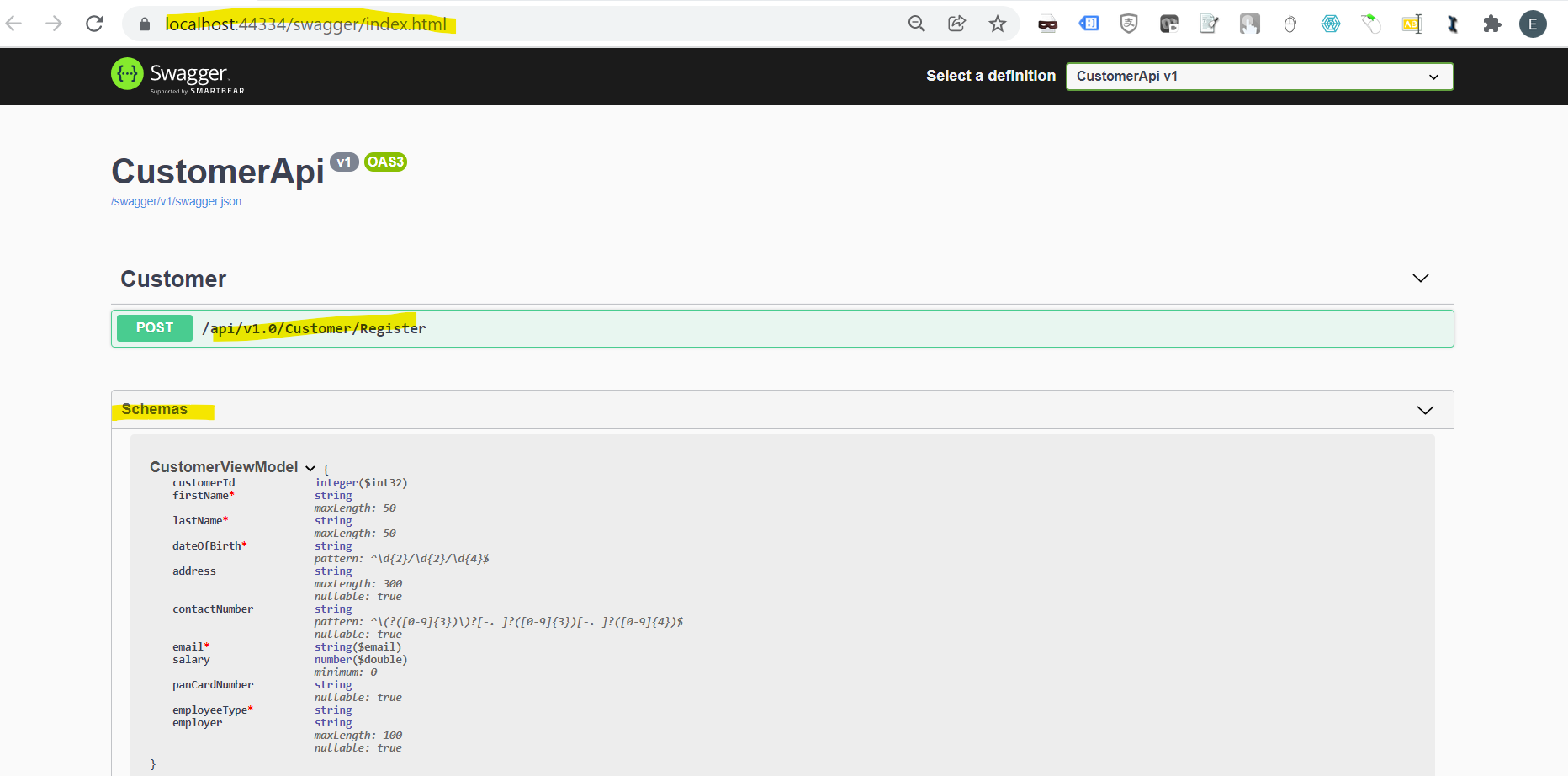


Shall see Angular app as below:

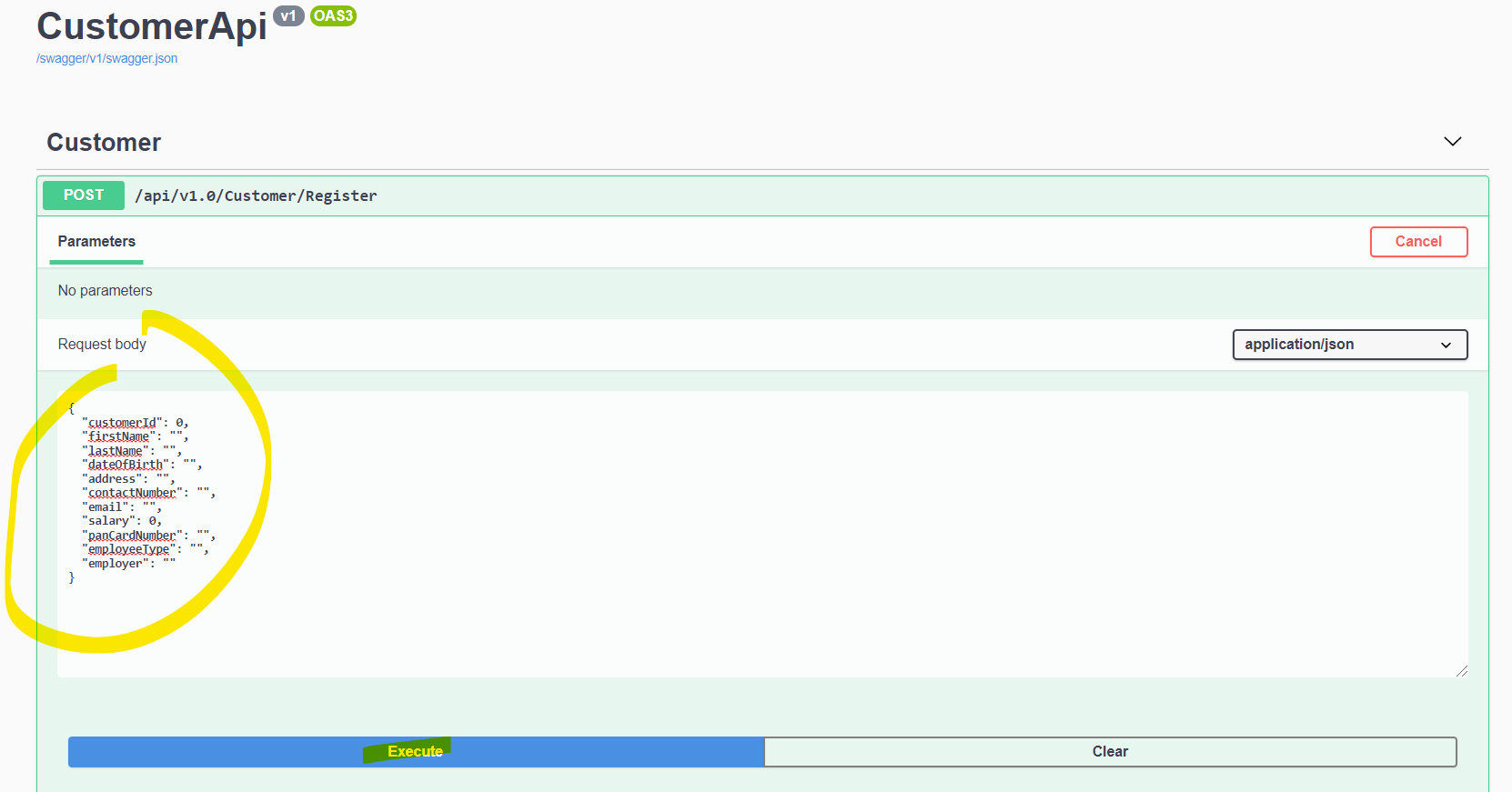


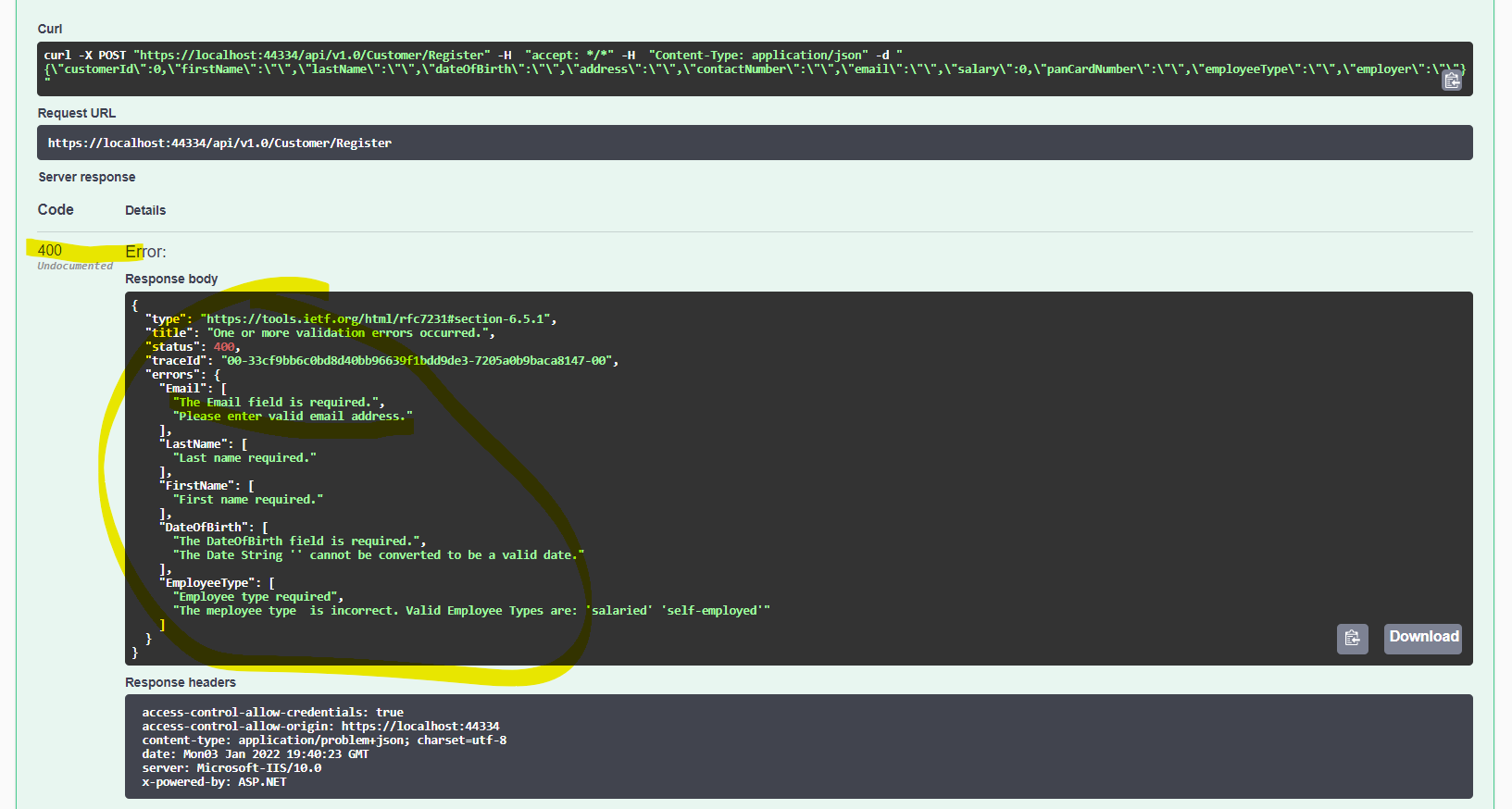
1. System Integration Testing
   1. Customer API Micro Service Testing

* Can see the API Register features and schema from Swagger blow:

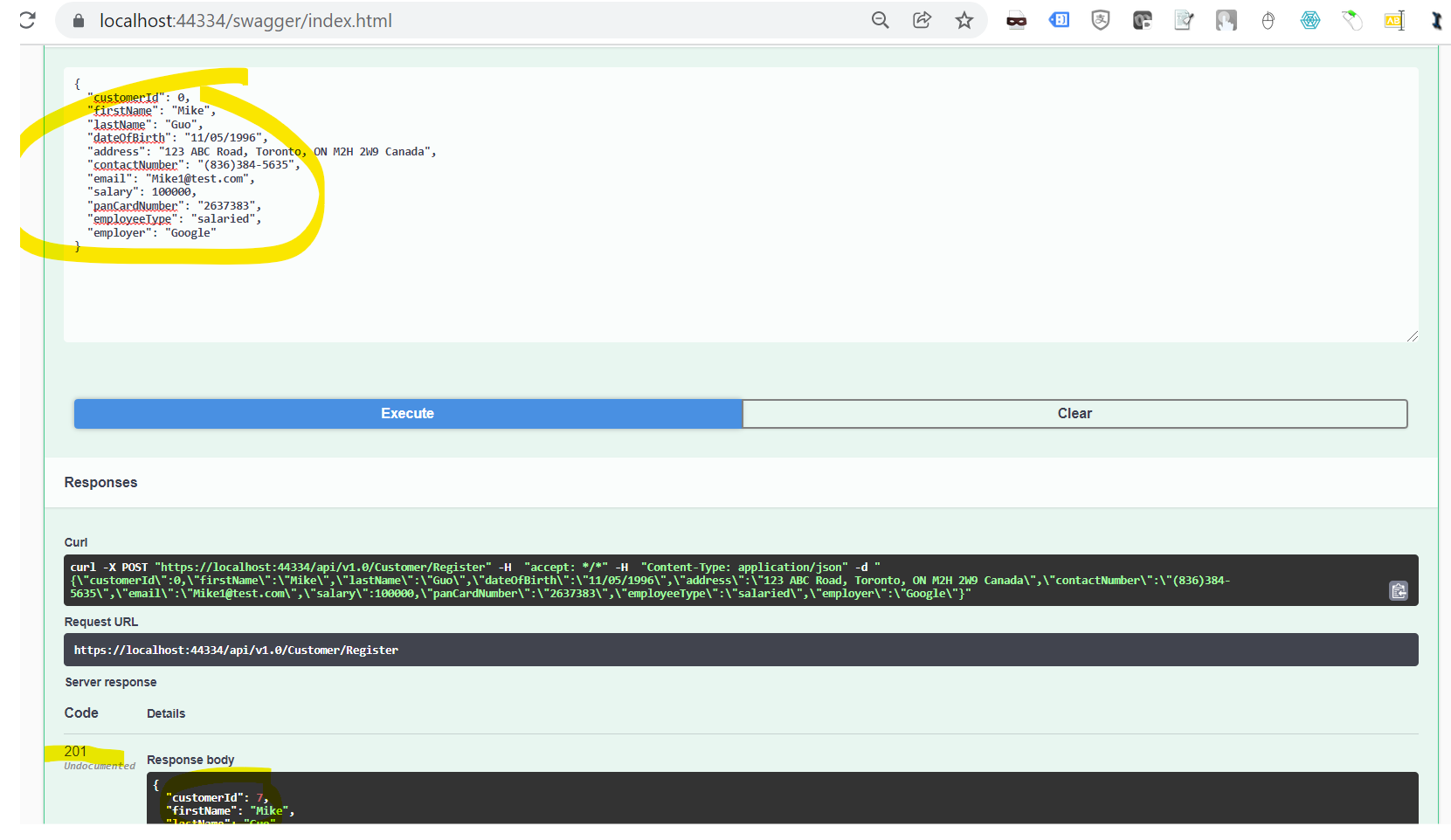


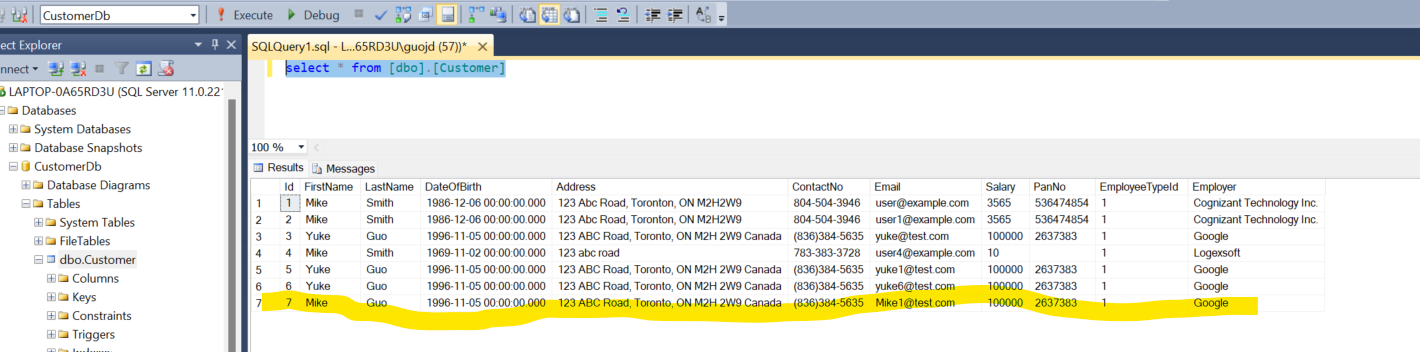
* Test from Swagger:
  1. Submit invalid customer data and see rejected with validation information



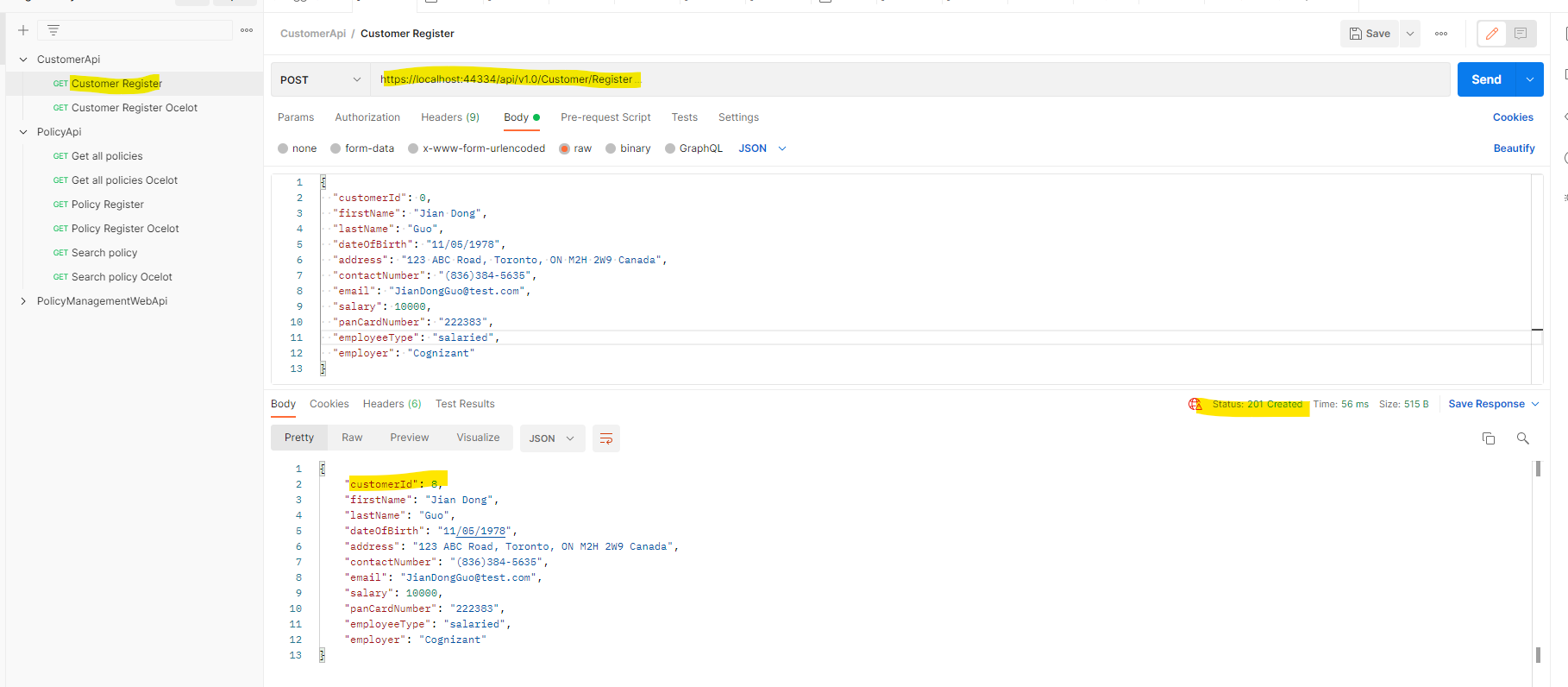


* 1. Submitted the valid customer data and see success feedback and check DB success too:

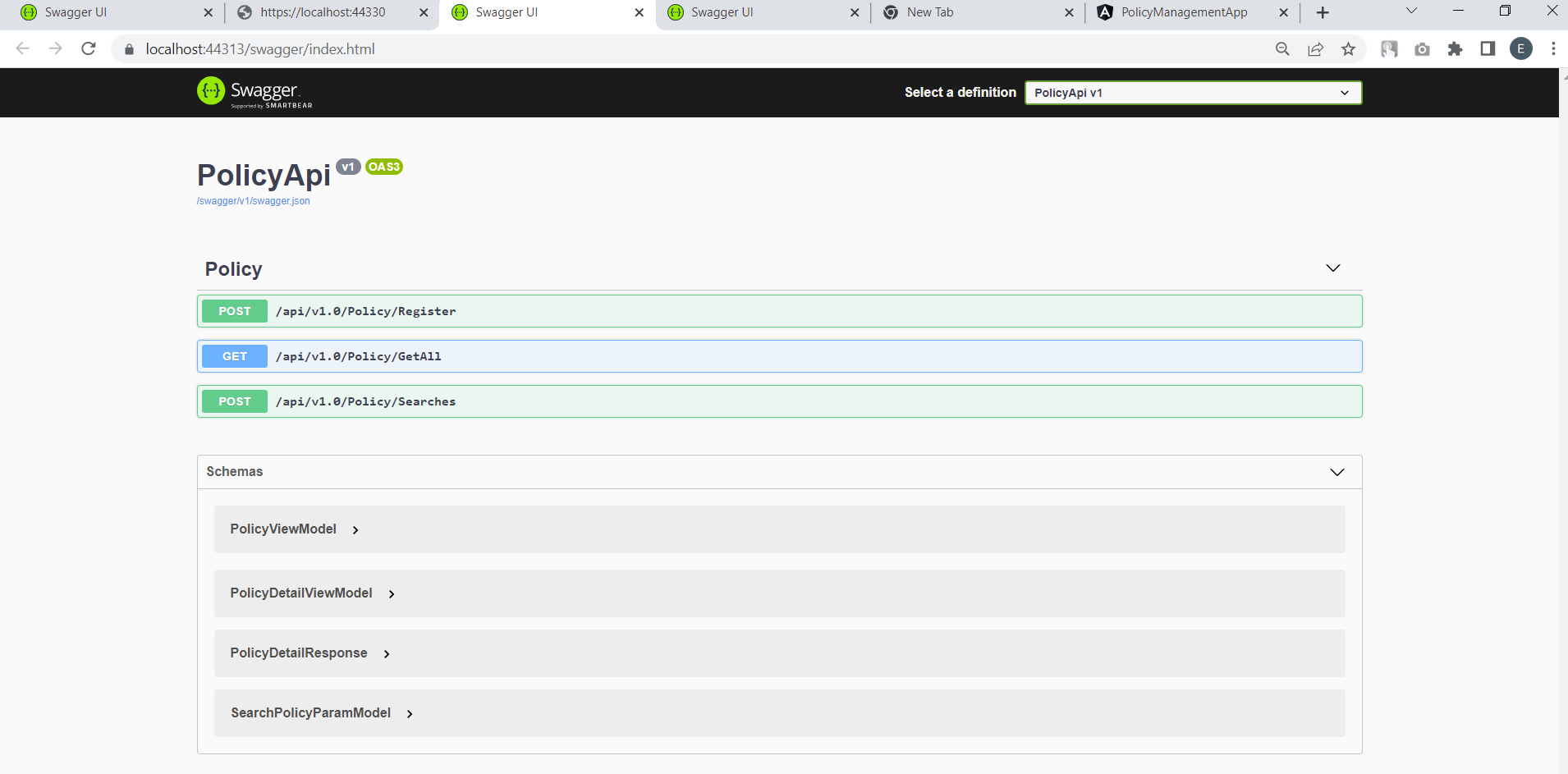




* Test from Postman:



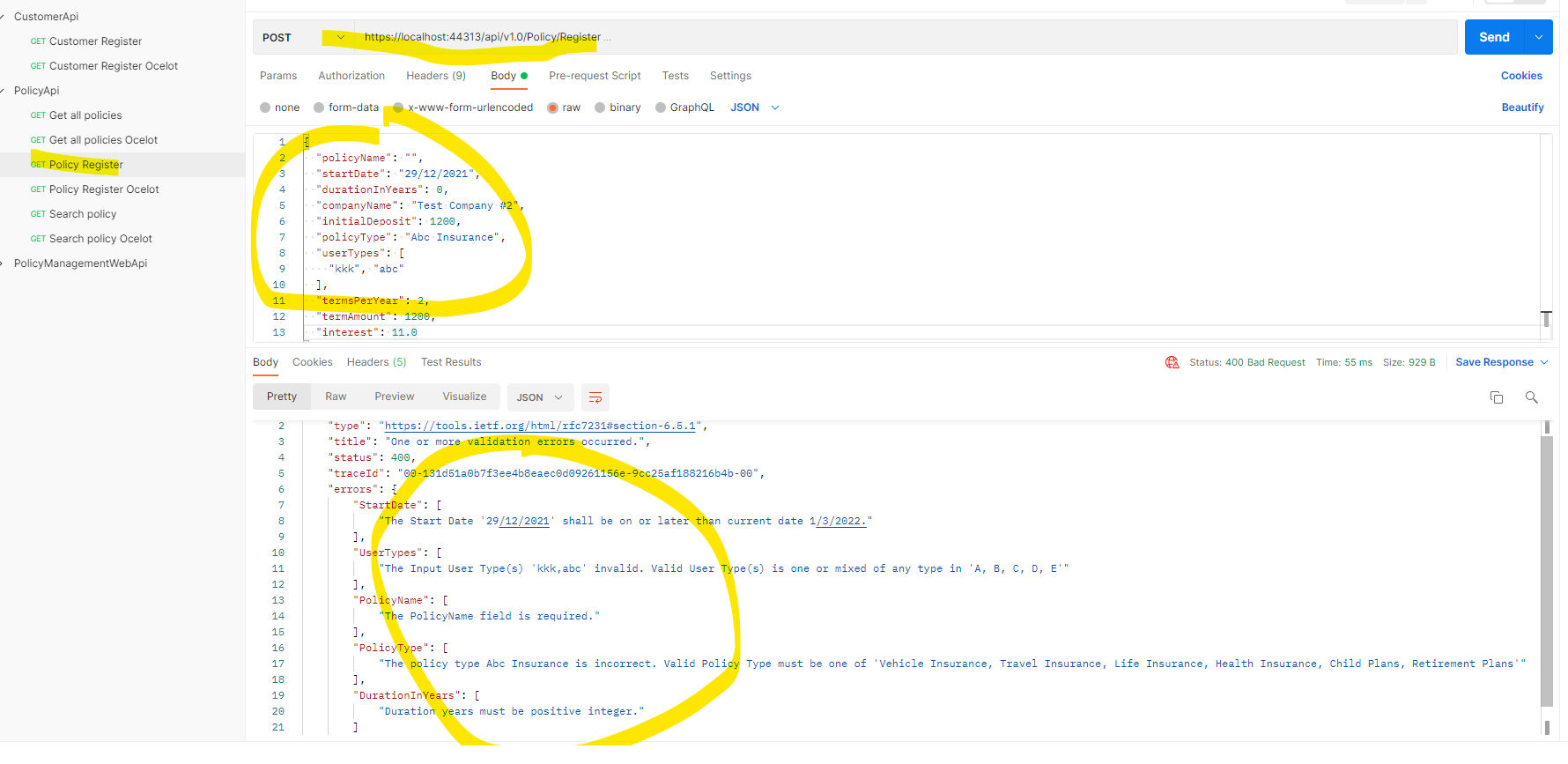
* 1. Policy API Microservice Testing
* Can find the API features and Schema from below swagger page:



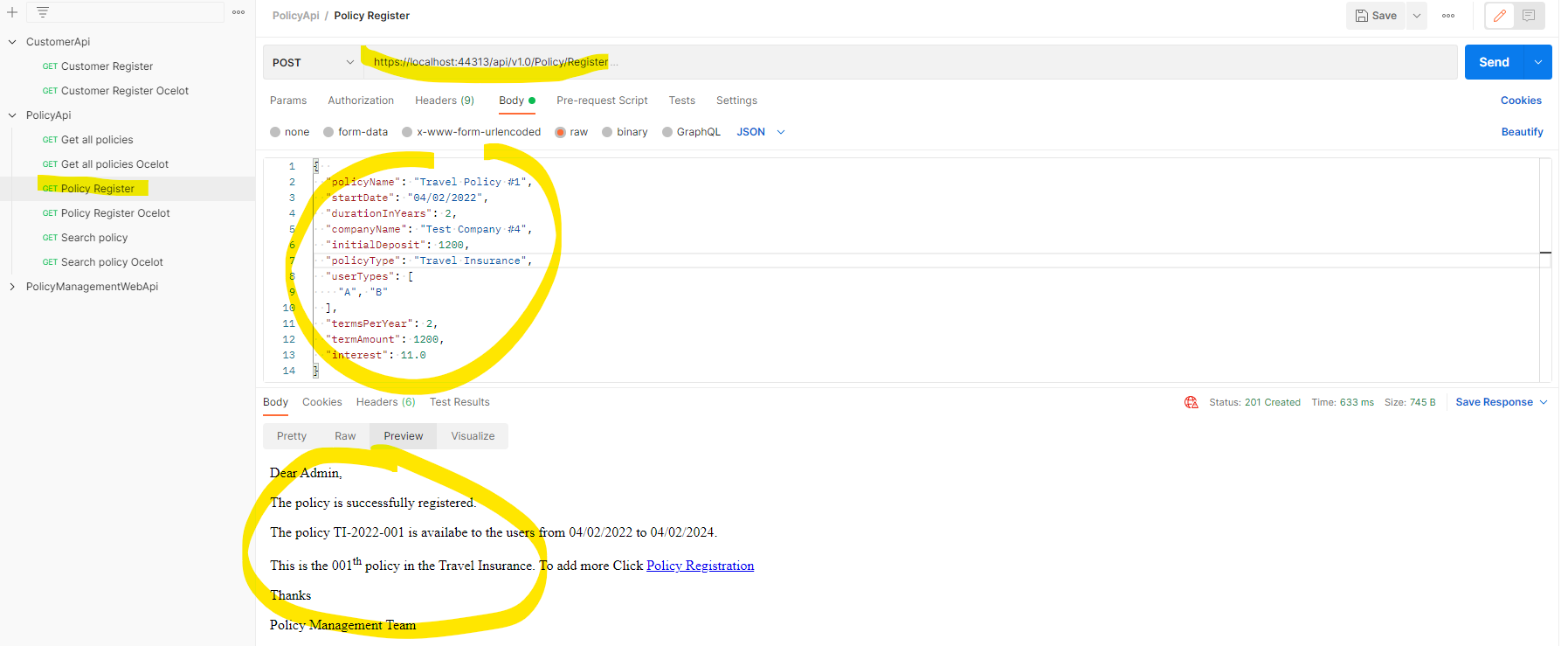
* Test on Swagger:

Omitted…

* Test on Postman:
  1. Submit invalid data and get validation message:

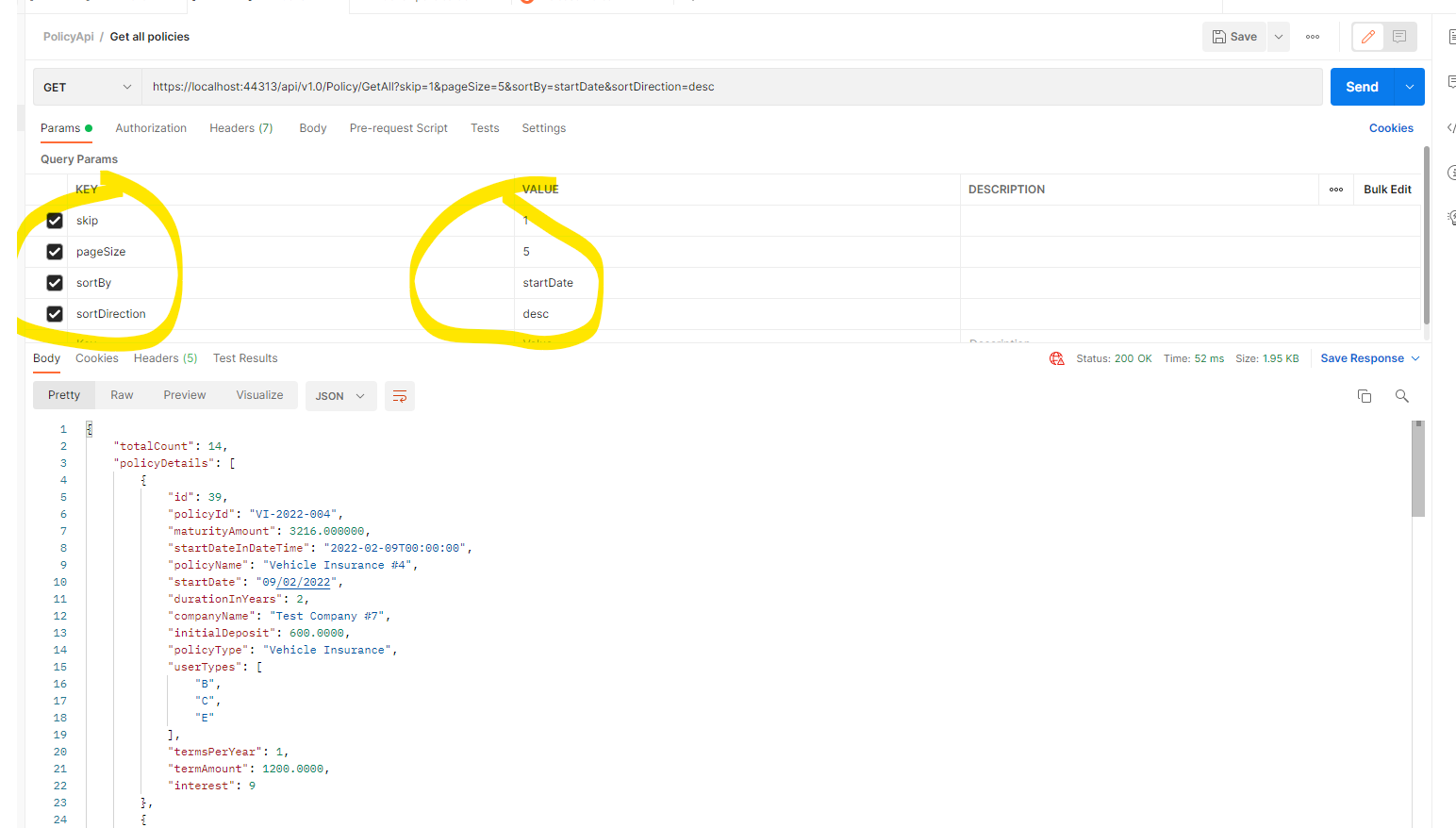


* 1. Submit valid data and get success message in html format and validate data in DB:

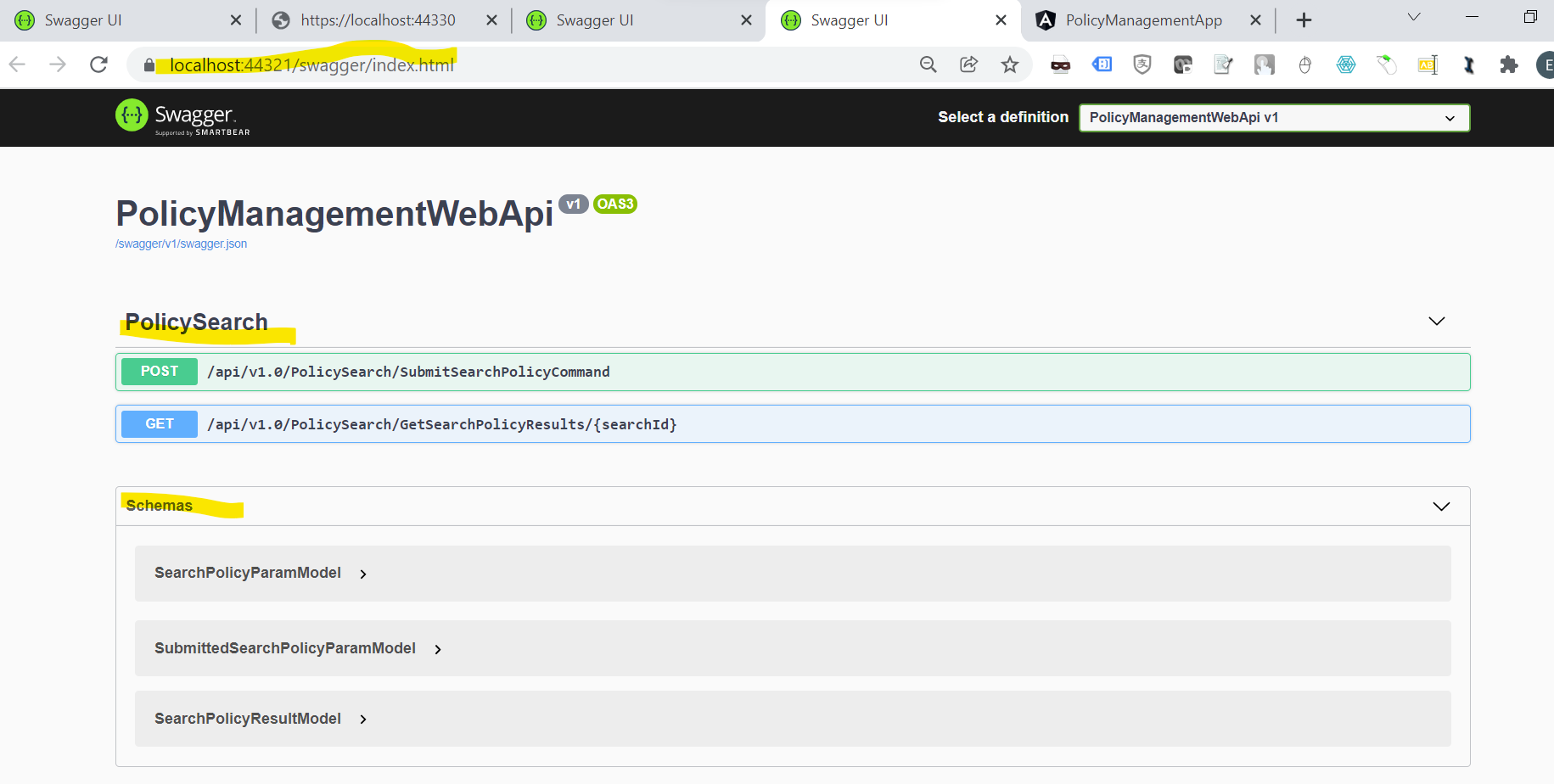




* 1. Test the Server side Pagination and sorting features in GetAll api:



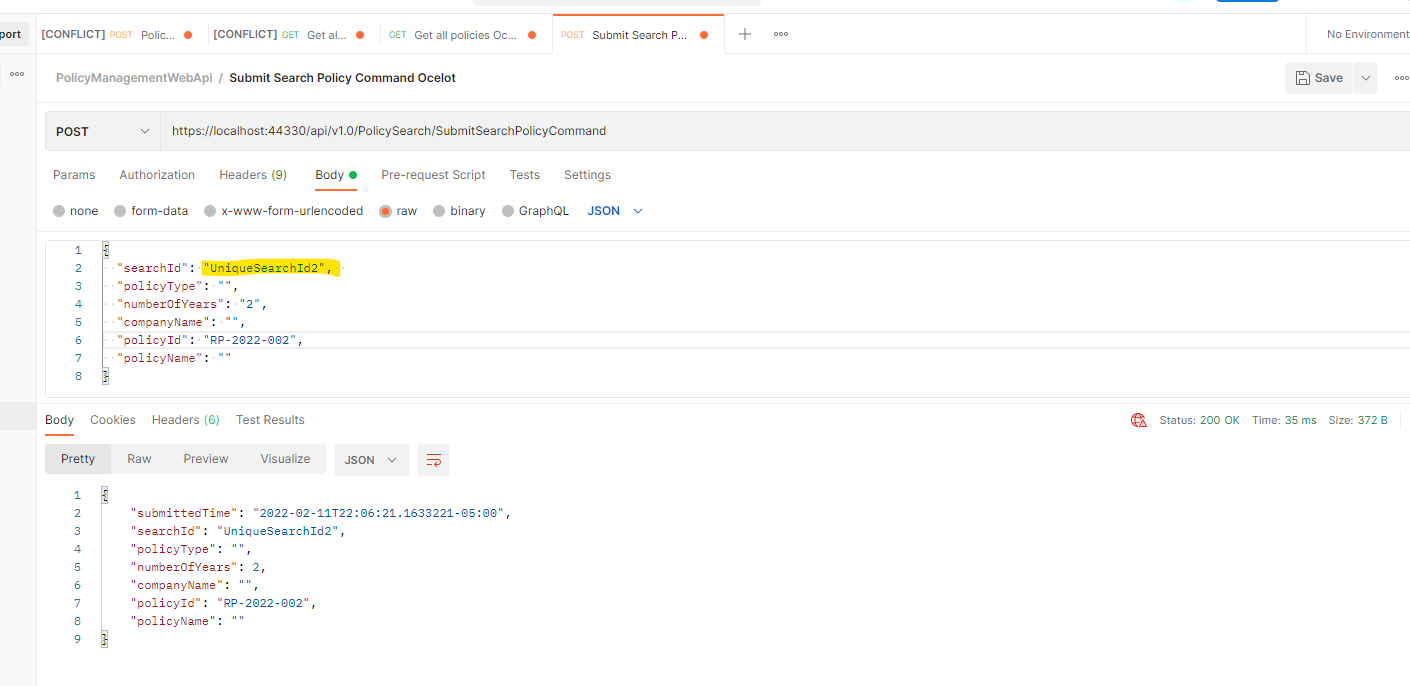
* 1. Policy Management Web API Testing
* Can find the API features and schemas from below Swagger page:



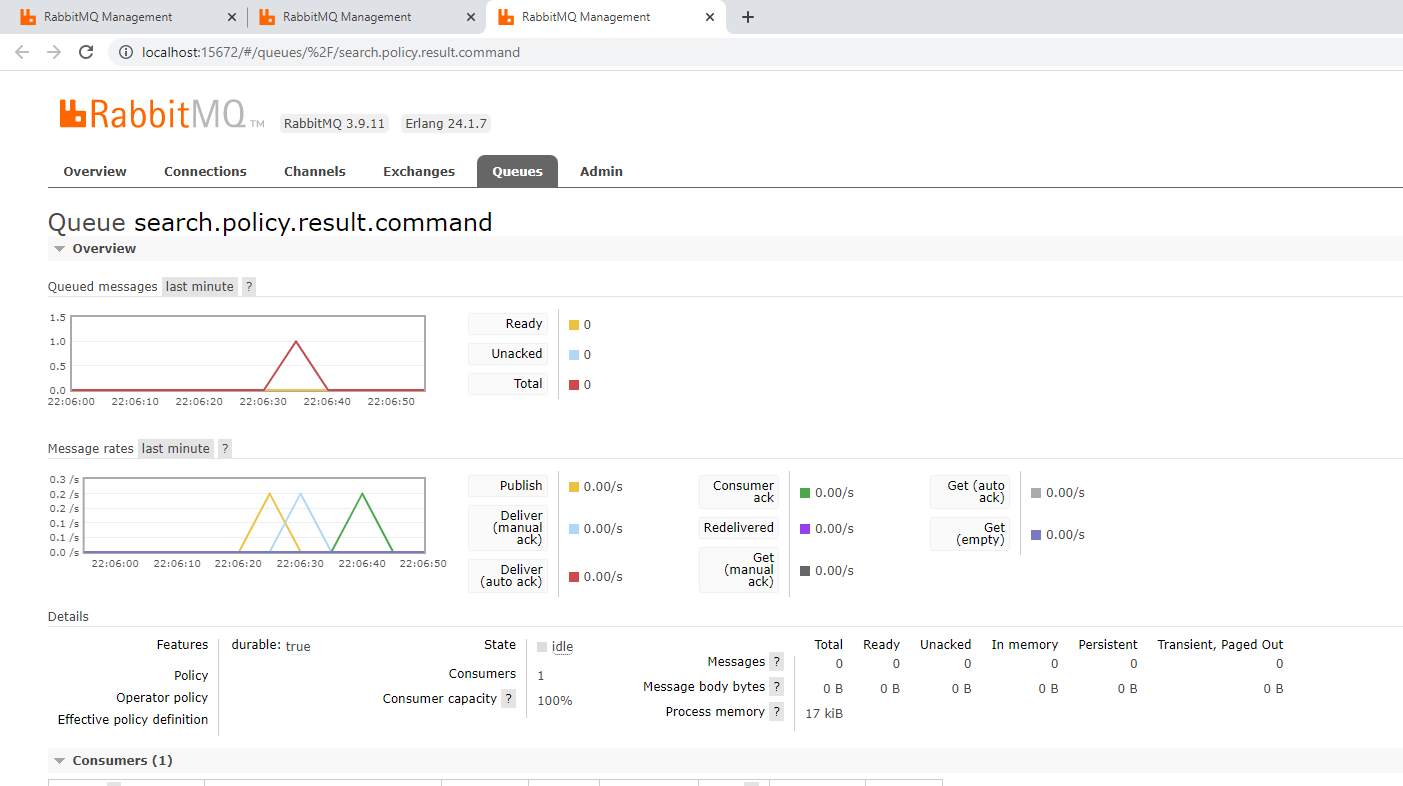
* Test API from Swagger page:

Omitted…

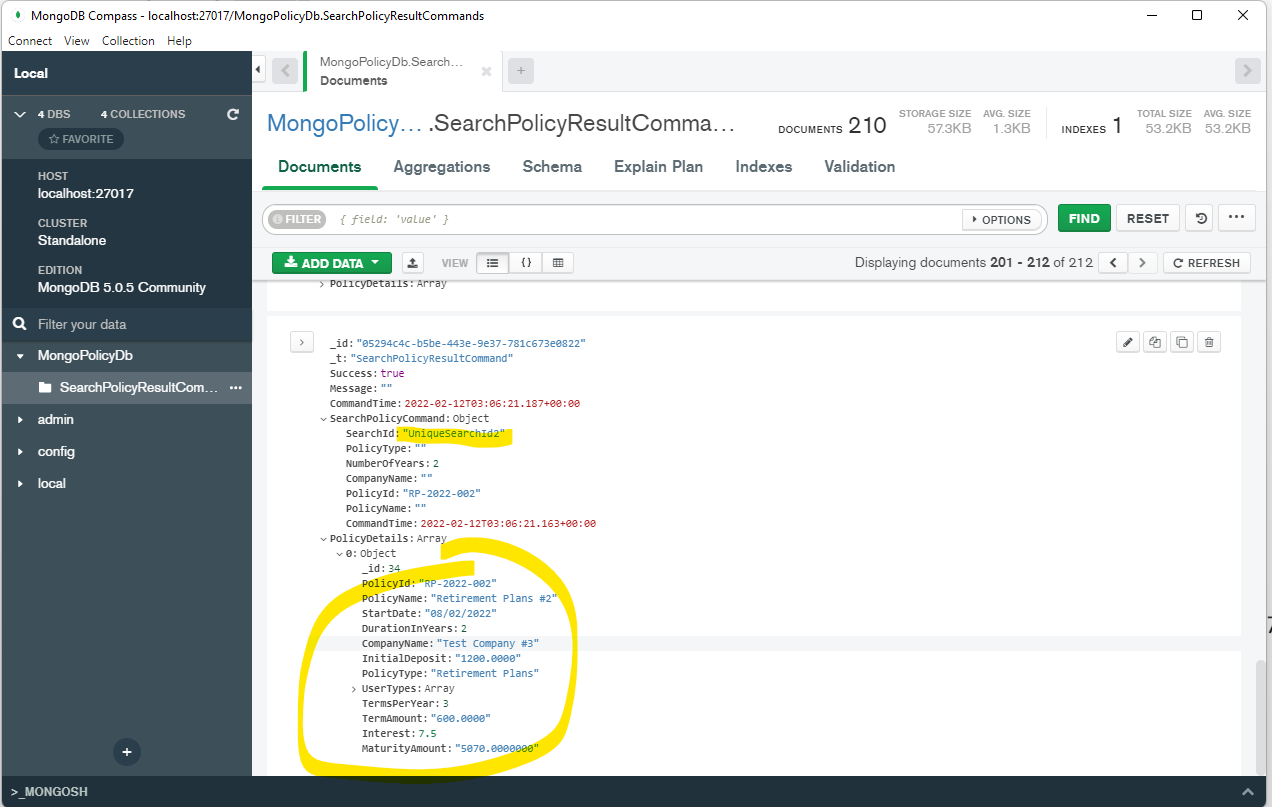
* Test API from Postman:
  1. Submit search policy parameters command to Event Bus (Rabbit MQ) with mocked unique Search Id “UnqiueSearchId2””. The back end Policy API will consume the search policy command and return the search result command to Event Bus, and then PolicyManagementWebApi will consume the search results from Event bus and store data in MongoDB):



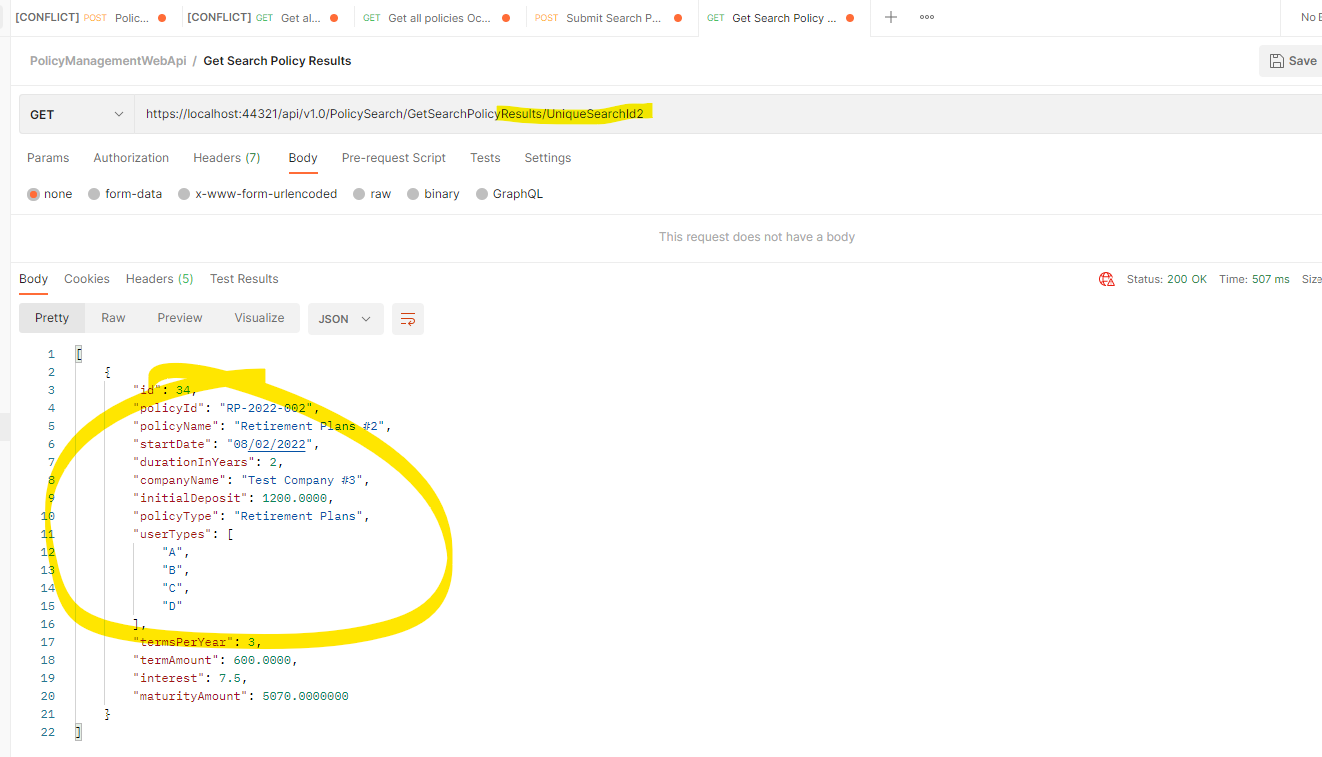
Rabbit MQ process the messages:



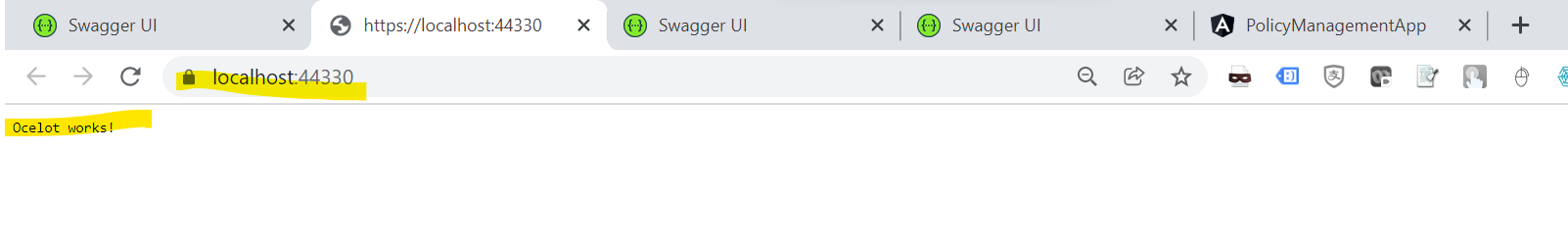
Search results are stored in MongoDB:



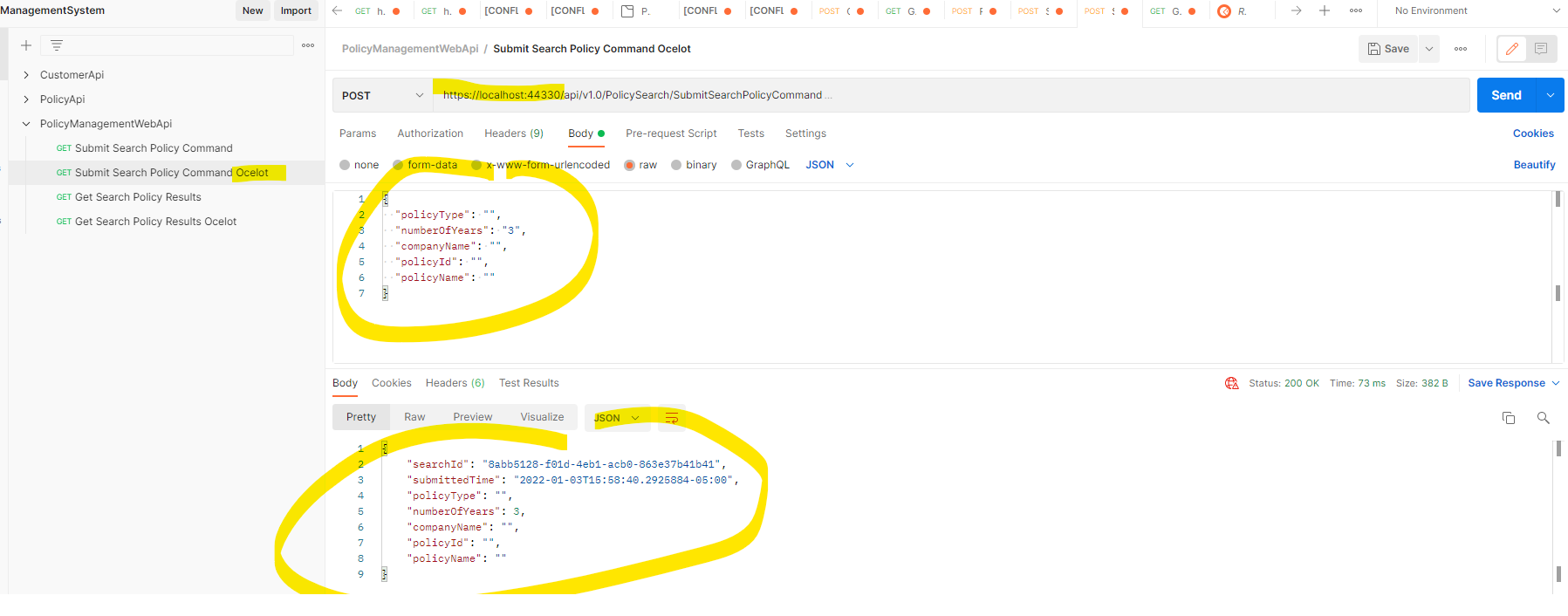
* 1. Get search policy result from stored data in MongoDB by above unique Search Id “UniqueSearchId2”:



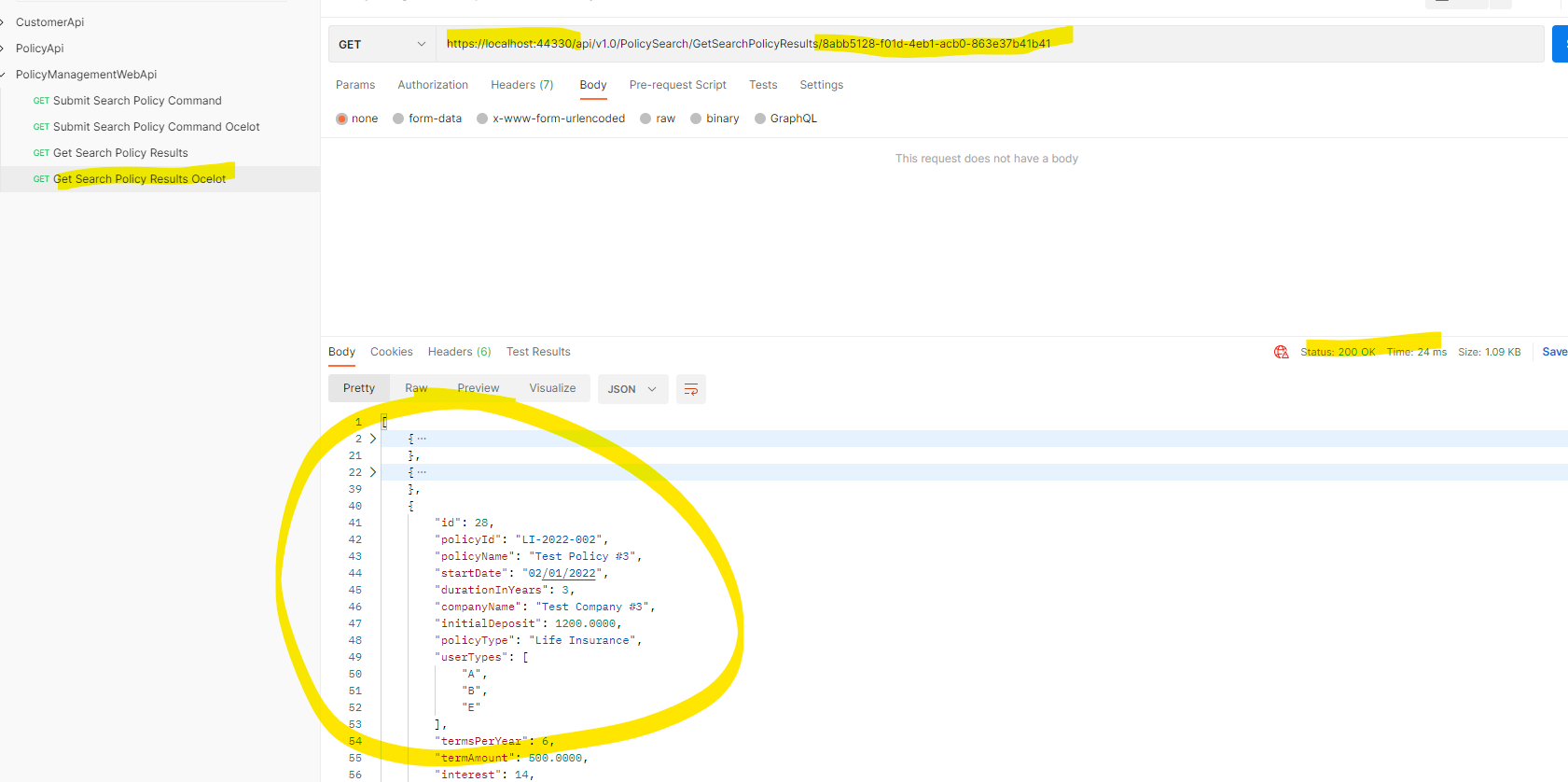
* 1. Ocelot Gateway Testing
* Can validate the app work from below link:



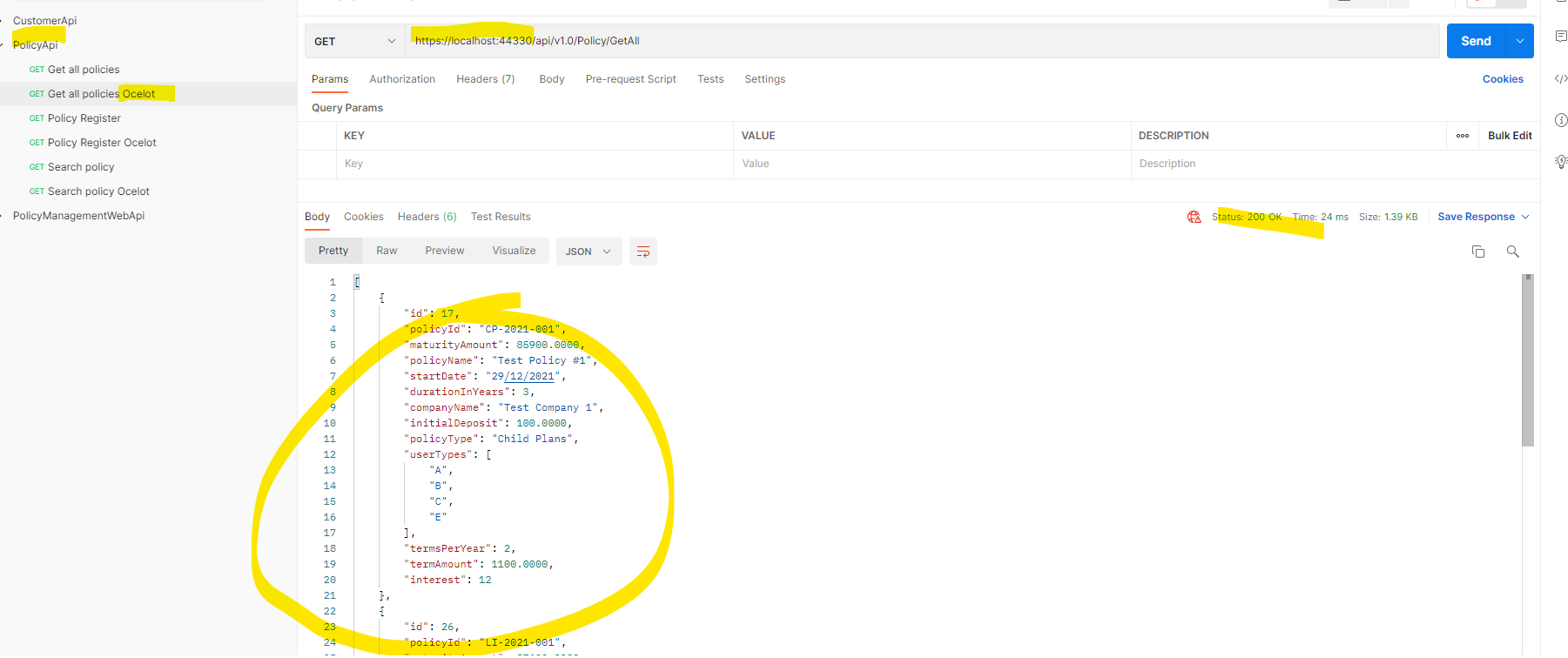
* Test common gateway from Postman:
  1. Submit the Search Policy command, get the unique Search Id:



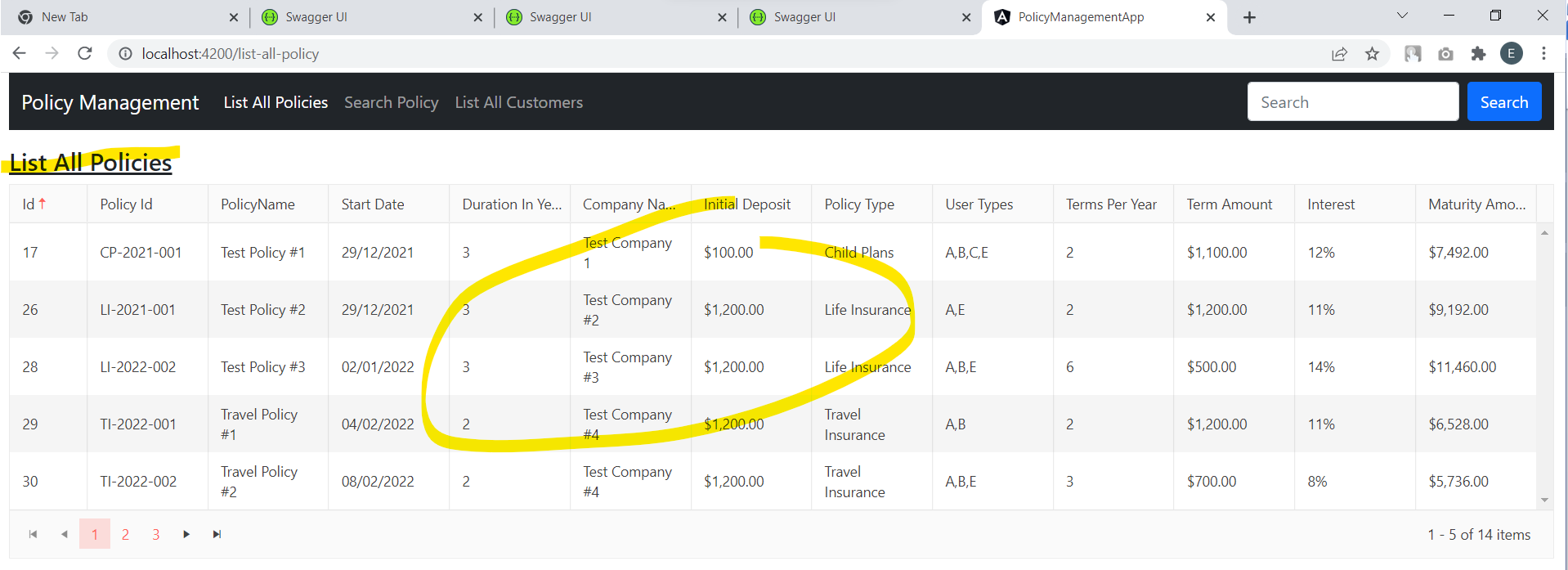
Retrieve the Search Results by the Search Id:



* 1. Test the Policy API GetAll feature from common gateway:



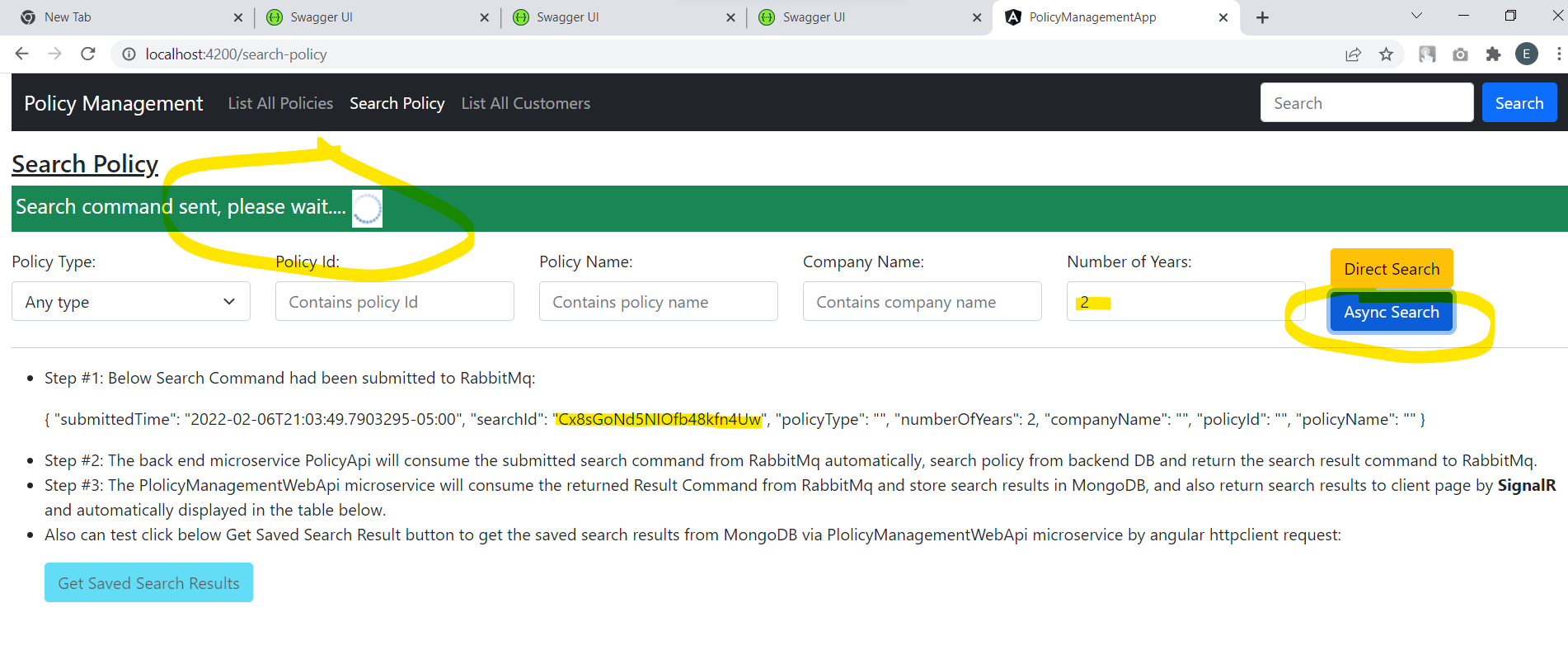
* 1. Angular app integration testing
* Load angular app as below, the default page is List All Polices page:



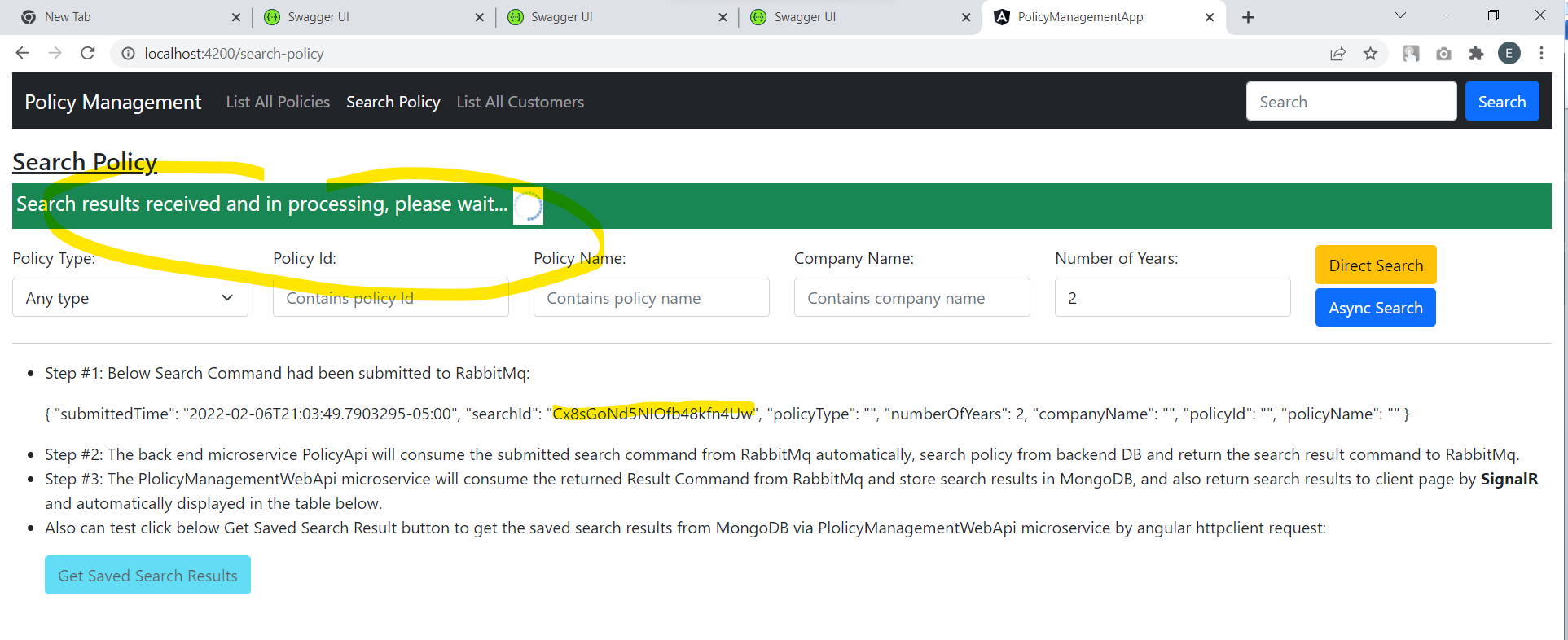
* Test Async Search policy via Event bus (Rabbit MQ) + SignalR:

SignalR will keep the persistent connection with Client angular app and keep update below status and data automatically. For each async search, the Connection ID (search ID below) is unique, so each search notification will not interact with other users’ search:

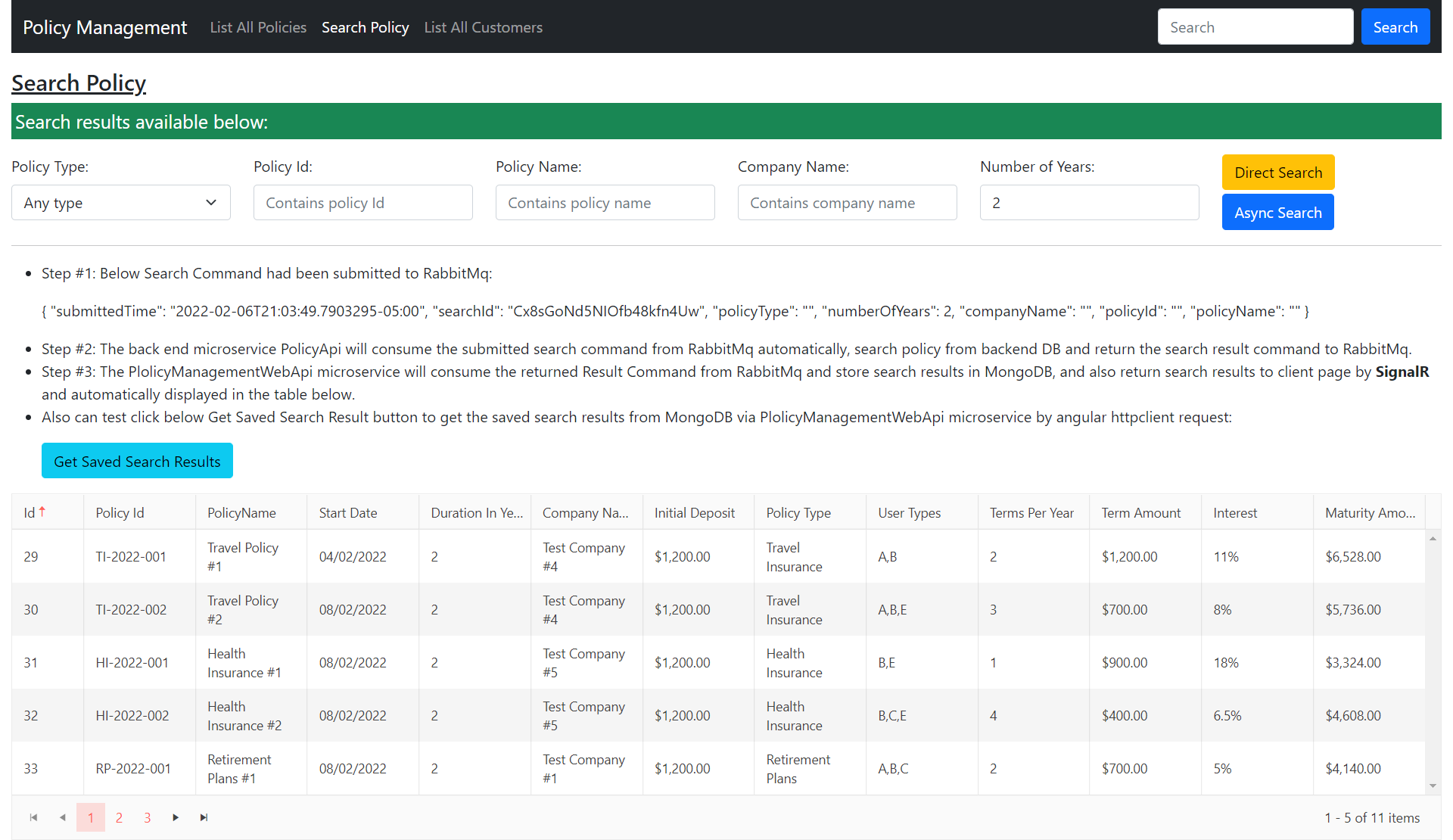
1. Submit Async Search, the message will show as below:



1. Backend Policy API consumes the Async search command and sends the search results back to Policy Management API vis Rabbit MQ Event Bus, Ploicy Management API will consume the search results command from Rabbit MQ and store data into MongoDB for future Auditing etc. Client app get below status notice automatically:

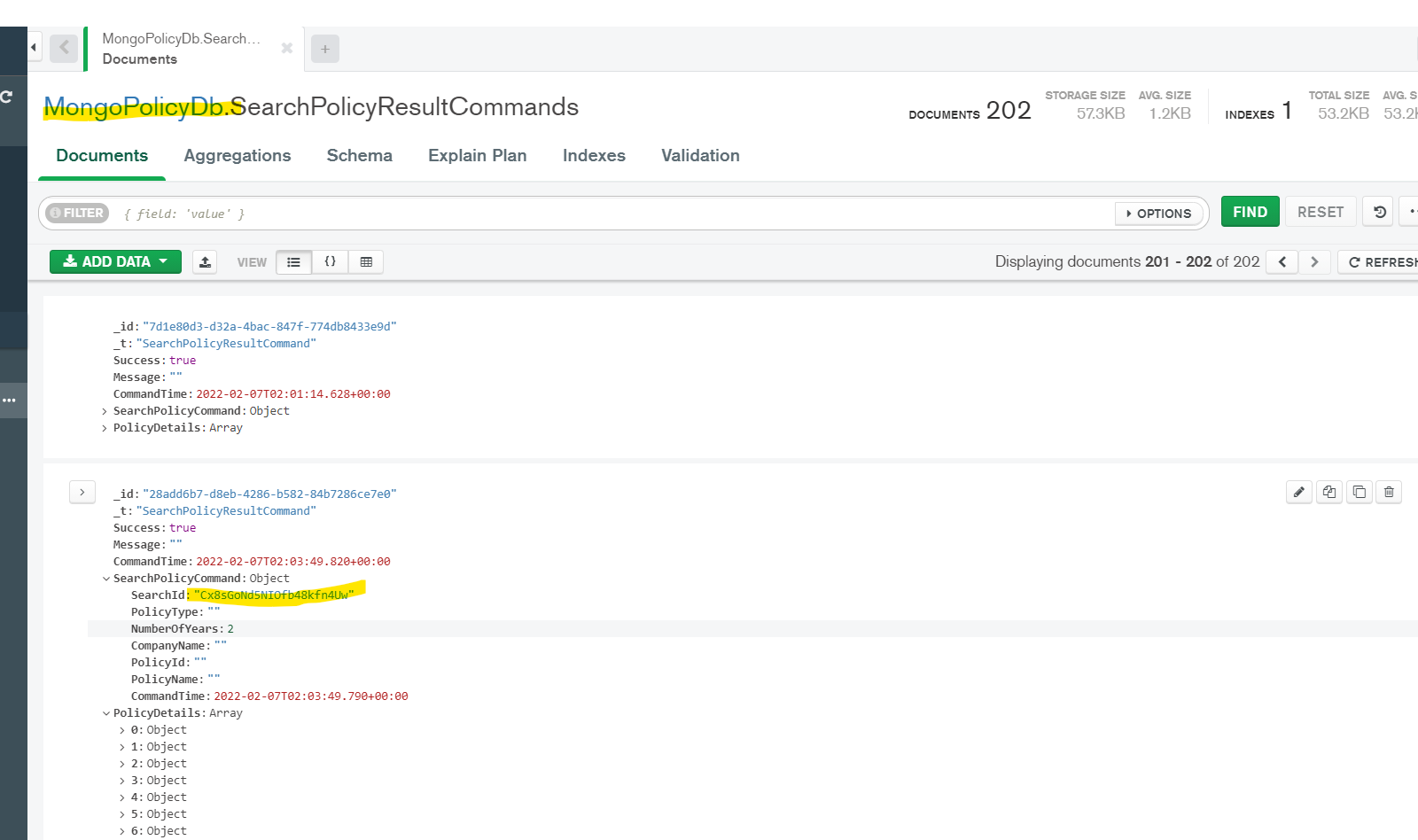


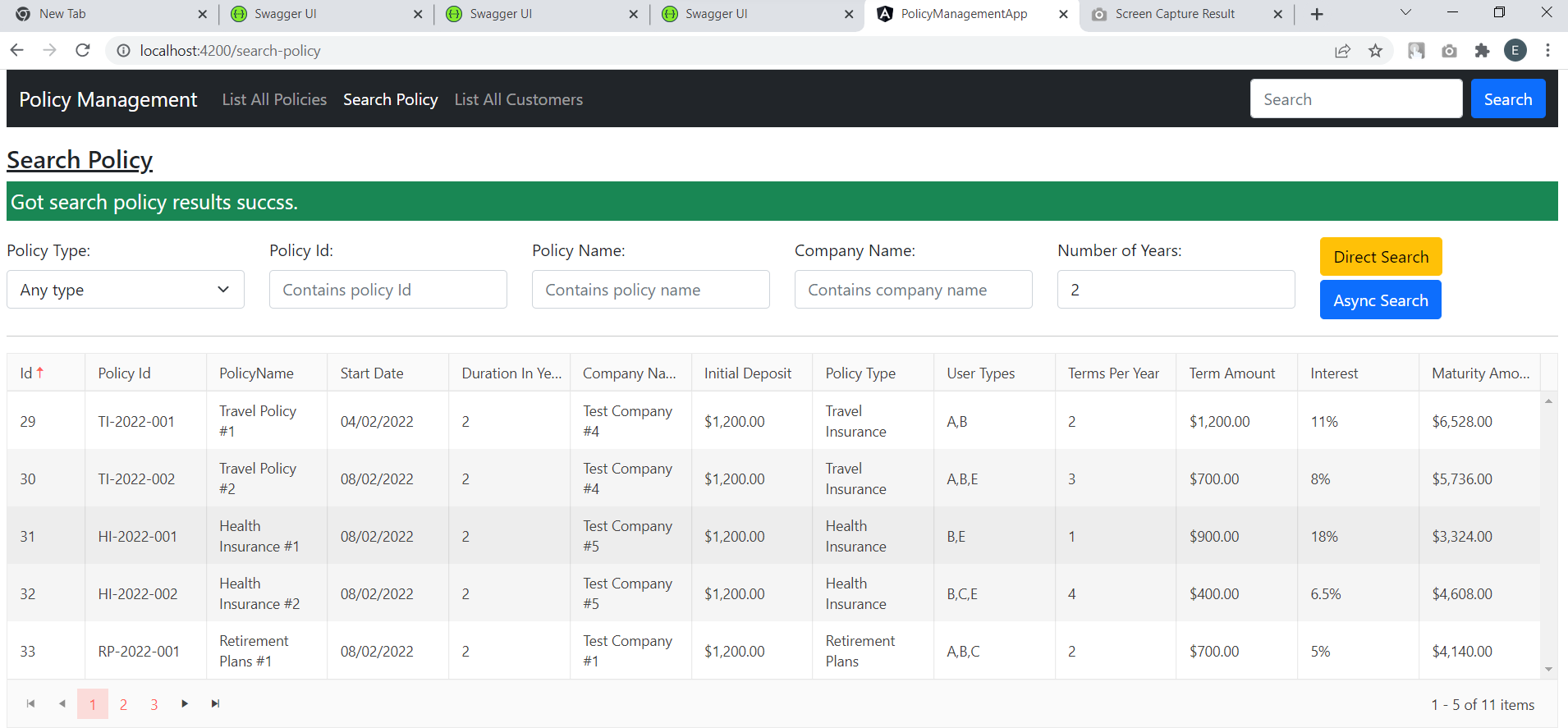
1. After saving to MongoDB, Client app can get searchs results back from SignalR event and display the results in page automcatically and the “Get Saved Search Results” button becomes enable too:



Notes: In above 3 steps, the Connection ID (Search ID) “Cx8sGoNd5NIOfb48kfn4Uw” is unique for each Async Search event.

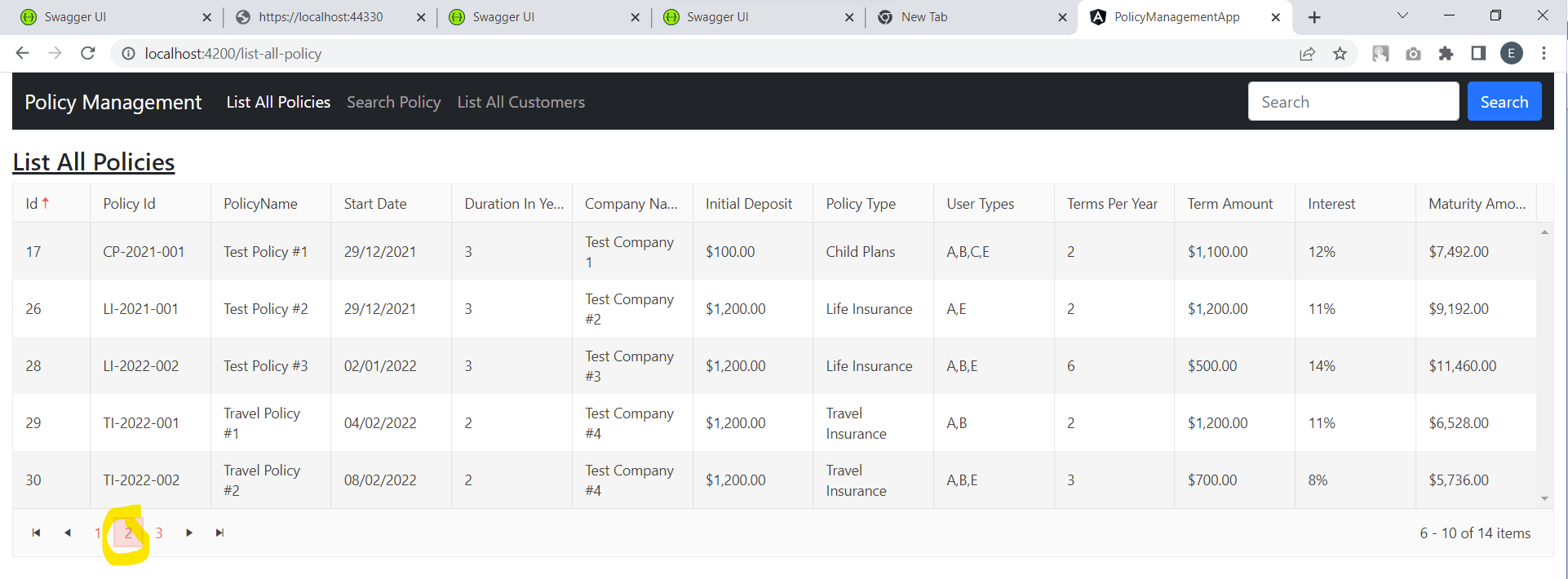
1. Since search results are stored in MongoDB as below, user can also click the “Get Saved Search Results” button to call Policy Management API to query MongoDB and load the search results by the Unique Connection ID (Search ID):



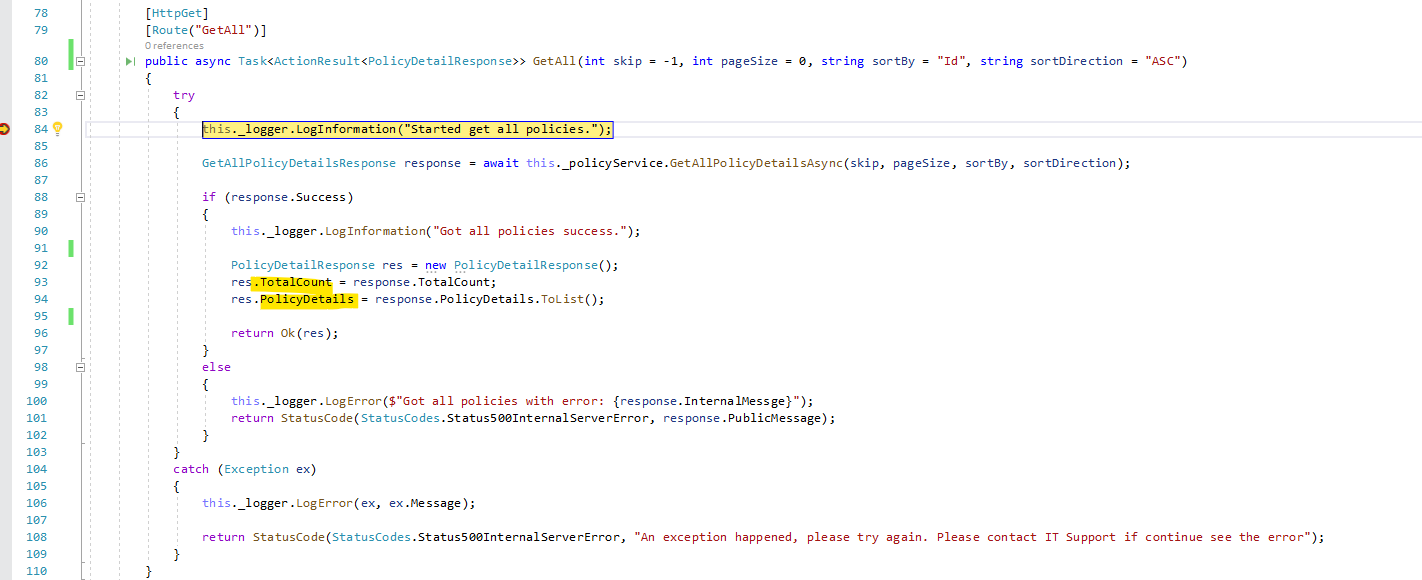


* Test list all policy direct feature in List All Policies page, this page implement the Server side pagination and sorting features:

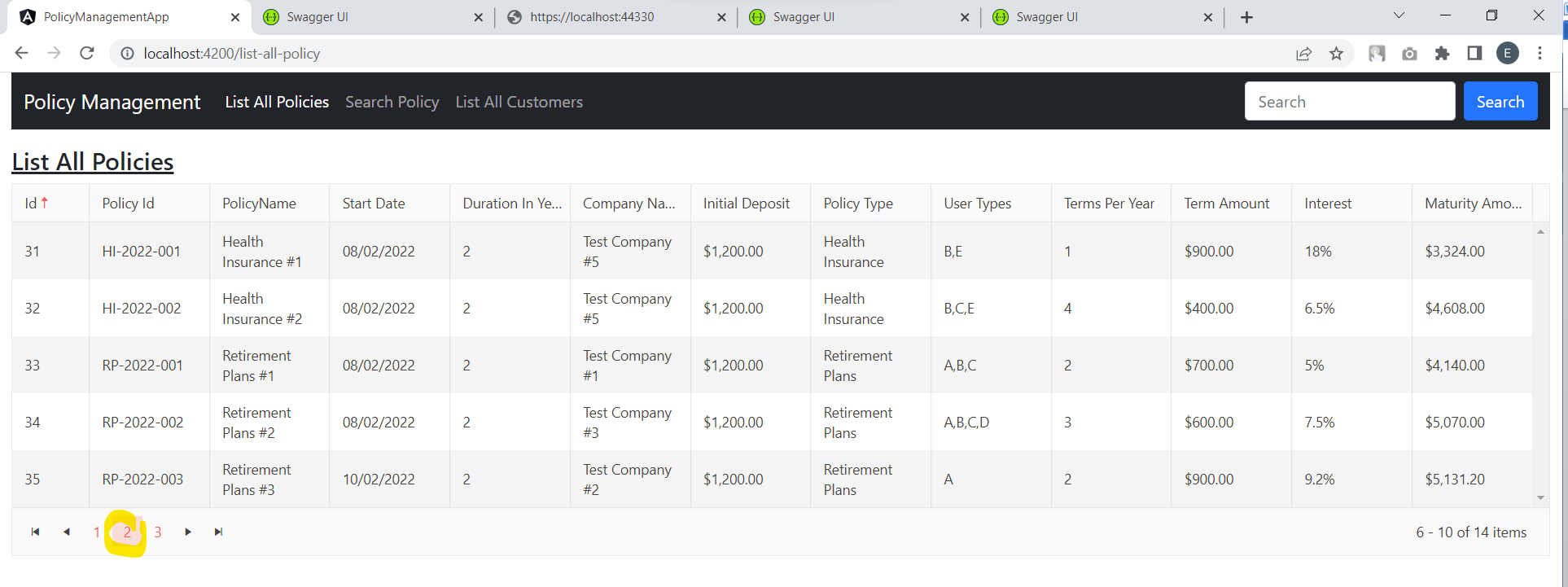
1. Click on page #2:



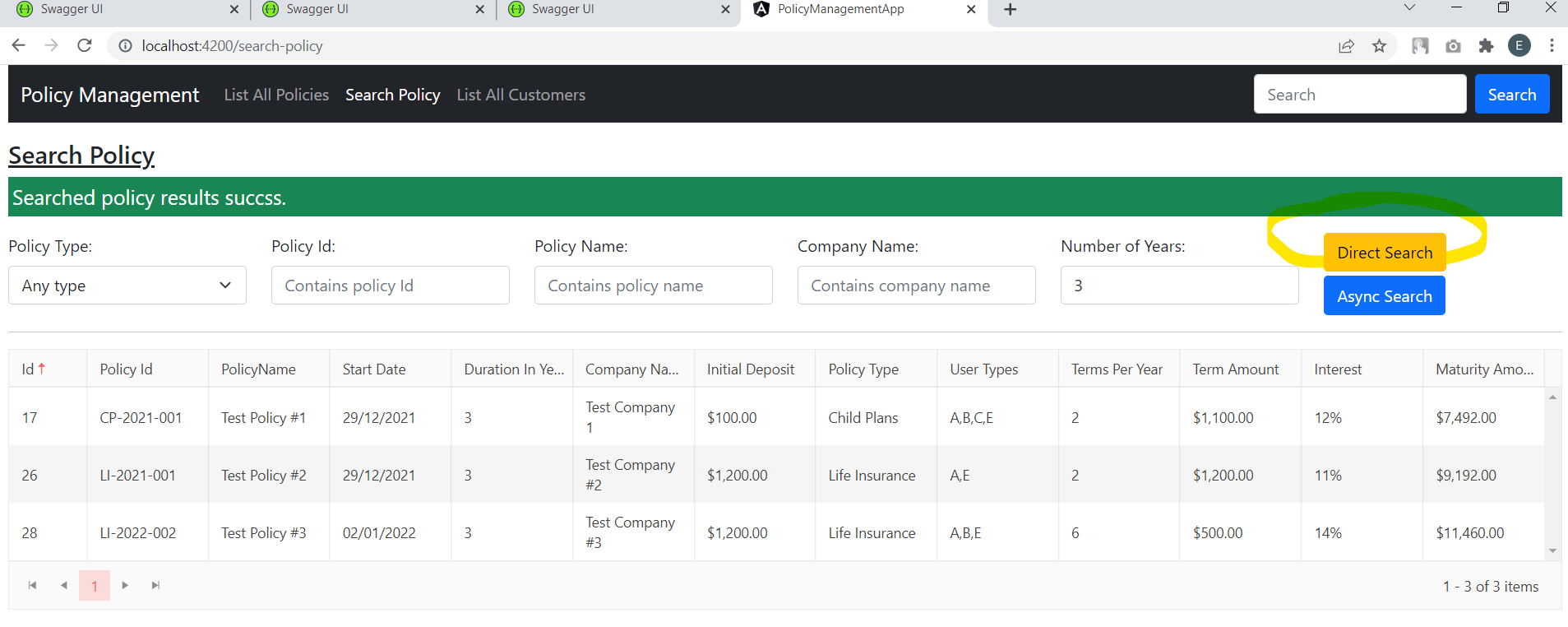
1. The backend Policy API will do server side pagination and sorting and return data including current page view and total record count:



1. Kendo Angular UI Grid will render the page#2 data:



* Test search policy directly feature by calling Policy API service directly from Client app:



1. Comments and improvements

Since time limit, may add below features in the future:

* Add the authentication and authorization features, for example use the JWT technology to implement those features.
* Dockerize all microservices, angular app, Sql Server DB and MongoDB.
* Add more Unit testing to cover more code.