تمرین معماری سرویس گرا - درس مبانی فناوری اطلاعات
نرگس سادات اسدی خوانساری - 810101582

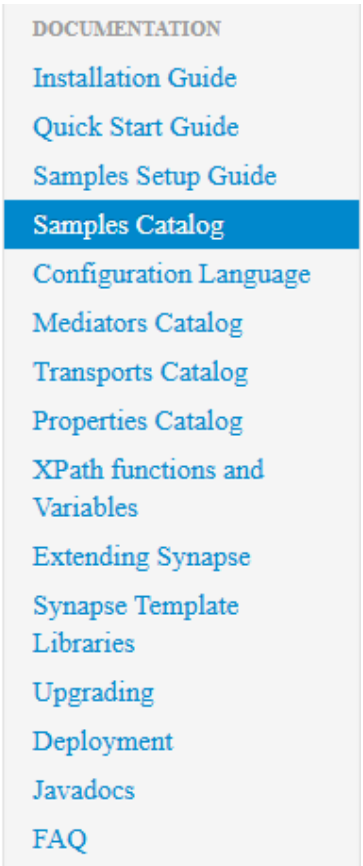
مرحله اول

- **معرفی ESB های مختلف:**

منبع: برای اطلاعات بیشتر و دریافت ESB های متنوع تر به ادرس
https://startupstash.com/enterprise-service-bus-esb-tools مراجعه کنید.

Apache Synapse-1: یک ESB سبک و با کارایی بالا است که از JMS، REST، XML/SOAP و پروتکل‌های دیگری مانند VFS، FTP، HTTP و TCP پشتیبانی می‌کند. این سیستم از یک سیستم پیکربندی ساده مبتنی بر XML استفاده می‌کند و از میانجیگری پیام ناهمزمان(asynchronous message mediation) با توان عملیاتی بالا(high throughput) پشتیبانی می‌کند. منطق میانجیگری آن(mediation logic) از طریق میانجی‌های داخلی یا سفارشی(built-in or custom mediators) تعریف می‌شود. آپاچی سیناپس یک نرم‌افزار رایگان و متن‌باز است که تحت مجوز نرم‌افزار آپاچی ۲.۰ توزیع شده است.

برای مطالعه ی Documentation میتوانید به وب سایت به نشانی: https://synapse.apache.org/ مراجعه کنید و در ستون سمت چپ، Documentation را مشاهده کنید:

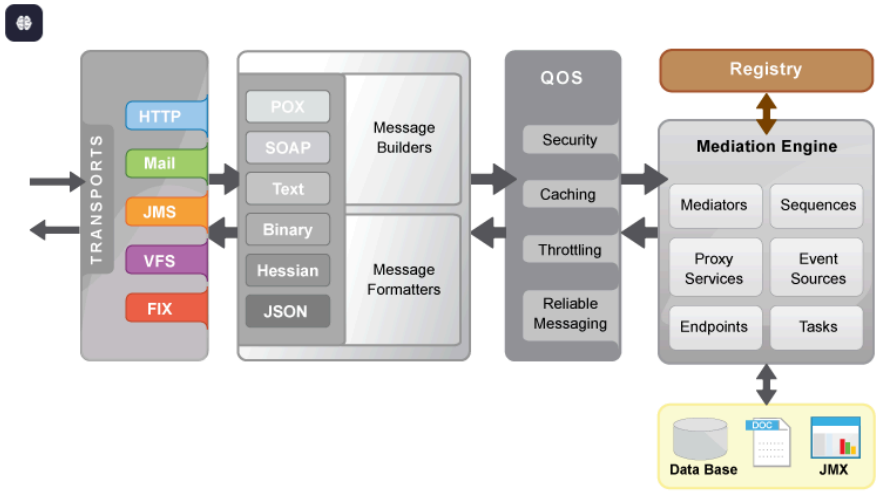


معماری: بر محور مدیاتورها و خطوط عملیاتی از پیش تعریف شده. توضیح معماری سطح بالا: آپاچی سیناپس طوری

طراحی شده است که سبک و سریع باشد. انتقال HTTP بدون انسداد، موتور میانجیگری چند رشته‌ای و مجموعه اطلاعات XML استریمینگ با هم ترکیب می‌شوند تا اطمینان حاصل شود که سیناپس می‌تواند حجم بسیار بالایی از پیام‌ها را از طریق گذرگاه سرویس(service bus) با حداقل تأخیر و استفاده از منابع میانجیگری کند. سیناپس همچنین دارای قابلیت‌های جامع ثبت وقایع، جمع‌آوری آمار و پشتیبانی از نظارت JMX است که در استقرارهای عملیاتی بسیار مهم هستند:

High Level Architecture

Apache Synapse is designed to be lightweight and fast. The non-blocking HTTP transport, the multi-threaded mediation engine and the streaming XML infoset combine to ensure that Synapse can mediate very high volumes of messages through the service bus with minimum delay and resource usage. Synapse also comes with comprehensive logging capabilities, statistics collection and JMX monitoring support which are crucial in production deployments.



Mule ESB -2: یکی از محبوب‌ترین ESBهای متن‌باز با تعداد دانلود بالا، توسعه سریع، و پشتیبانی از JMS، JDBC، HTTP و قالب‌های متنوع. مول یکی از پرکاربردترین ESBهای متن‌باز است. این نرم‌افزار یک موتور زمان اجرا سبک، کتابخانه بزرگی از کانکتورها و پشتیبانی از پروتکل‌های مختلف انتقال از جمله JDBC، JMS، HTTP و FTP را ارائه می‌دهد. مول با ابزارهای غنی و یک الگوی طراحی قدرتمند مبتنی بر جریان، توسعه و استقرار سریع را ارائه می‌دهد. در حالی که هسته آن متن‌باز است، دارای نسخه سازمانی نیز می‌باشد. این گذرگاه رایگان است و مانند اکثر ESBها، به شما امکان می‌دهد سیستم‌ها را با استفاده از JMS، سرویس‌های وب، JDBC، HTTP و سایر روش‌ها به هم متصل کنید.

برای مطالعه ی Documentation به وب سایت مراجعه کنید: https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb

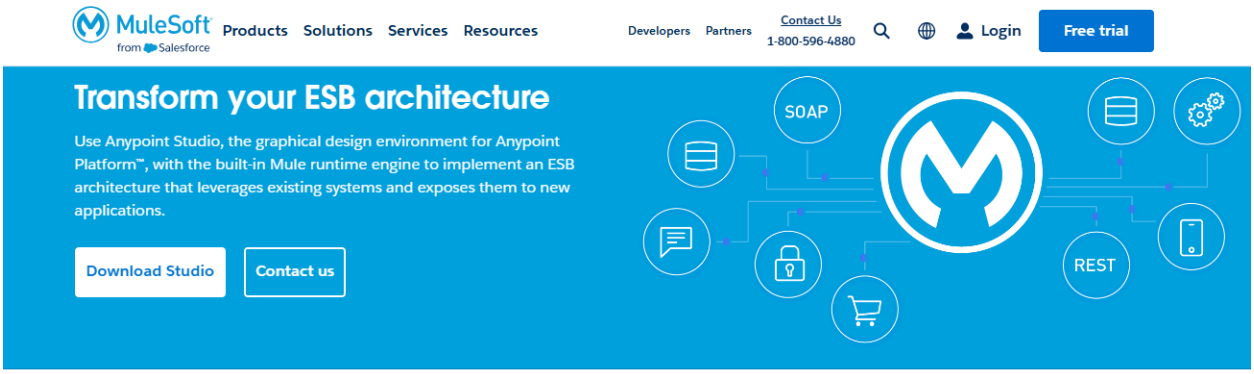
معماری:

چندپروتکلی، استاندارد-بی‌طرف، microkernel سبک همراه با Mule runtime، انعطاف‌پذیر و ادغام با ActiveMQ/ServiceMix

AMQP (پروتکل صف‌بندی پیام پیشرفته): روترهای کلاینت جاوا RabbitMQ برای پشتیبانی از AMQP استفاده می‌شوند. MuleSoft پیام‌ها را با استفاده از روترها تقسیم، ترکیب، مرتب‌سازی مجدد، ارزیابی و پخش می‌کند.

اتصالات پروتکل، پایگاه داده، انتقال و پایگاه داده از پیش ساخته شده از Anypoint Connectors در دسترس هستند.

در صورت لزوم، می‌توانید اتصالات خودتان را نیز بسازید.
موتور زمان اجرای Mule مغز متفکر MuleSoft Anypoint است. می‌توان از آن در فضای ابری یا در محل استفاده کرد.
مدیر زمان اجرای Mule: به نمونه‌های Mule اجازه می‌دهد تا مستقر، نظارت و عیب‌یابی شوند
موتور زمان اجرای Mule(همان Mule runtime engine)، هنگامی که به عنوان ESB مستقر می‌شود، قدرت ادغام داده‌ها و برنامه‌ها را در برنامه‌های قدیمی و SaaS ترکیب می‌کند.



A modern ESB for the digital era

Modernize and unlock the value of existing on-premises systems and applications with an Enterprise Service Bus (ESB) architecture that serves

3- Apache ServiceMix: آپاچی سرویس میکس (Apache ServiceMix) یک کانتینر یکپارچه‌سازی انعطاف‌پذیر و مقیاس‌پذیر است که آپاچی کمل (Apache Camel)، اکتیو ام کیو (ActiveMQ) و سی ایکس اف (CXF) را تحت چارچوب OSGi با استفاده از آپاچی کاراف (Apache Karaf) ترکیب می‌کند. این کانتینر برای برنامه‌های ماژولار و استقرارهای پیچیده مناسب است. سرویس میکس (ServiceMix) از بارگذاری پویای ماژول و مسیریابی سرویس از طریق کامل دی اس ال (Camel DSL) پشتیبانی می‌کند.

برای مطالعه ی Documentation به وب سایت مراجعه کنید: https://sourceforge.net/software/product/Apache-ServiceMix/

معماری: مبتنی بر OSGi با Karaf و Camel برای مسیریابی، ActiveMQ برای صف

4- UltraESB: یک ESB سبک وزن است که برای یکپارچه‌سازی با کارایی بالا ساخته شده است. این ESB از پروتکل‌ها و قالب‌های پیام زیادی از جمله FIX، JMS، REST، SOAP و HL7 پشتیبانی می‌کند. این ESB با NIO غیر مسدودکننده و انتقال بدون کپی ساخته شده است و از اسکریپت‌نویسی از طریق JSR-223 پشتیبانی می‌کند. این ESB یک رابط کاربری گرافیکی (GUI) و ابزارهای خط فرمان مناسب برای توسعه‌دهندگان را برای مدیریت و نظارت فراهم می‌کند. UltraESB تنها ESB است که از دسترسی مستقیم به حافظه (DMA) و فراخوانی سیستم ارسال فایل و همچنین ورودی/خروجی غیر مسدودکننده (Non-Blocking IO) برای ارائه پروکسی Zero-Copy برای عملکرد فوق‌العاده (extreme performance) استفاده می‌کند. UltraESB داده‌های زمان اجرا را از طریق REST APIها به طور ایمن در معرض نمایش قرار می‌دهد که می‌توان با استفاده از هر برنامه نظارتی خارجی آنها را مشاهده کرد.

برای مطالعه ی Documentation به وب سایت مراجعه کنید: https://www.adroitlogic.com/products/ultraesb/

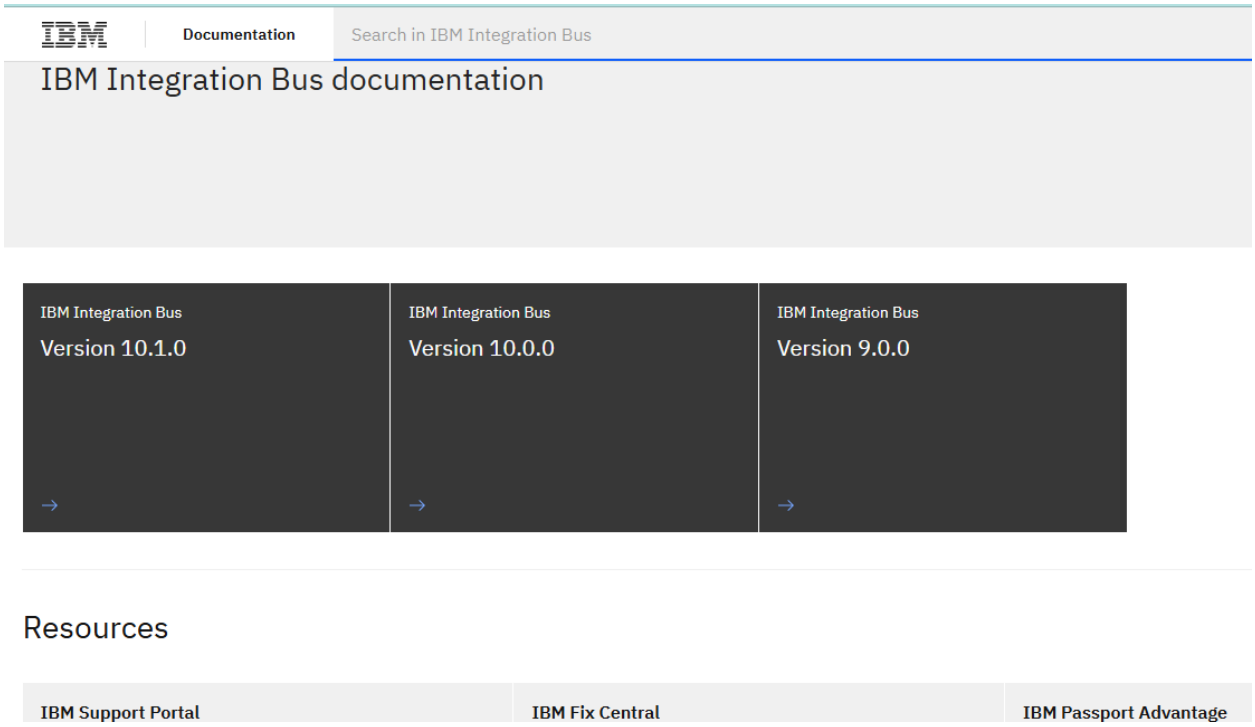
معماری: استفاده از NIO غیرمسدود، Zero-Copy، اسکریپت‌پذیری JSR-223، مدیریت خوشه‌ای و JMX.

Key Features:

XTerm, a scriptable command-line interface for administration, is included.
Contains a built-in metrics engine that collects message level and runtime statistics with historical data using Elasticsearch.
Ensures that performance is not harmed.
On top of the Project-X framework, it uses a series of connectors and processors.
The connector/processor repository contains dozens of connectors and processors that are ready to use right away.

برای مطالعه ی بیشتر فیچر ها، ویژگی ها و معماری به https://en.wikipedia.org/wiki/UltraESB مراجعه کنید.

5- IBM Websphere ESB: این ESB سازمانی شرکت IBM است. این گذرگاه به یکپارچه‌سازی سیستم‌های سرویس‌گرا، پیام‌گرا و رویدادگرا کمک می‌کند.



برای مطالعه ی Documentation به وب سایت مراجعه کنید: https://www.ibm.com/docs/en/integration-bus

Key Features:

- IBM's version of the JMS APIs is WebSphere MQ.
- Routing/EIPs: XSLT can be used to implement changes like content-based navigation and other corporate integration patterns.
- Adapters for WebSphere: Protocol, database, transportation, and database connectors are all pre-built.
- WebSphere Application Server: WebSphere ESB's runtime is built on top of WAS.
- Administrative Console: A browser-based interface for monitoring, updating, and starting and stopping WebSphere ESB applications, services, and resources.

- **نکته ای در مورد معماری ESB ها:**

همه ESB های ذکر شده از یک الگوی معماری مشابه پیروی می‌کنند:

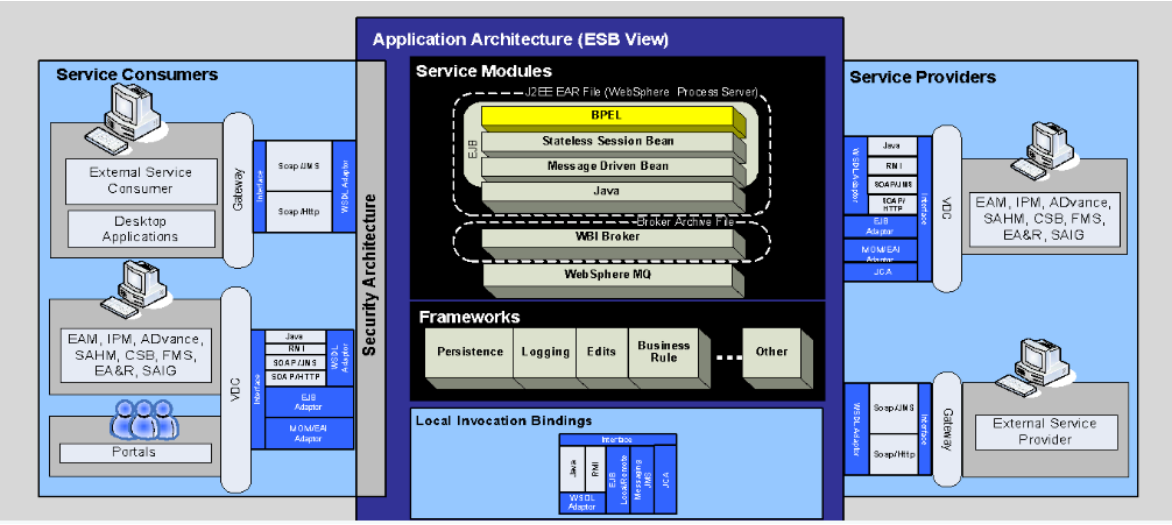
Gateway Layer: اتصالات ورودی/خروجی را از طریق HTTP/SOAP/JMS و غیره می‌پذیرد.

Mediation Layer: مسیریابی، تبدیل، اعتبارسنجی و امنیت را مدیریت می‌کند.

Service Invocation Layer: رابط با ماژول‌های سرویس یا سرویس‌های خارجی.

مدیریت و نظارت: JMX، ثبت وقایع، داشبوردها، کنسول‌ها برای ادمین.

این ساختار با نمودار معماری ESB که در اسلایدهای درس ارائه شد، همسو است، جایی که ماژول‌های سرویس در اطراف ESB قرار می‌گیرند و ESB تعاملات بین آنها را مدیریت می‌کند:



- **مقایسه بین ESB های گفته شده:**

Comparison of 5 ESBs Based on Key Features 📜

IBM WebSphere ESB	UltraESB	Apache ServiceMix	Mule ESB (Community)	Apache Synapse	Feature
Paid (Commercial License)	Free (AGPL License)	Free (Apache License 2.0)	Free (Core is open-source, paid enterprise)	Free (Apache License 2.0)	**Cost**
High (enterprise-level setup)	Medium	Medium to High (due to OSGi/Karaf)	Low	Low to Medium	**Setup & Maintenance Complexity**
Small (Enterprise licensed, closed community)	Smaller but active	Medium	Very Large	Medium (Apache Community)	**Developer Community Size**
Not open-source ❌	Fully open-source ✅ (AGPL)	Fully open-source ✅	Core only (Enterprise ✅ version is paid)	Fully open-source ✅	**Open Source**
Deep WebSphere integration, enterprise-grade features	High performance, non-blocking, scripting	Modular (Camel, Karaf, CXF), enterprise-grade	Rich connector library, visual dev tools	Lightweight, mediator-based, high speed	**Special Features**
HTTP, JMS, SOAP, MQ, SCA, .etc	HTTP, REST, SOAP, JMS, .HL7, FIX, etc	HTTP, JMS, SOAP, REST, CXF, ActiveMQ	HTTP, JMS, JDBC, FTP, WebSockets	HTTP, SOAP, REST, JMS, VFS, Email	**Protocol Support**
Integrated WebSphere admin tools	Web UI, CLI, JMX	Karaf Console, JMX	Built-in monitoring dashboard	JMX, Logging	**Monitoring Tools**
High	Medium	Medium to High	Low	Low to Medium	**Learning Curve**
Large enterprise environments with IBM stack	Performance-critical or protocol-rich use cases	Modular enterprise apps	Rapid development, scalable apps	Lightweight ESB needs	**Ideal For**

Mule ESB هم نسخه‌های عمومی (متن‌باز) و هم نسخه‌های سازمانی (پولی) دارد. بسیاری از رابط‌ها و ابزارهای نظارتی فقط در نسخه سازمانی موجود هستند.

Apache ServiceMix در مواقعی که به ماژولاریتی و OSGi نیاز است، ایده‌آل است، اما منحنی یادگیری آن سریع‌تر است.

UltraESB در مدیریت پیام‌های خاص پروتکل در سیستم‌های بلادرنگ بسیار قدرتمند است.

مرحله دوم

- **مراحل روشن کردن یک نسخه از WSO2:**

در این بخش، ما WSO2 Enterprise Service Bus، یک ESB با کارایی بالا و متن‌باز مبتنی بر موتور Apache Synapse را بررسی میکنیم. ESB را می‌توان به عنوان بخشی از WSO2 Enterprise Integrator دانلود کرد.

مراحل راه‌اندازی(WSO2 Micro Integrator (ESB

1: دریافت فایل نصبی WSO2 Micro Integrator

1. وارد سایت رسمی WSO2 می شویم
https://wso2.com/integration/micro-integrator

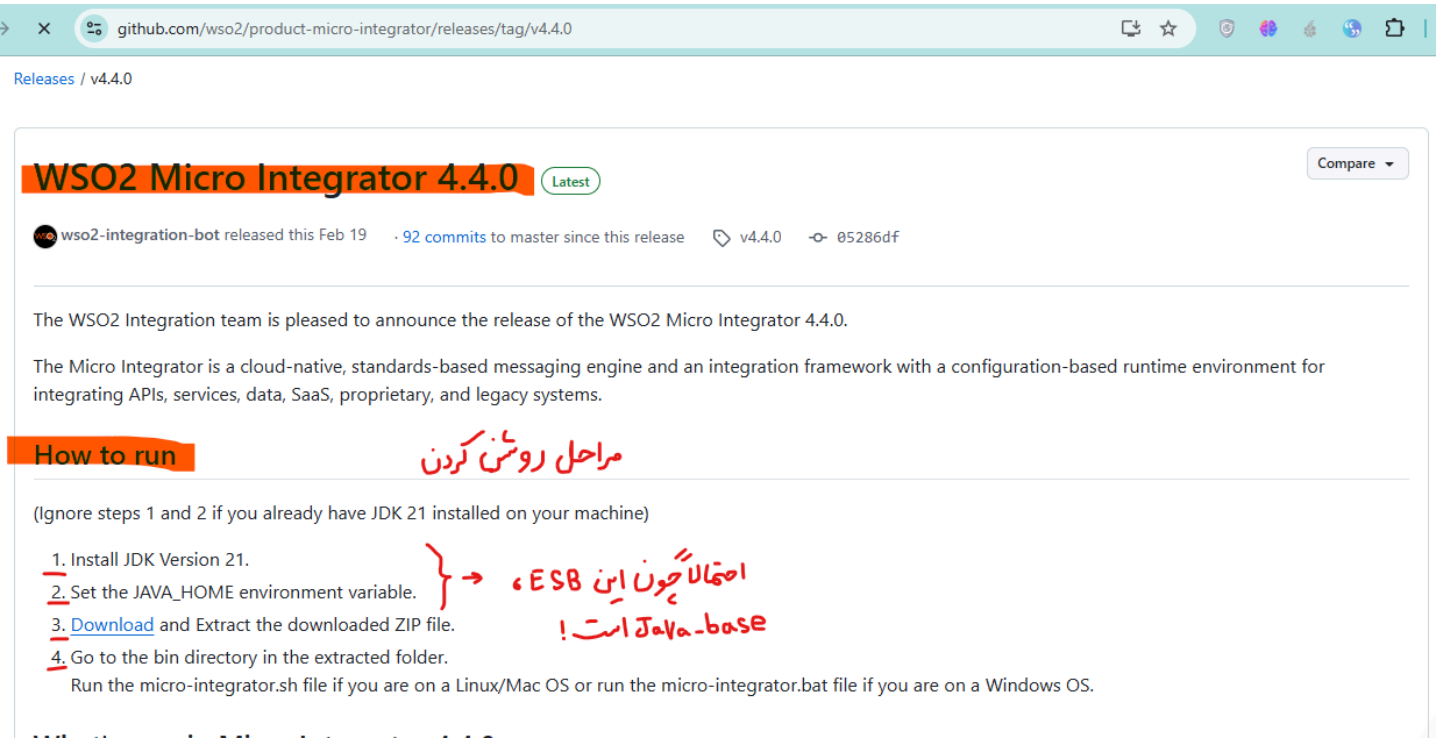
2. در صفحه‌ی نمایش داده شده، گزینه‌ی **WSO2 Integrator: MI** را انتخاب کرده و روی دکمه‌ی **Download** کلیک میکنیم

3. فایل zip را دریافت میکنیم

2: استخراج فایل و آماده‌سازی اجرا

4. فایل ZIP دریافت‌شده را در محل دلخواه Extract میکنیم.
5. وارد پوشه‌ی استخراج‌شده میشویم. معمولاً نام آن به صورت `wso2mi-x.x.x` است.(ما در اینجا اخرین نسخه را دانلود کردیم پس می شود `wso2mi-4.4.0`

نکته ی مهم: یک روش دیگر این است که می توانیم فایل زیپ را از گیت هاب wso2 دانلود کنیم و طبق مراحل گفته شده در صفحه ی گیت هاب پیش برویم:



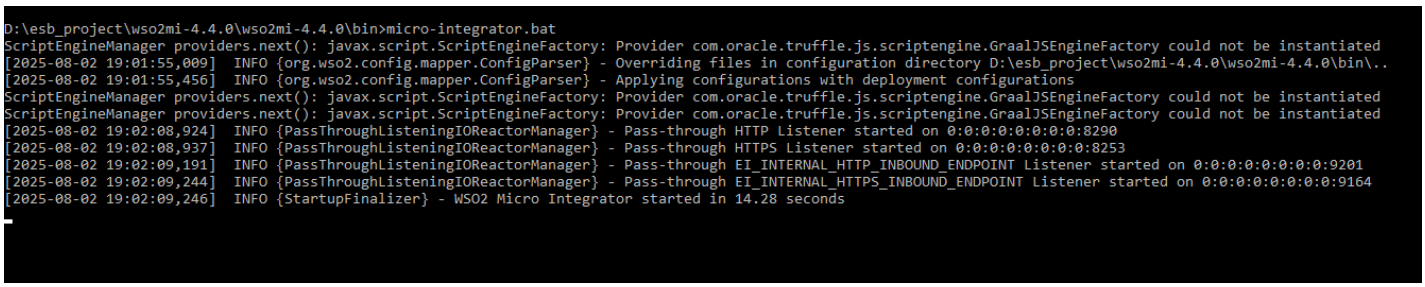
3: اجرای WSO2 Micro Integrator

6. بسته به سیستم‌عامل، به مسیر `/bin` در پوشه‌ی Micro Integrator میرویم و فایل مناسب را اجرا میکنیم:

```
bin\micro-integrator.bat
```

7. پس از چند ثانیه، در ترمینال پیام‌هایی ظاهر می‌شود که نشان‌دهنده‌ی روشن شدن سرور WSO2 Micro Integrator است.

اجرا:



پس تا اینجای کار:

- 1. ما WSO2 رو با micro-integrator.bat از ویندوز اجرا کردیم
- 2. حالا یک فایل xml می سازیم، این فایل XML یک REST API می‌سازه که وقتی به آدرس http://localhost:8290/hello/greet میرویم، به ما جواب می‌ده:
"Hello from WSO2 Micro Integrator!"، محتوای فایل xml:

D: > esb_project > wso2mi-4.4.0 > wso2mi-4.4.0 > repository > deployment > server > synapse-configs > default > api > HelloAPI.xml

```xml
<api xmlns="http://ws.apache.org/ns/synapse" name="HelloAPI" context="/hello">
    <resource methods="GET" uri-template="/greet">
        <inSequence>
            <payloadFactory media-type="text">
                <format>Hello from WSO2 Micro Integrator!</format>
            </payloadFactory>
            <respond/>
        </inSequence>
        <outSequence/>
    </resource>
</api>

```

فایل XML نقش تعریف سرویس REST API را دارد. در این فایل مسیر (URL)، متد (GET/POST) و پاسخ مورد نظر مشخص می‌شود. WSO2 MI با خواندن این فایل، سرویس را به‌صورت خودکار راه‌اندازی می‌کند. بدون این فایل، API قابل دسترسی نخواهد بود.

- 3. به فولدر extract شده ی wso2 ای که فایل زیپ ان را دانلود کردیم میرویم، سپس مسیر زیر را دنبال میکنیم(به فولدر repository میرویم، سپس به فولدر deployment، سپس به فولدر server، سپس به فولدر synapse-configs، سپس به فولدر default و سپس به فولدر api که باید خودمان ان را بسازیم)و فایل xmlمان را در پوشه ی api قرار می دهیم(پوشه ی api را باید خودمان بسازیم) میتوانید این فایل xml را در ریپازیتوری گیت هاب پیدا کنید

wso2mi-4.4.0 › wso2mi-4.4.0 › repository › deployment › server › synapse-configs › default › api

- 4. سرویس را روشن کرده بودیم و با قرار دادن فایل xml در فولدر api، نتیجه این می شود:

```
D:\esb_project\wso2mi-4.4.0\wso2mi-4.4.0\bin>micro-integrator.bat
ScriptEngineManager providers.next(): javax.script.ScriptEngineFactory: Provider com.oracle.truffle.js.scriptengine.GraalJSEngineFactory could not be instantiated
ScriptEngineManager providers.next(): javax.script.ScriptEngineFactory: Provider com.oracle.truffle.js.scriptengine.GraalJSEngineFactory could not be instantiated
ScriptEngineManager providers.next(): javax.script.ScriptEngineFactory: Provider com.oracle.truffle.js.scriptengine.GraalJSEngineFactory could not be instantiated
[2025-08-02 22:55:57,598]  INFO {API} - {api:HelloAPI} Initializing API: HelloAPI
[2025-08-02 22:55:57,791]  INFO {PassThroughListeningIOReactorManager} - Pass-through HTTP Listener started on 0:0:0:0:0:0:0:0:8290
[2025-08-02 22:55:57,808]  INFO {PassThroughListeningIOReactorManager} - Pass-through HTTPS Listener started on 0:0:0:0:0:0:0:0:8253
[2025-08-02 22:55:58,123]  INFO {PassThroughListeningIOReactorManager} - Pass-through EI_INTERNAL_HTTP_INBOUND_ENDPOINT Listener started on 0:0:0:0:0:0:0:0:9201
[2025-08-02 22:55:58,192]  INFO {PassThroughListeningIOReactorManager} - Pass-through EI_INTERNAL_HTTPS_INBOUND_ENDPOINT Listener started on 0:0:0:0:0:0:0:0:9164
[2025-08-02 22:55:58,194]  INFO {StartupFinalizer} - WSO2 Micro Integrator started in 13.12 seconds
```

معنی هرکدام از خطوط:

1. ScriptEngineManager...GraalJSEngineFactory could not be instantiated:
   مربوط به یک موتور اسکریپت‌نویسی جاوا (GraalJS) است که در این پروژه استفاده نمی‌شود و می‌توان از آن صرف‌نظر کرد.
2. [API] - {api:HelloAPI} Initializing API: HelloAPI

نشان می‌دهد که فایل REST API به‌درستی بارگذاری (deploy) شده است.

3. Pass-through HTTP Listener started on ... 8290
4. Pass-through HTTPS Listener started on ... 8253

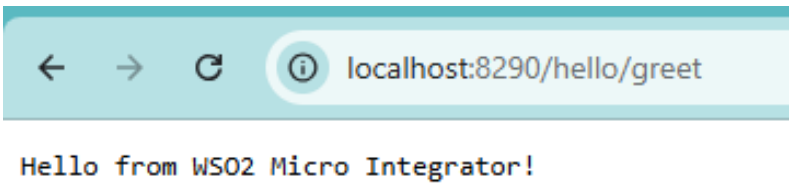
مربوط به Listenerهای داخلی WSO2 برای پردازش پیام‌ها در داخل سیستم است.

5. StartupFinalizer - WSO2 Micro Integrator started in ... seconds

تایید می‌کند که سرور WSO2 MI با موفقیت راه‌اندازی شده است.

هنگام اجرای فایل `micro-integrator.bat`، لاگ‌هایی ظاهر می‌شوند که نشان‌دهنده‌ی موفقیت‌آمیز بودن راه‌اندازی سرور WSO2 هستند. از جمله، بارگذاری موفق فایل API، فعال شدن پورت‌های HTTP/HTTPS، و اعلام راه‌اندازی کامل سرور. هشدارهای مربوط به ScriptEngine بی‌اهمیت بوده و تاثیری در عملکرد ندارند.

- **5**. API ما با موفقیت deploy شده، سرور روشنه، و آماده‌ایم که ان را تست کنیم، در مرورگر میزنیم:http://localhost:8290/hello/greet



پس از روشن‌کردن WSO2 Micro Integrator و deploy کردن فایل XML، آدرس `http://localhost:8290/hello/greet` در مرورگر تست شد. این آدرس توسط API تعریف‌شده در فایل XML پاسخ می‌دهد. نتیجه، نمایش موفق پیام مورد انتظار بود که نشان‌دهنده عملکرد صحیح سرویس REST API است

کار بخش دوم پروژه، در اینجا به پایان می رسد، ولی در ادامه، اگر بخواهیم کارهای بیشتری با wso2 انجام دهیم میتوانیم مراحل زیر را طی کنیم:

4: نصب افزونه‌ی VSCode (برای ساخت و مدیریت فایل‌های integration) (اختیاری برای پروژه ی ما)

نکته: این گام مربوط به محیط توسعه است، نه اجرای سرور.

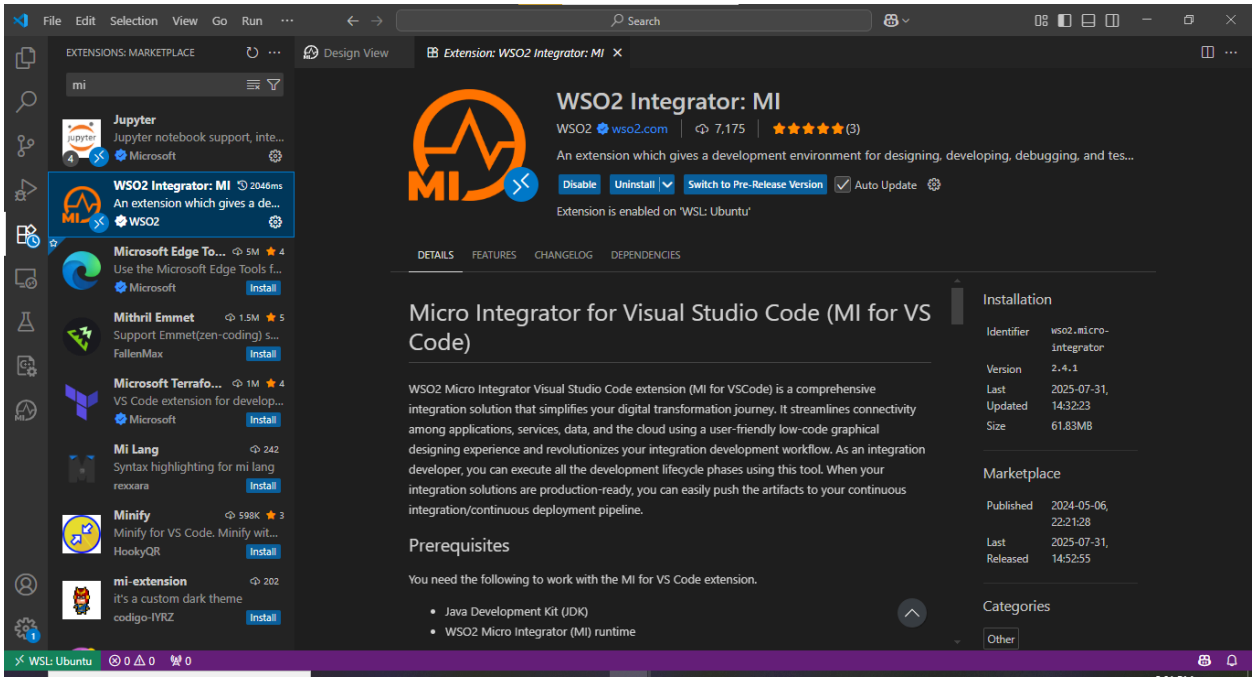
با استفاده از افزونه VSCode (WSO2 Integrator: MI)، می‌تونیم فایل‌های زیر رو بسازیم:

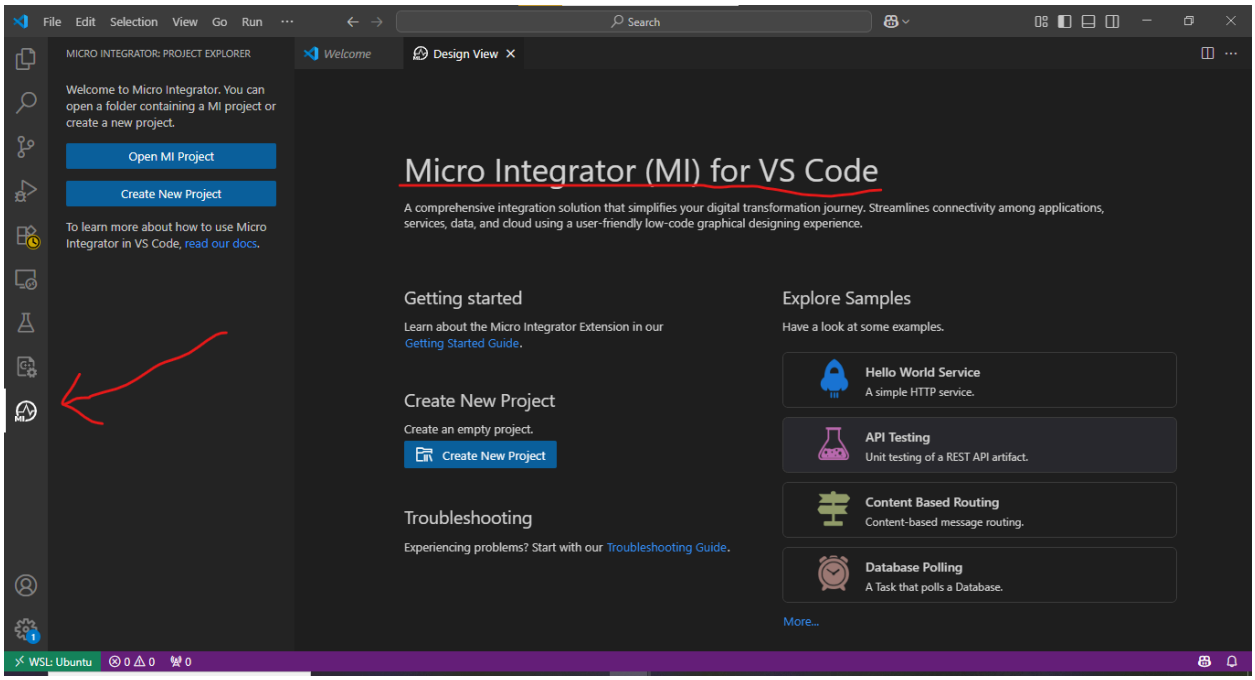
- REST API
- Proxy Services
- Sequences
- Endpoints

و بعد با استفاده از CLI یا Carbon Console اون‌ها رو deploy کنیم.

ولی ما در مرحله ی قبل فایل xml را ساخته ایم و لازم نیست با استفاده از این افزونه هم دوباره فایل دیگری بسازیم. از اینجا به بعد، مراحل فقط برای پروژه هایی با اهداف دیگر است و در پروژه ی ما، فایل xml را در گام قبل ساخته ایم و کارمان تمام است.

8. در محیط **Visual Studio Code**، وارد تب **Extensions** میشویم و عبارت `WSO2 Integrator: MI` را جستجو و نصب میکنیم.





این افزونه به ما کمک می‌کند تا:

- فایل‌های proxy service، REST API و غیره را با قالب صحیح بسازیم.

- ساختار پروژه‌ی integration خود را مطابق استاندارد WSO2 ایجاد کنیم.
- پکیج `.car` یا `.xml` برای deploy بسازیم.

نکته مهم: این افزونه فقط فایل‌ها را ایجاد می‌کند؛ برای اجرای آن‌ها همچنان نیاز به Micro Integrator داریم که باید طبق گام ۳ روشن باشد.

5. **تست روشن بودن سرور**

9. برای اطمینان از روشن بودن WSO2، می‌توانیم در مرورگر آدرس زیر را باز کنیم:

```
http://localhost:8290/healthz
```

در صورت موفقیت، پاسخ `OK 200` دریافت خواهیم کرد.

6: ساخت اولین REST API (اختیاری برای پروژه)

10. میتوانیم با استفاده از افزونه VSCode یک فایل REST API بسازیم
11. فایل ساخته‌شده را در مسیر مناسب قرار میدهیم و با استفاده از WSO2 CLI یا قرار دادن در فولدر `deployment` آن را deploy میکنیم.

مرحله سوم

راه اندازی یک نسخه از معماری apache synapse:-

Apache Synapse یک **ESB (Enterprise Service Bus)** سبک‌وزن هست که پیام‌ها (message) رو بین سرویس‌های مختلف منتقل، مسیریابی، تبدیل (transform) و فیلتر می‌کنه.

High Level Architecture:

آپاچی Synapse طوری طراحی شده است که سبک و سریع باشد. انتقال HTTP بدون انسداد، موتور میانجیگری چند رشته‌ای و مجموعه اطلاعات XML استریمینگ با هم ترکیب می‌شوند تا اطمینان حاصل شود که Synapse می‌تواند حجم بسیار بالایی از پیام‌ها را از طریق گذرگاه سرویس با حداقل تأخیر و استفاده از منابع، میانجیگری کند. Synapse همچنین دارای قابلیت‌های جامع ثبت وقایع، جمع‌آوری آمار و پشتیبانی از نظارت JMX است که در استقرارهای تولید بسیار مهم هستند.

معماری ساده‌ی Synapse: نسخه‌ی ساده‌شده‌ی معماری Apache Synapse ESB معمولاً از اجزای کلیدی زیر تشکیل شده:

مولفه	وظیفه
Message Receiver	دریافت پیام ورودی (مانند HTTP یا فایل)
Message Context	نگهداری اطلاعات پیام (header, body, property)
Mediators	انجام عملیات روی پیام (log، transform، route)
Message Sender	ارسال پیام به سرویس مقصد
Configuration Loader	بارگذاری تنظیمات (معمولاً از XML)

پس Apache Synapse ESB ما: پیامی را دریافت میکند، روی آن عملیاتی انجام می دهد (Log یا تبدیل)، آن را به مقصد می فرستد (یا فقط چاپ می کند برای تست). یعنی شامل دریافت، مسیریابی، تبدیل و ارسال پیام‌ها است. سپس دو سرویس mcp server می سازیم، دو agent هوش مصنوعی ما باید بتوانند یک سناریو ی فرضی اتصال دو سرویس (مثلا یک سرویس می تواند دیتای هواشناسی و یک سرویس می تواند دیتای جهانگردی باشد) را با استفاده از تعامل بین خودشان پیاده سازی کنند. یک سرویس، دیتای هواشناسی است و یک سرویس، دیتای گردشگری.

ما یک نسخه از معماری ساده‌ی Apache Synapse رو طراحی می‌کنیم که در اون:

1. سرویس هواشناسی (WeatherService): دما، بارندگی و آفتابی بودن یا نبودن شهر رو برمی‌گردونه.
2. سرویس گردشگری (TourismService): بر اساس شهر، جاهای دیدنی رو پیشنهاد می‌ده.

—> با رد و بدل کردن پیام بین سرویس ها، اگر هوای ان شهر افتابی باشد، سرویس گردشگری ان شهر را به عناون گزینه ی خوب برای سفر پیشنهاد می دهد و جاهای دیدنی اش را بر می گرداند، و اگر هوای ان شهر ابری باشد، ان شهر را برای سفر توصیه نمی کند

3. Mediator هوشمند (AgentMediator): لیستی از شهرها رو بررسی می‌کنه، اگر هوا آفتابی باشه برای اون شهر، درخواست اطلاعات گردشگری می‌فرسته و نتایج رو نمایش می‌ده.

سناریو ای که رخ می دهد(خیلی مهم!):

سناریوی اول:

در ابتدا ما لیستی از شهر ها داریم، انها را برای agent mediator می فرستیم. Agent mediator به ازای هر شهر، سرویس هواشناسی را روی ان شهر صدا می زند. سرویس هواشناسی به ما می گوید ان شهر افتابی است یا ابری(به ازای هر run ای که می گیریم، افتابی بودن یا ابری بودن شهر ها تغییر میکند، یعنی هر دفعه شبیه سازی هوای تصادفی برای شهر ها اتفاق می افتد، هوا یا افتابی است یا غیر افتابی)، سپس agent mediator پیام را بدست سرویس گردشگری می رساند، و به عنوان واسط، برای سرویس گردشگری نقش مترجم پیام سرویس هواشناسی را بازی می کند(چون سرویس گردشگری زبان سرویس هواشناسی را

نمیفهمد و نیاز به یک واسط دارد)، اگر شهر افتابی بود، یعنی ان شهر برای سفر انتخاب خوبی است، و سرویس گردشگری، مکان های دیدنی ان شهر را می گوید و ان شهر را برای سفر توصیه می کند(اینکه "افتابی بودن شهر" معادل "ان شهر مقصد خوب برای سفر است" را معماری esb ما پیاده سازی کرده)، ولی اگر شهر افتابی نبود، سرویس گردشگری، می گوید ان شهر مقصد خوبی برای سفر نیست.

توضیح چیزهایی که گفتیم از روی کد:

- در فایل Main.java ، فرستادن شهر ها برای agent mediator

```
//First scenario

List<String> cities = Arrays.asList("Tehran", "Shiraz", "Mashhad", "Tabriz");
mediator.mediateCities(cities);
```

- در AgentMediator: به ازای هر شهر عملیات را شروع میکنیم

```
public void mediateCities(List<String> cities) {
    for (String city : cities) {
        MessageContext context = new MessageContext(city);
        mediate(context);
        System.out.println("----------");
    }
}
```

افتابی بودن یا نبودن ان شهر را از سرویس هواشناسی می گیریم :

```
boolean sunny = weatherService.isWeatherSunny(context.getCity());
context.setSunny(sunny);
```

و سرویس گردشگری اگر هوای ان شهر افتابی باشد، ان شهر را برای سفر به ما توصیه می کند و جاهای دیدنی اش را به ما می گوید:

```
String info = tourismService.getTourismSuggestions(context.getCity());
context.setTourismInfo(info);
System.out.println("The weather is sunny! :) Tourism suggestion for" + context.getCity() + ": " + info);
```

```
    private WeatherService weatherService = new WeatherService();
    private TourismService tourismService;
    private Mediator logger = new LogMediator();

    public void mediate(MessageContext context) {
        logger.mediate(context);

        boolean sunny = weatherService.isWeatherSunny(context.getCity());
        context.setSunny(sunny);

        if (sunny) {
            String info = tourismService.getTourismSuggestions(context.getCity());
            context.setTourismInfo(info);
            System.out.println("The weather is sunny! :) Tourism suggestion for" + context.getCity() + ": " + info);
        } else {
            System.out.println("weather in" + context.getCity() + " Not suitable for travel.");
        }
    }
```

- در سرویس هواشناسی، افتابی بودن و نبون هر شهر بصورت رندم تعیین می شود(ولی می توانستیم داده های واقعی هم به این سرویس وصل کنیم، که پیش فرض من این بود که می شود افتابی بودن هر شهر را رندم تعیین کرد):

```
public boolean isWeatherSunny(String city) {

    Random random = new Random();
    return random.nextBoolean();
}
```

- جاهای دیدنی ای که سرویس گردشگری برای هر شهر توصیه می کند:

```
public String getTourismSuggestions(String city) {
    switch (city) {
        case "Shiraz":
            return "Visit Persepolis and enjoy the gardens.";
        case "Tehran":
            return "Visit Milad Tower and Golestan Palace.";
        case "Tabriz":
            return "Explore the Bazaar of Tabriz and nearby mountains.";
        default:
            return "No suggestions available.";
    }
}
```

- توصیه نشدن شهر بارانی:

```
} else {
    System.out.println("weather in" + context.getCity() + " Not suitable for travel.");
}
```

خروجی مربوط به سناریوی اول:

```
root@DESKTOP-BEI7BD8:/home/narges/projects/SOA_ESB/SOA-and-ESB-project# java Main
Logging: City = Tehran
The weather is sunny! :) Tourism suggestion forTehran: Visit Milad Tower and Golestan Palace.
----------
Logging: City = Shiraz
weather inShiraz Not suitable for travel.
----------
Logging: City = Mashhad
The weather is sunny! :) Tourism suggestion forMashhad: No suggestions available.
----------
Logging: City = Tabriz
weather inTabriz Not suitable for travel.
----------
```

سناریوی دوم:

در سناریوی بعدی، برای اینکه ارتباط میان سرویس ها دو طرفه باشد، 2 شهر (قم و رشت) را انتخاب می کینم، این بار سرویس گردشگری اسم شهر را برای AgentMediator می فرستد و AgentMediator با فراخوانی متد getWeather مربوط به سرویس هواشناسی، از سرویس هواشناسی افتابی بودن یا بارانی بودن هوای ان شهر را می گیرد، و برای سرویس گردشگری به گونه ای می فرستد که برای سرویس گردشگری قابل فهم باشد(نقش واسط و مترجم پیام های دو سرویس را بازی میکند، در واقع دارد یک استاندارد برای حرف زدن دو سرویس ارائه میدهد)، و سرویس گردشگری بر مبنای دیتایی که از ان شهر دارد، می گوید ان شهر مناسب سفر هست یا نه، اگر مناسب هست چه مکان های دیدنی ای برای ان شهر توصیه می شود. همه‌ی تعاملات مطابق معماری Synapse است (پیام، واسطه، سرویس‌ها).

توضیح چیزهایی که گفتیم از روی کد:

- به ازای هر شهر، AgentMediator را روی ان شهر صدا زده میشود:

Main.java

```java
public class Main {
    public static void main(String[] args) {
        //Second scenario


        String city1 = "Qom";
        String tourismAdvice1 = mediator.requestTourismAdvice(city1);
        System.out.println("Tourism agent says about " + city1 + ": " + tourismAdvice1);

        String city2 = "Rasht";
        String tourismAdvice2 = mediator.requestTourismAdvice(city2);
        System.out.println("Tourism agent says about " + city2 + ": " + tourismAdvice2);
    }
}
```

- سرویس گردشگری روی ان شهر صدا زده می شود:

```java
public class AgentMediator implements Mediator {
    }

    public String requestTourismAdvice(String city) {
        return tourismService.getTravelRecommendation(city);
    }
}
```

- سرویس گردشگری به ازای ان شهر، agent mediator را صدا میزند که agent mediator، وضعیت هوای ان شهر را از سرویس هواشناسی می گیرد:

```java
public class TourismService {

    public String getTravelRecommendation(String city) {

        String weather = mediator.requestWeather(city);
        if (weather.equals("Sunny")) {
            return "Great time to visit " + city + "! Recommended sites: City Center, Old Bazaar.";
        } else if (weather.equals("Rainy")) {
            return "Not the best time to visit " + city + " due to rain.";
        } else {
            return "No recommendation available for " + city + ".";
        }
    }
}
```

AgentMediator.java

```java
public class AgentMediator implements Mediator {

    public String requestWeather(String city) {
        // System.out.println(weatherService.getWeather(city));
         return weatherService.getWeather(city);
    }
```

و بر اساس وضعیت هوای ان شهر، ان شهر برای سفر توصیه می شود یا نمی شود.(اگر توصیه شود مکان های دیدنی پیشنهادی ان شهر توسط سرویس گردشگری ارائه می شود)

- وضعیت هوای هریک از شهر های قم و رشت را اینگونه به سرویس هواشناسی دادیم، ولی می شد توسط یک فایل هم به سرویس هواشناسی ارائه شود که در این صورت به روز رسانی وضعیت هوا اسان تر است:

WeatherService.java

```java
public class WeatherService {
    }

    public String getWeather(String city) {
        if (city.equalsIgnoreCase("Qom")) {
            return "Sunny";
        } else if (city.equalsIgnoreCase("Rasht")) {
            return "Rainy";
        } else {
            return "Unknown";
        }
    }
}
```

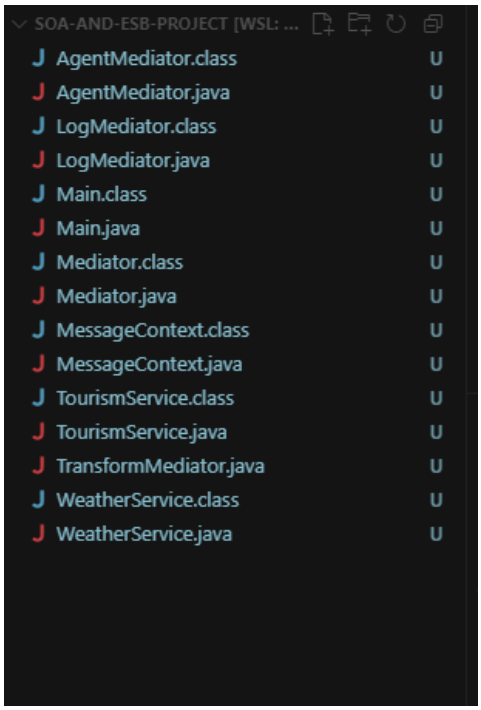
خروجی مربوط به سناریوی دوم:

لطفا برای اجرای سیستم، فایل Main.java را اینگونه اجرا کنید:

کامپایل: javac Main.java

ران: java Main

ساختار فایل ها:



توضیح هر فایل:

WeatherService.java: یک سرویس مستقل (Weather Service) که اطلاعات آب‌وهوا را ارائه می‌دهد. مانند یک Endpoint در معماری Synapse.

TourismService.java: یک سرویس گردشگری مستقل که بسته به وضعیت آب‌وهوا، توصیه‌های سفر می‌دهد. خودش نیز از WeatherService اطلاعات می‌گیرد. مشابه یک سرویس دیگر در معماری.

AgentMediator.java:نقش Enterprise Service Bus (ESB) را دارد. ارتباط بین سرویس‌ها را مدیریت می‌کند. مانند Mediator / Synapse Sequence عمل می‌کند.

Main.java: کلاینت یا مصرف‌کننده نهایی (Client Application) که تعامل با Mediator را آغاز می‌کند. مثل استفاده‌کننده‌ی نهایی در معماری SOA.

MessageContext.java : این کلاس مانند یک ظرف اطلاعاتی (Context) است که داده‌ها را در طول عبور از mediatorها نگه می‌دارد. می‌توان در آن اطلاعات مانند نام شهر، دما، وضعیت هوا، و پیشنهاد گردشگری را ذخیره کرد.

Mediator.java: این یک اینترفیس اصلی است که تمام mediatorها باید آن را پیاده‌سازی کنند. متدی به نام mediate( ) دارد که منطق mediatorها در آن پیاده‌سازی می‌شود.

LogMediator.java: یک mediator ساده برای چاپ اطلاعات موجود در MessageContext در کنسول (برای اشکال‌زدایی و مشاهده جریان پردازش). این کلاس مشابه یک Mediator لاگ‌گیری در Synapse است که وضعیت پیام را در حین عبور از ESB چاپ می‌کند یا لاگ می‌گیرد. برای نظارت و دیباگ جریان پیام‌ها کاربرد دارد.

TransformMediator.java: اگر لازم بود داده‌ای را قبل از ارسال به سرویس مقصد، تغییر دهیم (مثلاً تبدیل فرمت یا اضافه کردن پیشوند)، از این mediator استفاده می‌شود.
در این نسخه ممکن است کاربردش کم‌رنگ‌تر باشد، اما برای توسعه‌پذیری مفید است.

- اگر معماری را به یک سناریوی واقعی در Apache Synapse تشبیه کنیم:
- "سرویس گردشگری" و "سرویس هواشناسی" مانند دو سرویس REST مستقل هستند.
- Agent mediator مانند یک Synapse Proxy Service یا Mediator Sequence است که این دو سرویس را به هم متصل می‌کند.
- Main مانند یک API Consumer یا External Client است که درخواست را ارسال می‌کند

خروجی: برای اجرای سیستم باید Main.java اجرا شود

خروجی به ازای هر ران متفاوت است:

در این بخش، یک معماری ساده‌شده از Apache Synapse پیاده‌سازی شد که شامل دریافت، مسیریابی، تبدیل و ارسال پیام‌ها بود. سپس دو سرویس MCP Server برای ارائه‌ی اطلاعات هواشناسی و گردشگری طراحی شد. دو عامل (Agent) هوش مصنوعی با تولید پیام‌هایی برای هر سرویس، تعامل بین این دو سرویس را از طریق سیستم طراحی‌شده برقرار کردند.

این ساختار، شبیه‌سازی ساده‌ای از معماری Apache Synapse ESB است، چون:

1. Mediator مرکزی (کلاس agentMediator) شبیه به هسته‌ی Synapse، جریان پیام‌ها را کنترل می‌کند.
2. سرویس‌های جداگانه (مثل WeatherService و TourismService) مانند سرویس‌های Backend هستند که از طریق Mediator به هم متصل می‌شوند، نه به‌صورت مستقیم.
3. تصمیم‌گیری درون Mediator، مشابه *Rule-based routing* در Synapse است، مثلاً فقط اگر هوا آفتابی باشد، داده به سرویس گردشگری ارسال می‌شود.
4. ارتباط دوطرفه سرویس‌ها، نشان‌دهنده‌ی *Service Orchestration* در سطح ساده است.

با این ساختار، مفاهیم کلیدی Enterprise Service Bus (مثل Routing، Decoupling و Transformation) به‌شکل ساده و قابل فهم پیاده‌سازی شده‌اند.

لینک گیت هاب مخزن کد ها: https://github.com/nargesasa/SOA-and-ESB-project