**Chapter 3**

**Introduction - XML, JSON**

Sanjay Patnia\*

a UCSC Extension, Santa Clara, California, USA

**Abstract**

This chapter introduces basic concepts about XML and JSON. At the end of this chapter there is an exercise for environment setup.

This chapter introduces basic concepts about XML and JSON. At the end of this chapter there is an exercise to demonstrate XML and JSON responses message from Micronaut app.

**What is XML?**

e**X**tensible **M**arkup **L**anguage - XML is a text-based markup language which is standard for data interchange on the Web. As with HTML, you identify data using [tag](https://www2.informatik.hu-berlin.de/~xing/Lib/Docs/jaxp/docs/tutorial/glossary.html%252525252525252523tag)s (identifiers enclosed in angle brackets, like this: <...>). Collectively, the tags are known as “markup.” It puts a label on a piece of data that identifies it (for example: <message>...</message>). In the same way that you define the field names for a data structure, you are free to use any XML tags that make sense for a given application. Naturally, though, for multiple applications to use the same XML data, they have to agree on the tag names they intend to use. Here is an example of some XML data you might use for a messaging application:

<message>

<to>you@yourAddress.com</to>

<from>me@myAddress.com</from>

<subject>XML Is Really Cool>

</subject>

<text>

How many ways is XML cool? Let me count the ways...

</text>

</message>

Tags can also contain [attributes](https://www2.informatik.hu-berlin.de/~xing/Lib/Docs/jaxp/docs/tutorial/glossary.html%252525252525252523attribute) (additional information included as part of the tag itself) within the tag’s angle brackets. If you consider the information in question to be part of the essential material that is being expressed or communicated in the XML, put it in an element. For human-readable documents, this generally means the core content that is being communicated to the reader. For machine-oriented records formats, this generally means the data that comes directly from the problem domain. If you consider the information to be peripheral or incidental to the main communication, or purely intended to help applications process the main communication, use attributes. The following example shows an email message structure that uses attributes for the to, from, and subject fields:

<message to=you@yourAddress.com [from=me@myAddress.com](mailto:from=me@myAddress.com)

subject="XML Is Really Cool">

<text>

How many ways is XML cool? Let me count the ways...

</text>

</message>

One really big difference between XML and HTML is that an XML document is always constrained to be [well-formed](https://www2.informatik.hu-berlin.de/~xing/Lib/Docs/jaxp/docs/tutorial/overview/4_design.html). There are several rules that determine when a document is well-formed, but one of the most important is that every tag has a closing tag. So, in XML, the </to> tag is not optional. The <to> element is never terminated by any tag other than </to>.

**Note**

Another important aspect of a well-formed document is that all tags are completely nested. So you can have <message>..<to>..</to>..</message>, but never <message>..<to>..</message>..</to>.

An XML Schema is a language for expressing constraints about XML documents. There are several different schema languages in widespread use, but the main ones are Document Type Definitions (DTDs). It defines the legal building blocks of an XML document. It also defines the document structure with a list of legal elements and attributes.

**XML Comments**

XML comments look just like HTML comments:

<message to=you@yourAddress.com [from=me@myAddress.com](mailto:from=me@myAddress.com)

subject="XML Is Really Cool">

<!-- This is comment -->

<text>

How many ways is XML cool? Let me count the ways...

</text>

</message>

To complete this introduction to XML, note that an XML file always starts with a [prolog](https://www2.informatik.hu-berlin.de/~xing/Lib/Docs/jaxp/docs/tutorial/glossary.html%252525252525252523prolog). The minimal prolog contains a [declaration](https://www2.informatik.hu-berlin.de/~xing/Lib/Docs/jaxp/docs/tutorial/glossary.html%252525252525252523declaration) that identifies the document as an XML document, like this:

<?xml version="1.0"?>

The declaration may also contain additional information, like this:

<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

* **version**: Identifies the version of the XML markup language used in the data. This attribute is not optional.
* **encoding**: Identifies the character set used to encode the data. “ISO-8859-1” is “Latin-1”, the Western European and English language character set. (The default is compressed Unicode: UTF-8.)
* **standalone**: Tells whether or not this document references an external entity or an external data type specification. If there are no external references, then “yes” is appropriate.

**Why is XML Important?**

It is important because it allows the flexible development of user-defined document types, which means that it provides a persistent, robust, non-proprietary, and verifiable file format which can be used for the storage and transmission of data for both on and off the Web. In addition, XML:

* provides plain text: plain text makes it readable;
* provides data identification: by use of tags, data can be identified;
* provides styleability: using XSLT (eXtensible StyLe Sheet), data can be made in a presentable form;
* is easily processed (XML parsers, as well as well-formed parsers);
* is hierarchical (through nested tags).

**How can you use XML?**

There are several basic ways to make use of XML:

* Document-driven programming, where XML documents are containers that build interfaces and applications from existing components
* Archiving: the foundation for document-driven programming, where the customized version of a component is saved (archived) so it can be used later
* Binding, where the DTD or schema that defines an XML data structure is used to automatically generate a significant portion of the application that will eventually process that data

**Pros and Cons of XML**

Some of the pros and cons of XML are explained below.

* Pros
* Readable and editable by developers
* Error checking by means of schema and DTDs
* Can represent complex hierarchies of data
* Unicode gives flexibility for international operation
* Plenty of tools in all computer languages for both creation and parsing
* Cons
* Bulky text with low payload/formatting ratio (but can be compressed)
* Both creation and client-side parsing are CPU intensive
* Common word processing characters are illegal (MS Word “smart” punctuation, for example)
* Images and other binary data require extra encoding

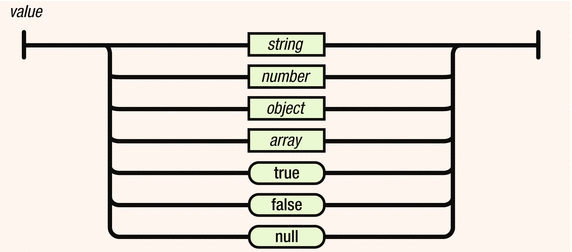
**What is JSON?**

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. Conventions used by JSON are known to programmers, which include those with knowledge of C, C++, Java, Python, Perl, etc.

* The format was specified by Douglas Crockford.
* It was designed for human-readable data interchange.
* It has been extended from the JavaScript scripting language.
* The filename extension is .json.
* JSON Internet media type is application/json.
* JSON is easy to read and write.
* JSON is language-independent.

**JSON Syntax**

In this section we will discuss what JSON’s basic data types are and syntax. Figure [3-1](#bookmark) shows basic data types of the JSON.



***Figure 3-1.*** *Basic data types*

**Strings**

Strings are enclosed in double quotes, and can contain the usual assortment of escaped characters.

**Numbers**

Numbers have the usual C/C++/Java syntax, including exponential (E) notation. All numbers are decimal—no octal or hexadecimal.

**Objects**

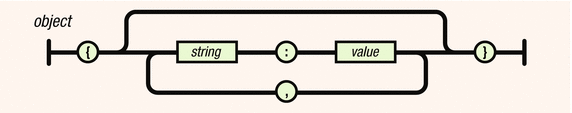
An object is an unordered set of a name/value pair. The pairs are enclosed within braces ({ }).

Example:

{ "name": "html", "years": 5 }

Pairs are separated by commas. There is a colon between the name and the value.

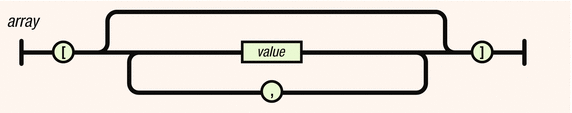
The syntax of a JSON object is shown in Figure [3-2](#bookmark1).



***Figure 3-2.*** *JSON object*

**Arrays**

An array is an ordered collection of values. The values are enclosed within brackets. The syntax of JSON arrays is shown in Figure [3-3](#bookmark2).



***Figure 3-3.*** *JSON arrays*

**Booleans**

It can have either true or false values.

**Null**

The value is that it’s empty.

**Why is JSON Important?**

There is a reason why JSON is becoming very popular as a data exchange format (more important than it being less verbose than XML): programmers are sick of writing parsers! But “wait,” you say. “Surely there are XML parsers available for you to use so that you don’t have to roll your own.” Yes, there are. But while XML parsers handle the low-level syntactic parsing of XML tags, attributes, etc., you still need to walk the DOM tree or, worse, build one yourself with nothing but a SAX parser (Objective-C iPhone SDK I’m looking at you!). And that code you write will of course depend on whether the XML you need to make sense of looks like this:

1 <person first-name="John" last-name="Smith"/>

or this:

1 <person>

2 <first-name>John</first-name>

3 <last-name>Smith</last-name>

4 </person>

or this:

1 <object type="Person">

2 <property name="first-name">John</property>

3 <property name="last-name">Smith</property>

4 </object>

or any of the myriad of other ways one can conceive of expressing the same concept (and there are many). The standard XML parser does not help you in this regard. You still need to do some work with the parse tree.

Working with JSON is a different, and superior, experience. First, the simpler syntax helps you avoid the need to decide between many different ways of representing your data (as we saw above with XML), much less which rope to hang yourself with. Usually there is only one straightforward way to represent something:

1 { "first-name" : "John",

2 "last-name" : "Smith" }

**How can you use JSON?**

The following discusses how you can use JSON.

* It is used while writing JavaScript-based applications that include browser extensions and websites.
* JSON format is used for serializing and transmitting structured data over a network connection. It is primarily used to transmit data between a server and web applications.
* Web services and APIs use JSON format to provide public data.

**Pros and Cons of JSON**

The following are pros and cons of JSON:

Pros:

* Easy to read/write/parse
* Reasonably succinct (compared with XML, for instance)
* Common “standard” with many libraries available

Cons:

* Not as light as binary formats
* Can’t use comments
* It’s “encapsulated,” meaning that you can’t readily stream/append data, but have to break it up into individual objects. XML has the same problem, whereas CSV does not.
* Difficult to describe the data you’re presenting (easier with XML)
* Unable to enforce, or validate against, a structure/schema

**XML - JSON Comparison**

This section compares XML and JSON based upon different properties.

***Table 3-1.*** *XML - JSON comparison*

|  |  |  |
| --- | --- | --- |
| Property | XML | JSON |
| Simplicity | XML is simple, human-readable | But JSON is much simpler than XML as well as human-readable |
| Self- Describing | Yes | Yes |
| Processing | XML is processed easily. | JSON is processed more easily because its structure is simpler. |
| Performance | Not optimized for performance due to tags | Faster than XML because of size |
| Openness | XML is open. | JSON is at least as open as XML, perhaps more so because it is not in the center of a corporate/political standardization struggle. |
| Object- Oriented | XML is document-oriented | JSON is data-oriented. JSON can be mapped more easily to object-oriented systems. |
| Interoperability | XML is interoperable. | JSON has the same interoperability potential as XML. |
| Internationalization | Supports unicode | Supports unicode |
| Extendability | XML is extensible. | JSON is not extensible because it does not need to be. JSON is not a document markup language, so it is not necessary to define new tags or attributes to represent data in it. |
| Adoption | XML is widely adopted by industry. | JSON is just beginning to become known. Its simplicity and the ease of converting XML to JSON makes JSON ultimately more adoptable. |

**Rest Service returning XML and JSON message**

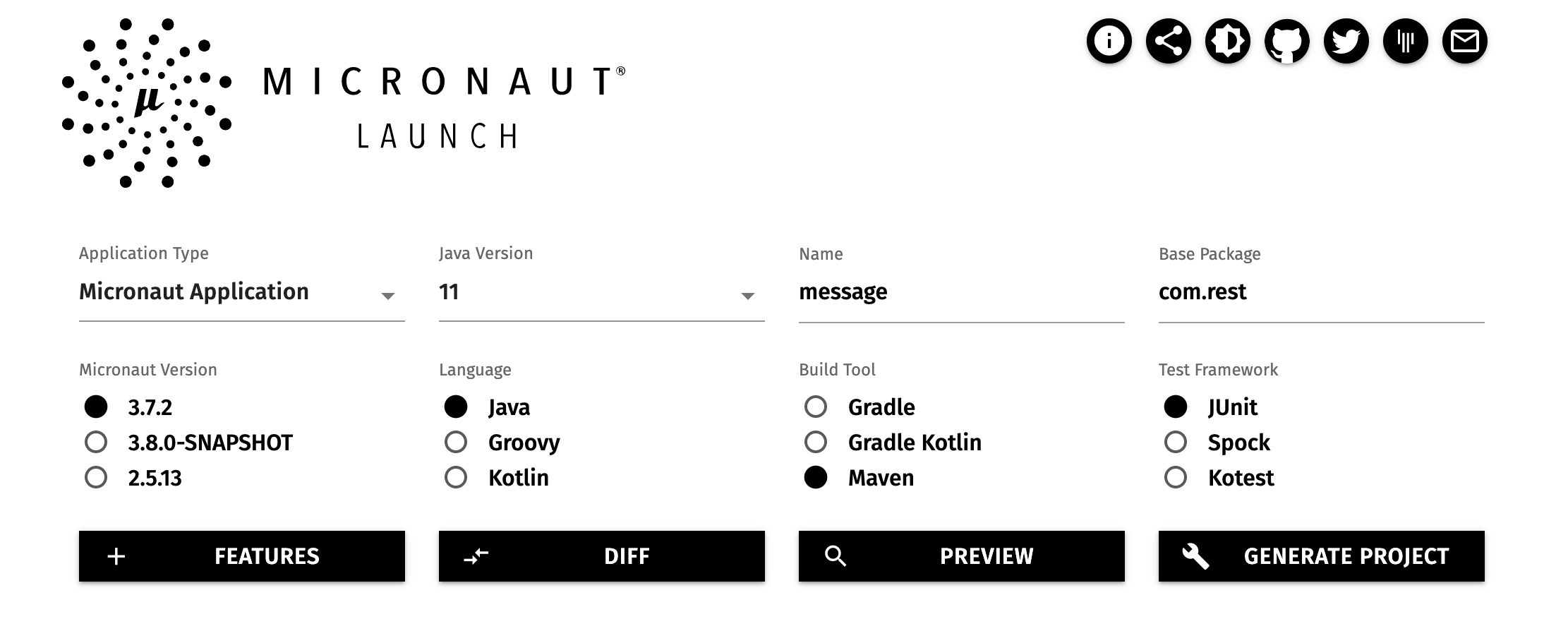
<https://micronaut.io/launch/>

As per screen select

*Application Type - Micronaut Application*

*Java Version - 11*

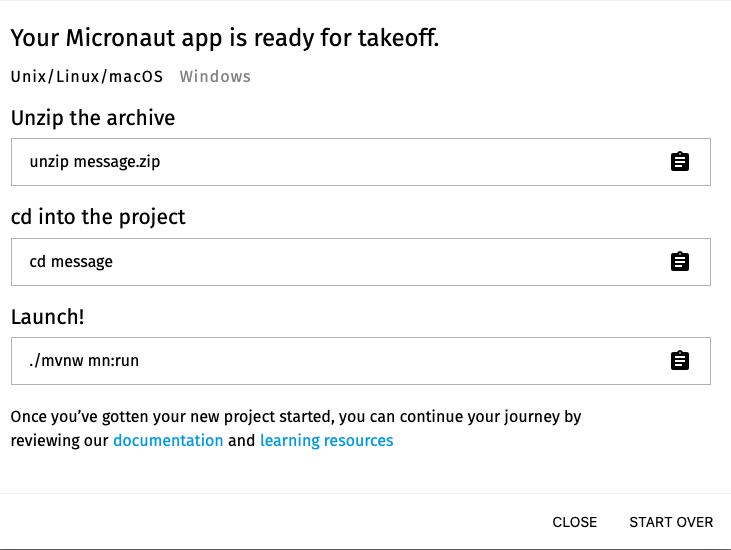
*Name - message*

*Package - com.rest*

*Build Tool : maven*

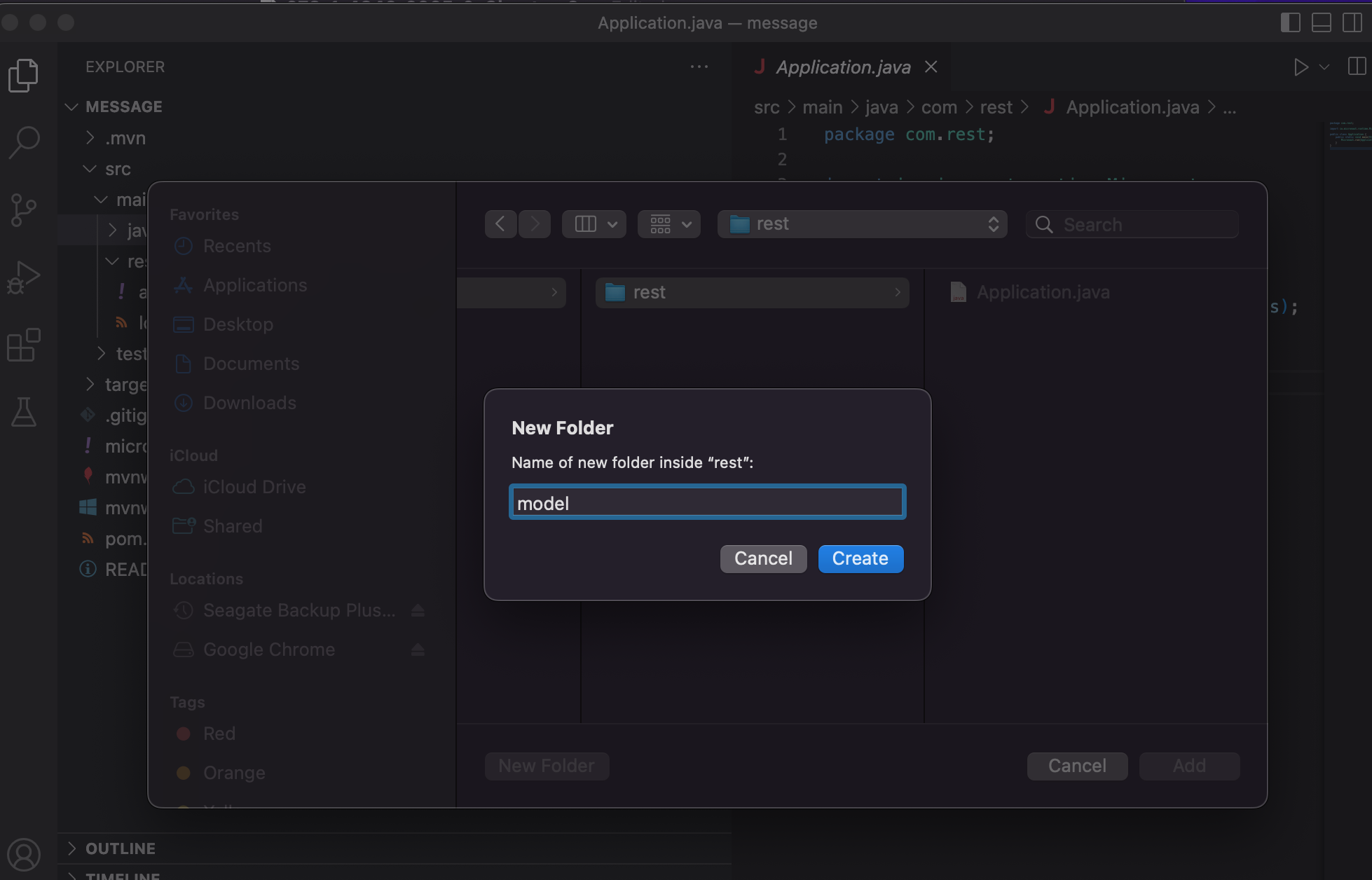
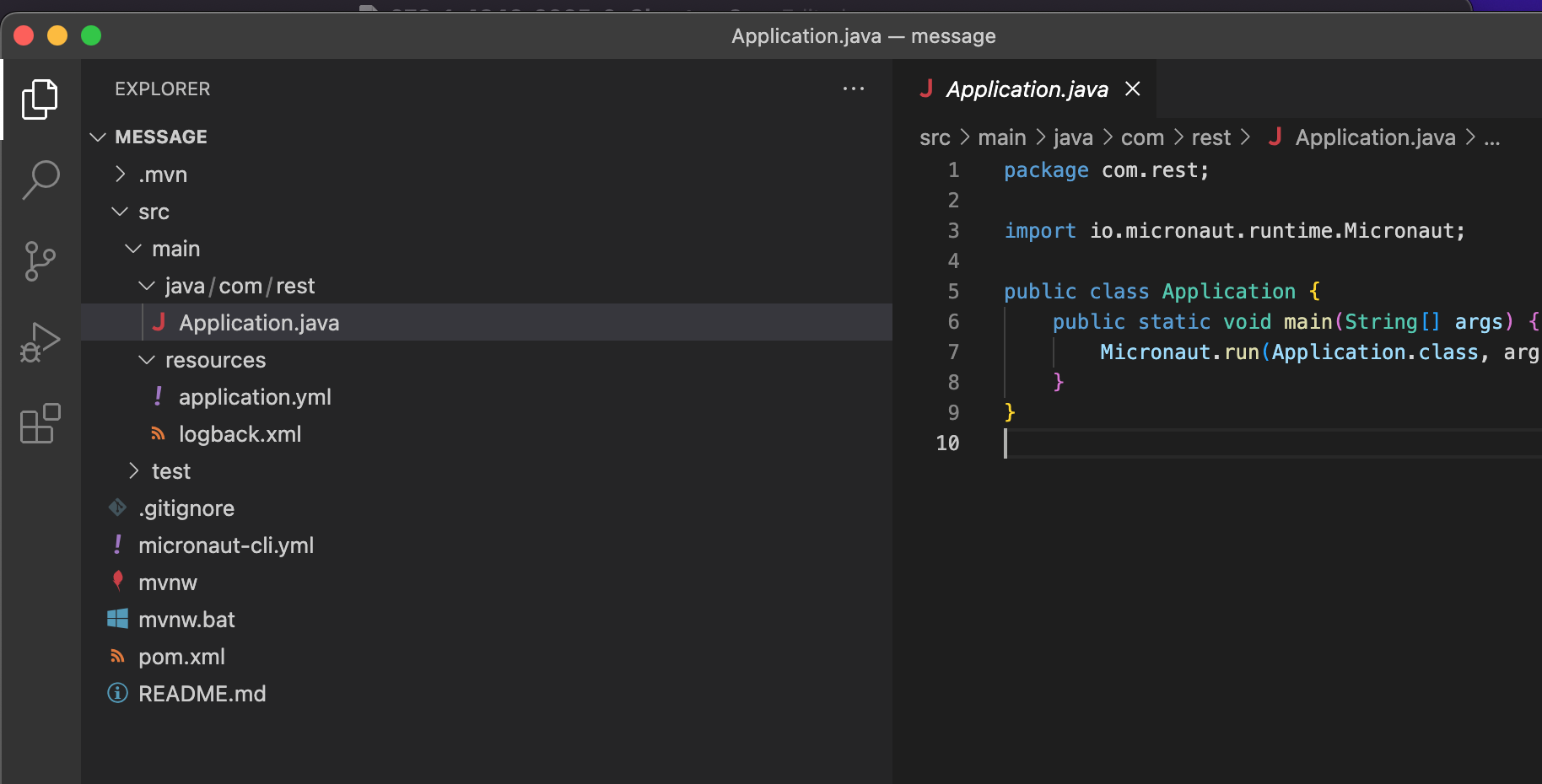
*Press “Generate Project”*

*Open message in Visual Source Code*

**

*Now add new folder model by selecting File->Add new folder and navigating to code generated by Micronaut.*

*Then create new domain class Message*

**

*Code below creates a domain object Message with an attribute message. Getter and setter methods are created in IDE.*

*package com.rest.model;*

*import javax.validation.constraints.NotNull;*

*public class Message {*

*@NotNull*

*private String message;*

*public String getMessage() {*

*return message;*

*}*

*public void setMessage(String message) {*

*this.message = message;*

*}*

*}*

*~*

*Now create a new folder controller after navigating to message code generated by Micronaut.*

*Code below exposes two endpoints*

1. *message/xml for getting message attribute value in XML*
2. *Message/json for getting message attribute value in JSON format.*

*Create Controller:*

*package com.rest.controllers;*

*import com.rest.model.Message;*

*import io.micronaut.http.annotation.Get;*

*import io.micronaut.http.annotation.Controller;*

*import io.micronaut.http.HttpResponse;*

*import io.micronaut.http.MediaType;*

*import io.micronaut.http.annotation.Produces;*

*@Controller("/message") // <2>*

*public class MessageController {*

*@Produces(MediaType.TEXT\_XML)*

*@Get("/xml")*

*public HttpResponse<?> messageXml() {*

*Message message = new Message();*

*message.setMessage("Hello from Micronaut");*

*final String xml = encodeAsXml(message);*

*return HttpResponse.ok(xml).contentType(MediaType.APPLICATION\_XML\_TYPE);*

*}*

*@Produces(MediaType.TEXT\_JSON)*

*@Get("/json")*

*public HttpResponse<?> messageJson() {*

*Message message = new Message();*

*message.setMessage("Hello from Micronaut");*

*return HttpResponse.ok(message);*

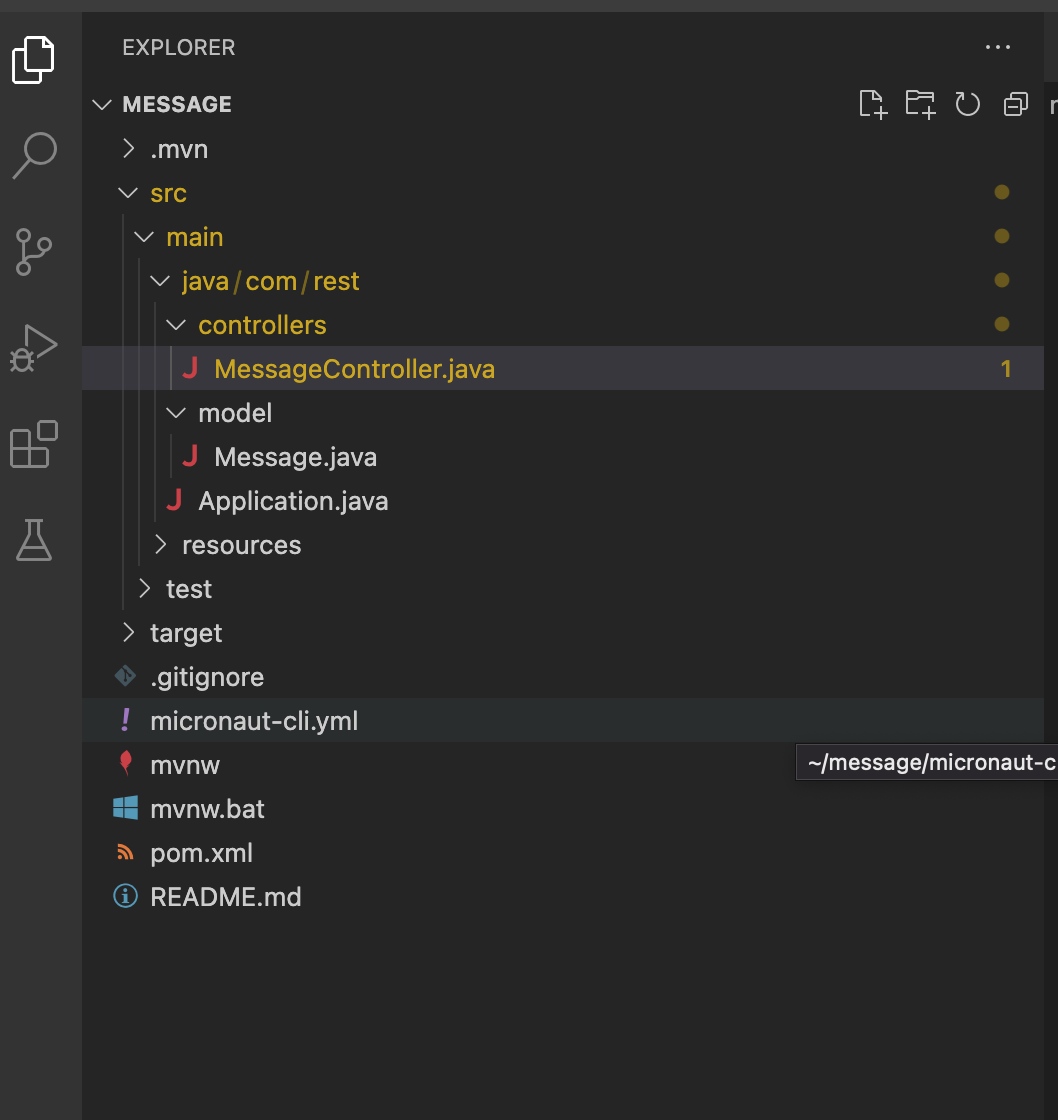
*}*

*private String encodeAsXml(final Message message) {*

*return String.format("<message>%s</message>", message.getMessage());*

*}*

*}*

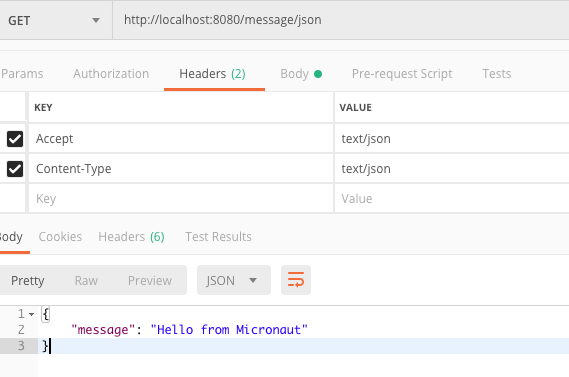
**

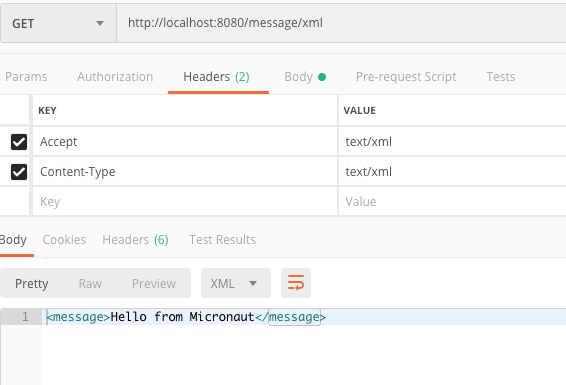
*You should have files as shown above.*

*Run app in IDE using Run -> Run without Debugging*

*You could also run in IDE by selecting Application.java and then right clicking on it.*

*Using POSTMAN as per screen-shot view JSON and XML response of message*

**

**

To Review

In this chapter we reviewed messaging using XML and JSON formats and compared them. Then developed APIs to return XML and JSON responses from a Micronaut app.