Side-By-Side: CORE, Python and Spring-Boot

We are going to do two thing in here:

1. Compare three trending language side by side.
2. Do performance testing for these three.

1. Tools and setup

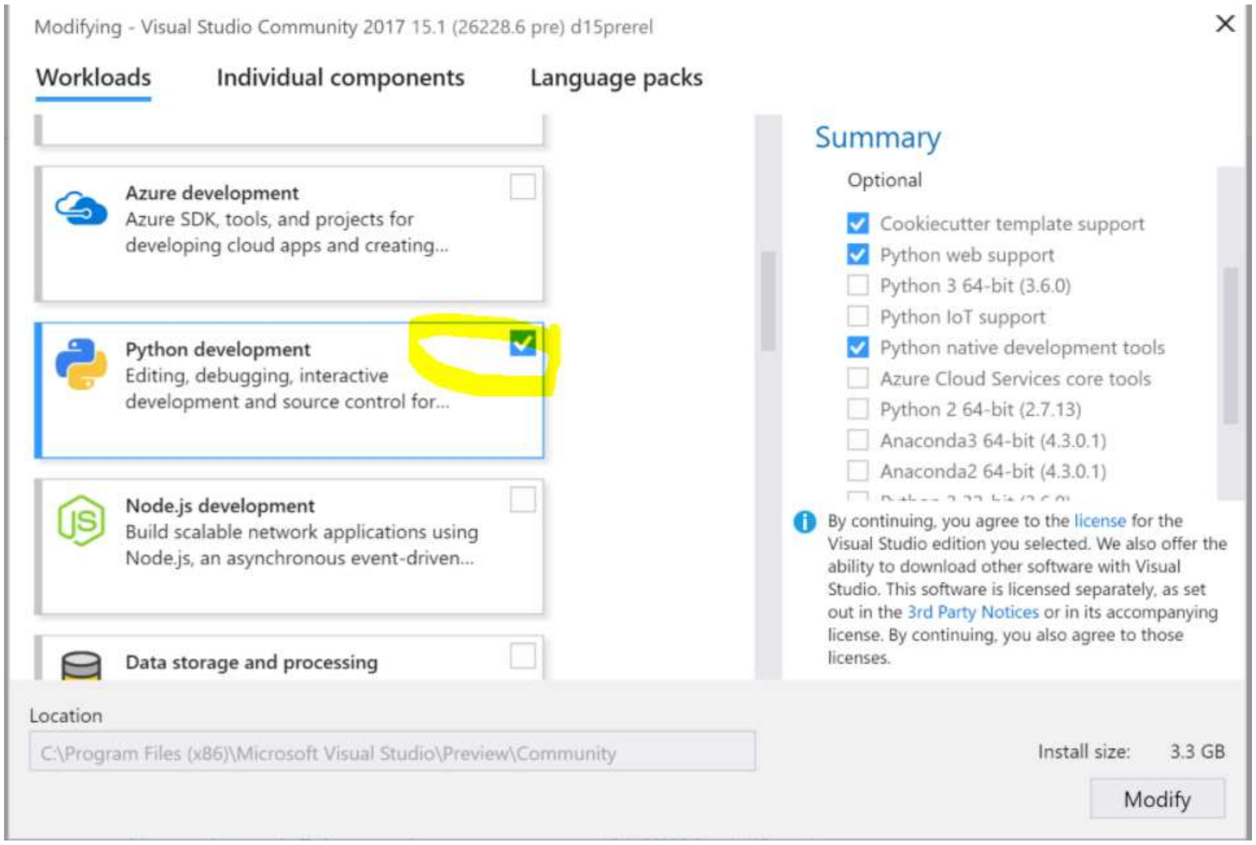
Let's begin with developer environment setup, just for simplicity and controlling g of scope I am not covering PostgressSQL installation.

a. .NET CORE

Install Visual Studio 2017, it take care of everything by it's own.

b. Python

Microsoft's Flagship VS2017 comes with an inbuilt option for Python. Checked option to install same and update setup and pretty much it will do all things for you.



Modifying - Visual Studio Community 2017 15.1 (26228.6 pre) d15prerel

Azure development — Azure SDK, tools, and projects for developing cloud apps and creating... [ ]

Python development — Editing, debugging, interactive development and source control for... [✓]

Node.js development — Build scalable network applications using Node.js, an asynchronous event-driven... [ ]

Data storage and processing [ ]

Summary

Optional

- [✓] Cookiecutter template support
- [✓] Python web support
- [ ] Python 3 64-bit (3.6.0)
- [ ] Python IoT support
- [✓] Python native development tools
- [ ] Azure Cloud Services core tools
- [ ] Python 2 64-bit (2.7.13)
- [ ] Anaconda3 64-bit (4.3.0.1)
- [ ] Anaconda2 64-bit (4.3.0.1)

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Location: C:\Program Files (x86)\Microsoft Visual Studio\Preview\Community

Install size: 3.3 GB

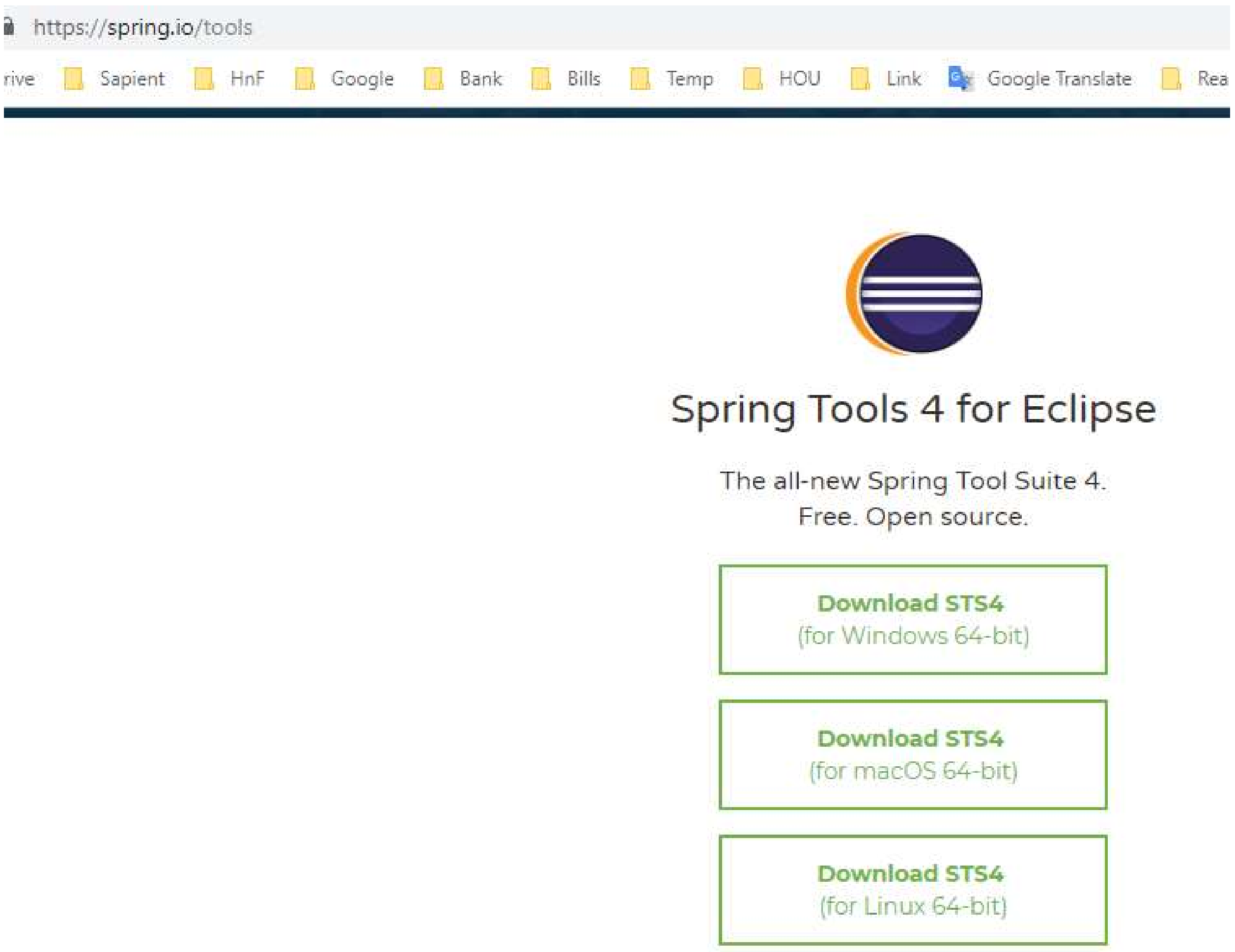
c. With no VS

- Download latest stable version from https://www.python.org/downloads/ and install same

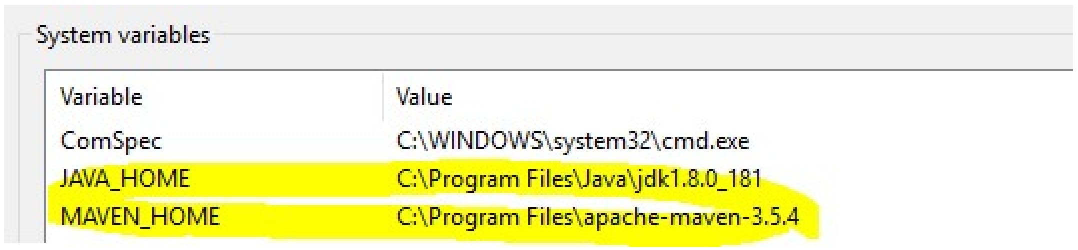
- After installation, open command prompt and type **pip –version**, this command should recognize and expect to see Python version installed

d. Spring-Boot

Get STS (Spring Tool Suite) it is a lighter version of Eclipse specially design for Spring development.


https://spring.io/tools
rive Sapient HnF Google Bank Bills Temp HOU Link Google Translate Rea
Spring Tools 4 for Eclipse
The all-new Spring Tool Suite 4.
Free. Open source.
Download STS4 (for Windows 64-bit)
Download STS4 (for macOS 64-bit)
Download STS4 (for Linux 64-bit)

Make sure JAVA_HOME and MAVEN_HOME path are setup correctly.



System variables

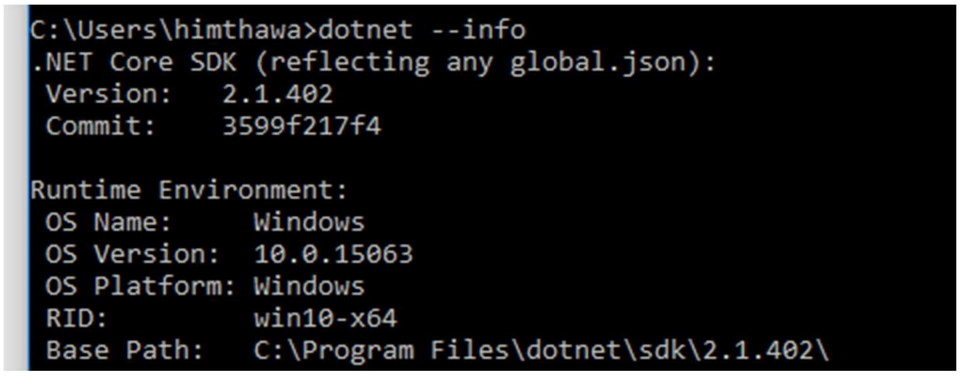
Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_181
MAVEN_HOME	C:\Program Files\apache-maven-3.5.4

This may differ as per your local path

2. Quick check

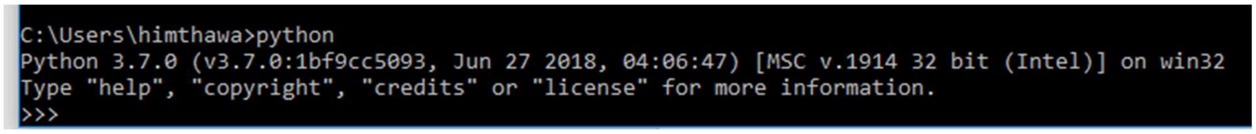
Before we begin with our development, it’s good idea to check if everything install and setup correctly. Open command prompt and run fallowing commands

a. .NET CORE



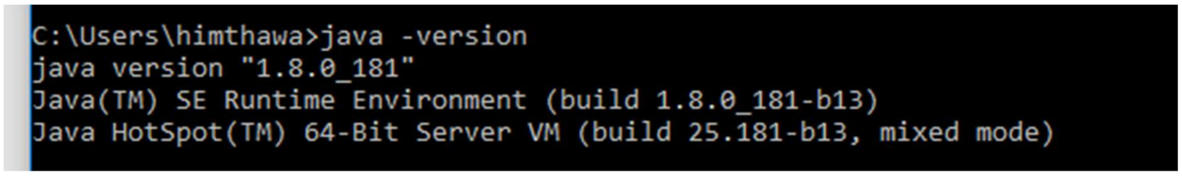
```
C:\Users\himthawa>dotnet --info
.NET Core SDK (reflecting any global.json):
 Version:   2.1.402
 Commit:    3599f217f4

Runtime Environment:
 OS Name:     Windows
 OS Version:  10.0.15063
 OS Platform: Windows
 RID:         win10-x64
 Base Path:   C:\Program Files\dotnet\sdk\2.1.402\
```

b. Python



```
C:\Users\himthawa>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

c. Spring-Boot

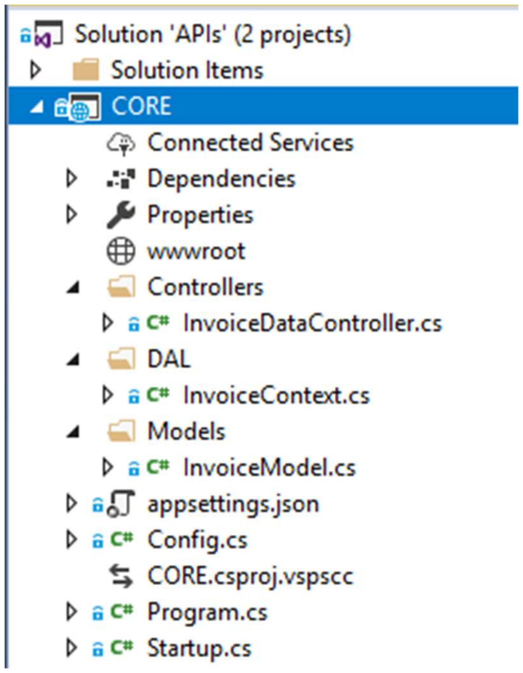


```
C:\Users\himthawa>java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

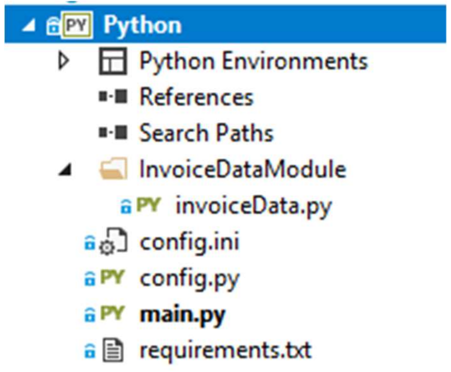
3. Solution structure

A typical bare-bone solution structure of API project. Further in this articles we are going to learn more about these files and folders and also compare among these three.

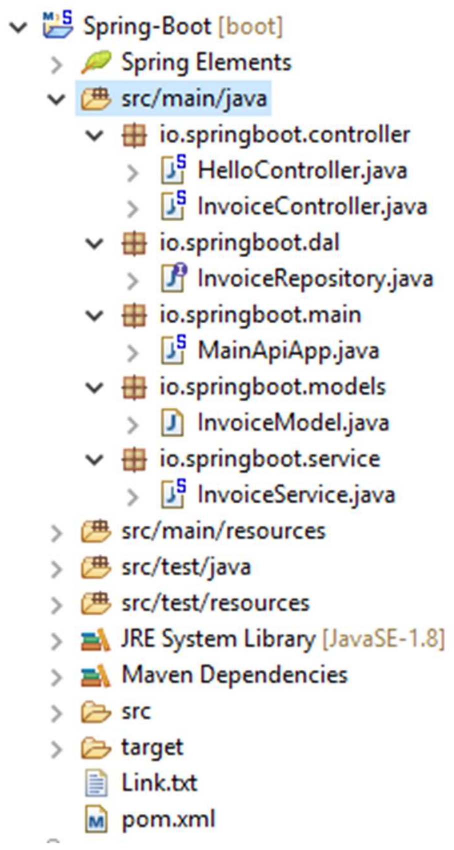
a. .NET CORE


Solution 'APIs' (2 projects)
Solution Items
CORE
Connected Services
Dependencies
Properties
wwwroot
Controllers
InvoiceDataController.cs
DAL
InvoiceContext.cs
Models
InvoiceModel.cs
appsettings.json
Config.cs
CORE.csproj.vspscc
Program.cs
Startup.cs

b. Python


Python
Python Environments
References
Search Paths
InvoiceDataModule
invoiceData.py
config.ini
config.py
main.py
requirements.txt

c. Spring-Boot



```
Spring-Boot [boot]
  Spring Elements
  src/main/java
    io.springboot.controller
      HelloController.java
      InvoiceController.java
    io.springboot.dal
      InvoiceRepository.java
    io.springboot.main
      MainApiApp.java
    io.springboot.models
      InvoiceModel.java
    io.springboot.service
      InvoiceService.java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  Link.txt
  pom.xml
```

4. Project dependencies

All these has some sort check list where it maintain what library/ package/ dll it need to run and they are smart enough to pull it from there repositories.

a. .NET CORE

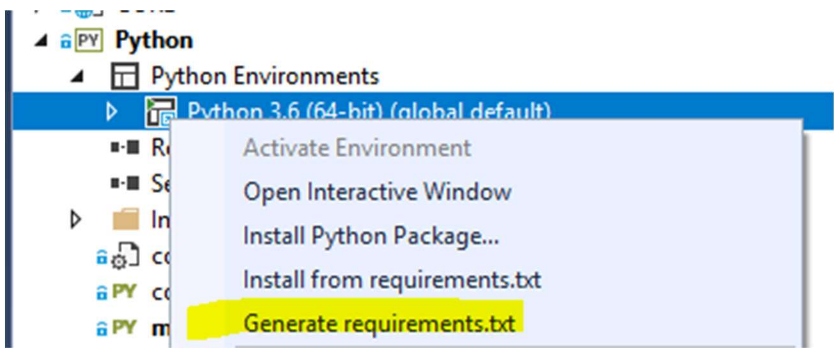
CORE.csproj ItemGroup section maintain list of dependency

```
 <ItemGroup>

   <PackageReference Include="Microsoft.AspNetCore.App" />

   <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.1.4">

     <PrivateAssets>all</PrivateAssets>

     <IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>

   </PackageReference>

   <PackageReference Include="Npgsql.EntityFrameworkCore.PostgreSQL" Version="2.1.2" />

   <PackageReference Include="Npgsql.EntityFrameworkCore.PostgreSQL.Design"
Version="1.1.1" />
```

```
</ItemGroup>
```

b. Python

requirements.txt list all dependency, only thing is you need to generate manually, simply right click on Python Environment and generate this file


Python
Python Environments
Python 3.6 (64-bit) (global default)
Activate Environment
Open Interactive Window
Install Python Package...
Install from requirements.txt
Generate requirements.txt

Same can be used to install dependency by selection **Install from requirments.txt**

c. Spring-Boot

pom.xml old and gold file from java world, which maintain all dependencies

```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-actuator</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>

	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-actuator</artifactId>
	</dependency>

	<dependency>
		<groupId>org.postgresql</groupId>
		<artifactId>postgresql</artifactId>
		<scope>runtime</scope>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-jpa</artifactId>
	</dependency>
	</dependencies>
```

5. Config file

Each of these support many types of config but will go with default and most common one.

a. .NET CORE

appsettings.json is new web.config, it is a Jason file which support direct config key-value and section and sub-section too. Will add our ConnectionString here.

```
  "ConnectionStrings": {
    "DefaultConnection":
"Server=localhost;Database=PocDB;Username=dbadmin;Password=dbadminpwd"
  }
```

You can read these config value using **Microsoft.Extensions.Configuration**. Update your **Startup.cs** constructor to inject dependency of `IConfiguration` and read `DefaultConnection`

```
 public IConfiguration Configuration { get; }

 public Startup(IConfiguration configuration)
 {
     Configuration = configuration;
 }



public void ConfigureServices(IServiceCollection services)
 {
     string connectionString =
Configuration.GetConnectionString("DefaultConnection");
     services.AddDbContext<InvoiceContext>(options =>
options.UseNpgsql(connectionString));
     services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
 }
```

b. Python

config.ini is default config file, it is key valve pare flat file where each line contain one key value and it also support sections.

```
[postgresql]
host=localhost
port=5432
database=PocDB
user=dbadmin
password=dbadminpwd
```

You can read these config value using `configparser`

```
   def __init__(self,comingFrom):
           print("__init__, comingFrom:",comingFrom)
           config = configparser.ConfigParser()
           config.sections()
           config.read("config.ini")

           #Global level variable
           self.DBServer = config['postgresql']['host']
           self.Port = config['postgresql']['port']
           self.Db = config['postgresql']['database']
           self.User = config['postgresql']['user']
```

```
                self.Password = config['postgresql']['password']
```

c. Spring-Boot

application.properties is default config file, it already has lots of thing baked in and it a properties like file where you can directly assign values.

```
server.port=8081
spring.datasource.url=jdbc:postgresql://localhost:5432/PocDB
spring.datasource.username=dbadmin
spring.datasource.password=dbadminpwd
spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

Nothing to read (at least for this articles) all value already assigned.

6. Connect with Data base

a. .NET CORE

We are using Entity Framework to connect with PostgresSQL Db

Create you **Model** and Entity object in **InvoiceModel.cs** decorate with correct attributes.

```
    [Table("invoice")]
    public class InvoiceModel
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public long record_id { get; set; }
        public string invoice_number { get; set; }
        public double amount { get; set; }
        public string invoice_description { get; set; }
        public string vendor_name { get; set; }
    }
```

Create DAL **InvoiceContext.cs** for by inheriting base

```
using CORE.Models;
using Microsoft.EntityFrameworkCore;

namespace CORE.DAL
{
    public class InvoiceContext : DbContext
    {
        public InvoiceContext(DbContextOptions<InvoiceContext> options) :
base(options)
        {
        }
        public DbSet<InvoiceModel> Invoice { get; set; }

    }
}
```

And that's it your basic CRUID operation is ready to go.

b. Python

We are not using ORM for Python, here is db connection

```
    def getData(self):
        dbConnetion = 
psycopg2.connect(host=self.DBServer,port=self.Port,user=self.User,password=self.Password,
database=self.Db)
        cursor = dbConnetion.cursor()
        sql = sql = "SELECT * FROM public.invoice"
        print("sql:",sql)
        rowCount = cursor.execute(sql)
        rows = cursor.fetchall()
                  jsonRows = []
```

c. Spring-Boot

We are using **JPA** to connect with PostgresSQL Db

Just like .NET CORE create Model in **InvoiceModel.java** decorate with correct attributes.

```
@Entity
@Table(name = "invoice")
public class InvoiceModel  implements Serializable{

	private static final long serialVersionUID = -3223451991002372807L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long record_id;

	private String invoice_number;

	private Double amount;

	private String invoice_description;

	private String vendor_name;
```

Create DAL **InvoiceRepository.java** for by inheriting base

```
@Repository
public interface InvoiceRepository extends CrudRepository<InvoiceModel, Long>
{

}
```

And that’s it your basic CRUID operation is ready to go.

7. End points

As these are going to be API we need to create end point to perform DB operations. To do this we are going to create controllers with some actions

a. .NET CORE

Create **InvoiceDataController.cs** which contain actions mapping with request type to perform db operations.

On **attribute** you can specify the Routing and different type of request these get invoke.

```
[Route("api/[controller]")]
[ApiController]
public class InvoiceDataController : ControllerBase
{
    private readonly InvoiceContext _context;

    public InvoiceDataController(InvoiceContext context)
    {
        _context = context;
    }

    // GET api/values
    [HttpGet]
    public ActionResult<IEnumerable<InvoiceModel>> Get()
    {
        return _context.Invoice.ToList();
    }

    // POST api/values
    [HttpPost]
    public void Post()
    {
        InvoiceModel invoice = new InvoiceModel { amount = 100, invoice_description =
"invoice_From CORE", invoice_number = "101", vendor_name = "vendor_name" };
        _context.Invoice.Add(invoice);
        _context.SaveChanges();
    }

    // DELETE api/values/5
    [HttpDelete("{id}")]
    public void Delete(int id)
    {
        InvoiceModel invoice = new InvoiceModel { record_id = id };
        _context.Invoice.Remove(invoice);
        _context.SaveChanges();
    }
          }
```

b. Python

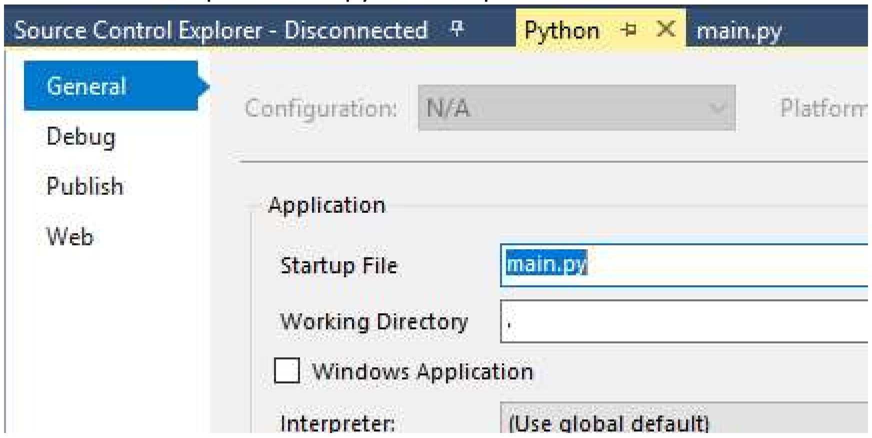
Create **invoiceData.py** which contain actions mapping with request type to perform db operations.

```
def on_get(self, req, resp):
    print("req.query_string:",req.query_string)
    jsonRows = self.getData()

    # Create a JSON representation of the resource
    resp.body = json.dumps(jsonRows, ensure_ascii=False)
    resp.status = falcon.HTTP_200

def on_post(self, req, resp):
    print("Insert data")
    self.insertData()
    resp.body = json.dumps("{result:Data saved.}", ensure_ascii=False)
    resp.status = falcon.HTTP_200
    def on_delete(self, req, resp,recordid):
        print("recordid:",recordid)
        print("req.query_string:",req.query_string)

        self.deleteData(recordid)
        resp.body = json.dumps("{result:Data delete.}", ensure_ascii=False)
                 resp.status = falcon.HTTP_200
```

main.py you can define the routing, like:

```
app.add_route('/invoicedata', invoiceData)
app.add_route('/invoicedata/{recordid}', invoiceData)
```

You need to setup this main.py as startup file



Configuration: N/A

Application

Startup File: main.py

Working Directory: .

☐ Windows Application

Interpreter: (Use global default)

c. Spring-Boot

Create **InvoiceController.java** which contain actions mapping with request type to perform db operations.

On **attribute** you can specify the Routing and different type of request these get invoke.

```
@CrossOrigin
@RestController
@RequestMapping("/invoice")
public class InvoiceController {

        private InvoiceService invescoService;

        @Autowired
        public InvoiceController(InvoiceService invescoService) {
                super();
                this.invescoService = invescoService;
        }

        @GetMapping("/all")
        public List<InvoiceModel> findAllInvoices() {
                return invescoService.findAllInvoices();
        }


        @RequestMapping(value = "/save", method = RequestMethod.POST)
        public boolean saveInvoice() {
                  InvoiceModel invoice = new InvoiceModel();
                 // invoice.setRecordID(1);
```

```
            invoice.setAmount(100.0);
            invoice.setInvoiceDescription("invoice_From Spring-Boot");
            invoice.setInvoiceNumber("101");
            invoice.setVendorName("vendor_name");


        return invescoService.addInvoices(invoice);
}


@RequestMapping(value = "/delete/{invoice_id}", method = RequestMethod.DELETE)
public boolean deleteSecurity(@PathVariable("invoice_id") Long invoice_id) {
        return invescoService.deleteInvoices(invoice_id);
}
```

8. Running as API

All got all setup it’s time for action.

a. .NET CORE

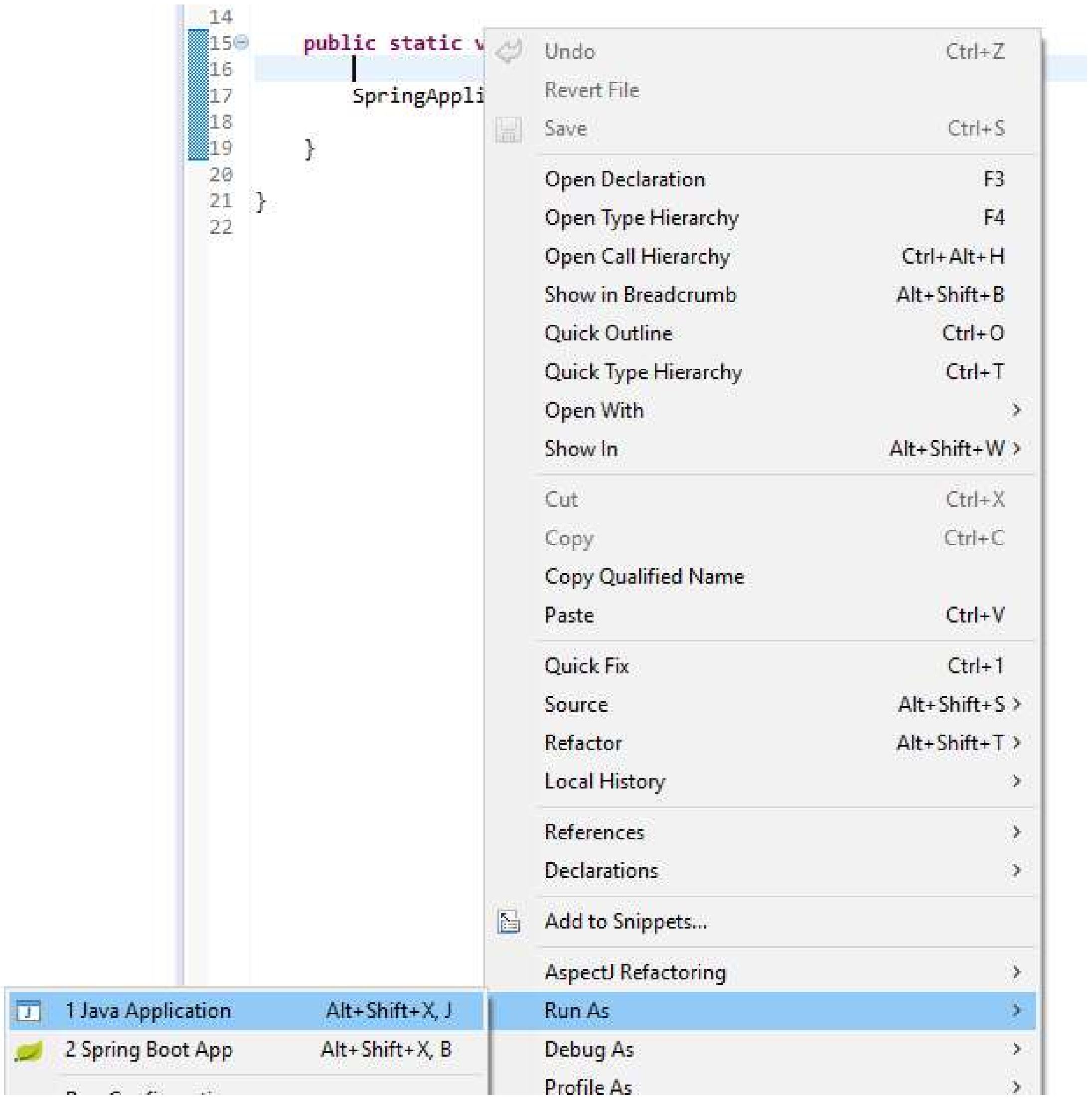
Just press **F5** and it launch IIS Express and host your API there at runtime.

b. Python

Same press **F5** it will open Python console and do the self-hosting of API there

c. Spring-Boot

You can run it as simple java app, it will launch Apachetop cat server and host API there at runtime.


```
    public static v
        |
        SpringAppli
    }
}
```

9. Invoking Action

All end point are ready to accept request, here are sample request. Just for shake of simplicity I am keeping POST call empty and creating new model in controllers to insert new data in DB.

1. .NET CORE

GET	http://localhost:63757/api/invoiceData
POST	http://localhost:63757/api/invoiceData
DELETE	http://localhost:63757/api/invoiceData/1

2. Python

GET	http://localhost:8080/invoicedata/
POST	http://localhost:8080/invoicedata/
DELETE	http://localhost:8080/invoicedata/2

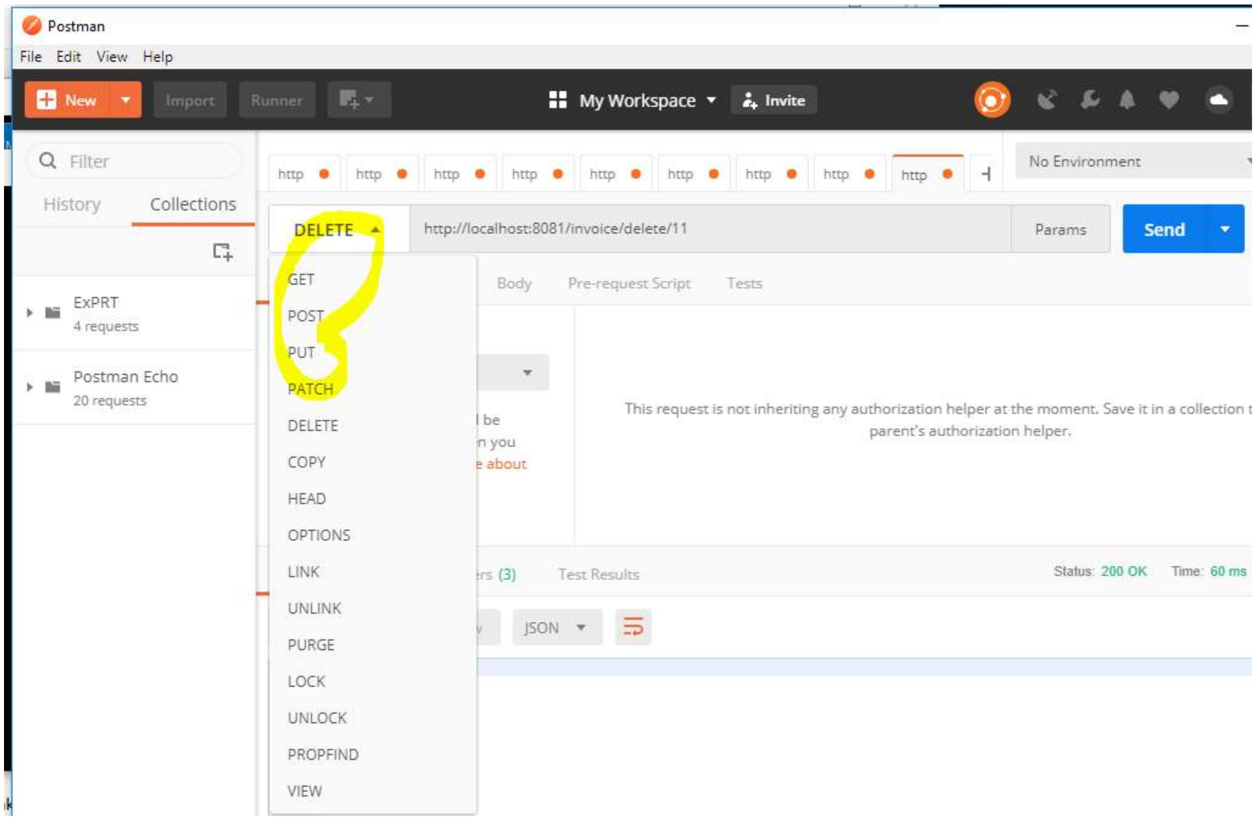
3. Spring-Boot

GET	http://localhost:8081/invoice/all

POST http://localhost:8081/invoice/save

DELETE http://localhost:8081/invoice/delete/7

I used **Postman** to call these service



http://localhost:8081/invoice/delete/11

This request is not inheriting any authorization helper at the moment. Save it in a collection t parent's authorization helper.

Status: 200 OK Time: 60 ms

10. Performance testing

That's my TODO item. I am planning to do the performance comparison of these three API

Here is scenario what I am thinking of:

1. Truncate DB table
2. Bombard Create API to create 50K rows – for load testing
3. Call API to get all 50K result - to get large dataset back
4. Loop thru 50K dataset and call delete API one-by-one
5. Repeat this for all three API
6. Find out which takes what time.

11. Reference

1. https://www.getpostman.com/
2. https://www.youtube.com/watch?v=msXL2oDexqw&list=PLqq-6Pq4lTTbx8p2oCgcAQGQyqN8XeA1x
3. https://docs.python.org/3.7/tutorial/index.html
4. https://falconframework.org/
5. https://docs.microsoft.com/en-us/visualstudio/python/managing-python-environments-in-visual-studio

Thanks & Regards,

Himanshu Thawait |¬Sapient Global Markets|Mobile: +1 (402) 547-6299 |e-mail: hthawait@sapient.com|www.sapient.com