PriceProvider Application

PriceProvider Application 1
Introduction 2
Technology 2
Design Approach 2
Installation Instructions 2
Swagger UI 3
Domain Model 3
Entities 5
Deletion of Stale Prices 6
Deletion of Stale Prices Process Flow 6
REST Endpoints 7
POST Endpoints 7
GET Endpoints 10
Recommended Improvements 11
Persistence Layer 11
Testing 11
Security 11
User-Interface 11
Enhanced Logging 11

Introduction

This document provides usage and design details for the PriceProvider application.

Technology

The PriceProvider application was written using the following technologies: -

1. Java version 11 (though it has been tested using Java 17)
2. Spring Boot
3. H2 database (for persistence via Spring JPA)
4. Swagger (providing a REST UI interface)

Design Approach

Upon reading the task specification, it felt appropriate to develop a RESTful application where various endpoints could be provided to the users to perform the various tasks. Coupled with Swagger-UI, these endpoints are easily accessible using a browser of their choice.

For quick and easy development, Spring Boot was the obvious choice to provides the RESTful API interfaces. The only other choice to make was how to persist the data. One or more concurrent maps of various types could be used but to maintain a relationship between vendors, instruments and prices it wouldn't be easy to keep all the maps in sync, particularly in a multi-user environment. I felt then that a database, with tables defined with referential integrity, would be a better approach and this allowed me to handle all the data joins very easily.

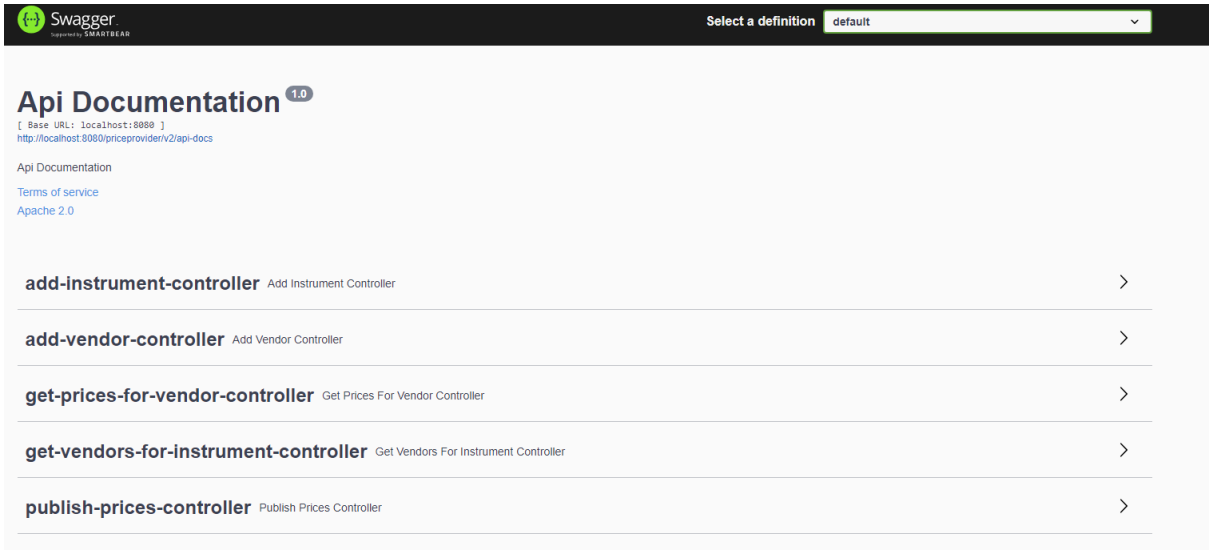
I used a simple H2 database for this purpose. With Spring JPA handling all the database interactions, it means that very little effort would be required in the future to replace the H2 database with something better.

Installation Instructions

1. Copy the PriceProvider folder and its contents to somewhere on your local computer.
2. Open an MSDOS-window by typing "cmd" in the Start bar.
3. CD to the location of the PriceProvider folder.
4. Type the command **priceprovider.bat**.
5. The application should launch displaying the familiar Spring Boot logo (as below).

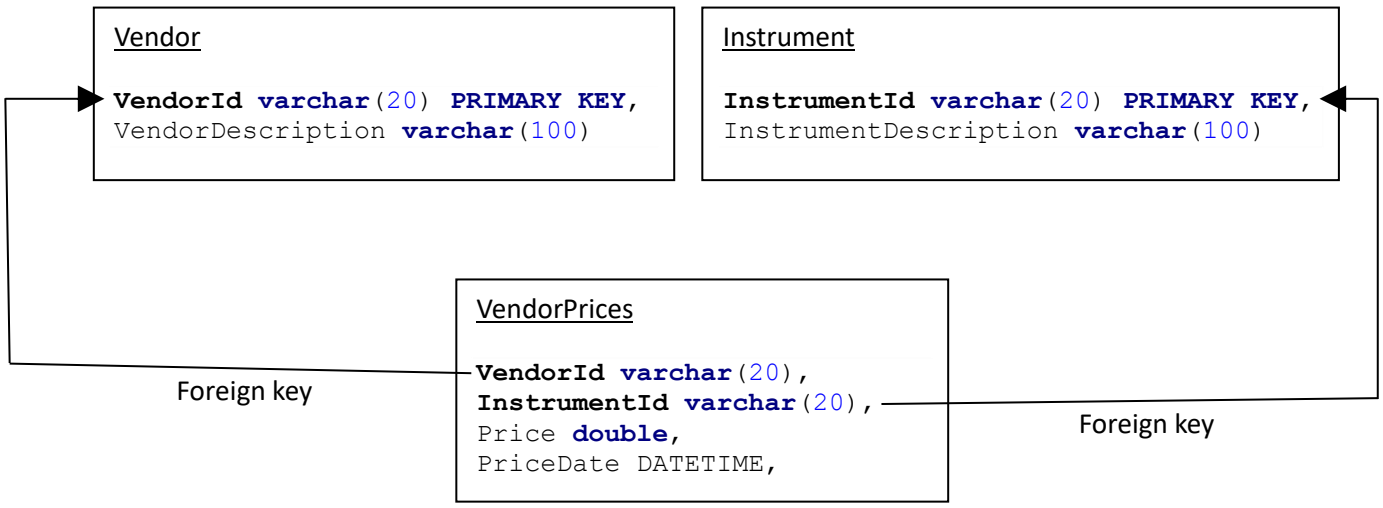
Swagger UI

Once the application has completed loading, which takes about 30 seconds, the user will be able to access a built-in user-interface. The PriceProvider application provides a Swagger front-end making it easy to invoke the five REST endpoints (more about those later). To use the Swagger UI, type http://localhost:8080/priceprovider/swagger-ui/#/ into a browser window. The following screen will appear: -



Domain Model

The PriceProvider application uses 3 database tables to persist data. The diagram below depicts the relationship between the 3 tables which store the Vendor, Instrument and price information:-



The VendorsPrices table drives two GET endpoints, namely “Get Prices For Vendor” and “Get Vendors For Instrument”. Note: Further information about these endpoints are covered in the section later in this document.

The VendorsPrices table adopts foreign keys, which references the Vendor and Instrument tables, to prevent insertion of data if either of the corresponding keys are missing.

The following data-rows have been preloaded into the database:-

SELECT * FROM Instrument;

InstrumentId	InstrumentDescription
In001	Instrument Description 001
In002	Instrument Description 002
In003	Instrument Description 003

(3 rows, 3 ms)

SELECT * FROM Vendor;

VendorId	VendorDescription
V001	Vendor Name 001
V002	Vendor Name 002
V003	Vendor Name 003
V004	Vendor Name 004

(4 rows, 0 ms)

SELECT * FROM VendorPrices;

VendorId	InstrumentId	Price	PriceDate
V001	In001	123.45	2024-06-24 01:12:53
V002	In001	7351.55	2024-06-23 11:52:33
V003	In001	662.32	2024-06-25 12:54:21
V003	In002	5552.23	2024-05-24 01:29:23
V002	In003	224.75	2024-06-24 04:34:37
V004	In003	3572.21	2024-05-23 08:55:58

(6 rows, 8 ms)

You can view the H2 database directly by using the following console: -

http://localhost:8080/priceprovider/h2-console/

User/Password credentials are admin/admin.

Entities

Though the application has 3 tables, I have created 6 entity classes to facilitate the endpoints in use within the PriceProvider application:

GetPricesForVendor

Used by the PricesForVendorRepository class and exposes the following fields:-

- Instrument Id
- Vendor Id
- Instrument Description
- Price
- Price Date

GetVendorsForInstrument

Used by the VendorsForInstrumentRepository class and exposes the following fields:-

- Instrument Id
- Vendor Id
- Vendor Description
- Price
- Price Date

Instrument

Used by the AddInstrumentRepository class for inserting and updating the Instrument table.

Vendor

Used by the AddVendorRepository class for inserting and updating the Instrument table.

VendorPrices

Used by the VendorPricesRepository class for inserting and updating the VendorPrices table.

InstrumentIdVendorId

This is the class used to represent the embedded key-value of Instrument and Vendor Id concatenated.

Deletion of Stale Prices

Every 5 minutes a scheduled process will automatically query the VendorPrices table to check for entries where the PriceDate field is older than the default value of 30 days.

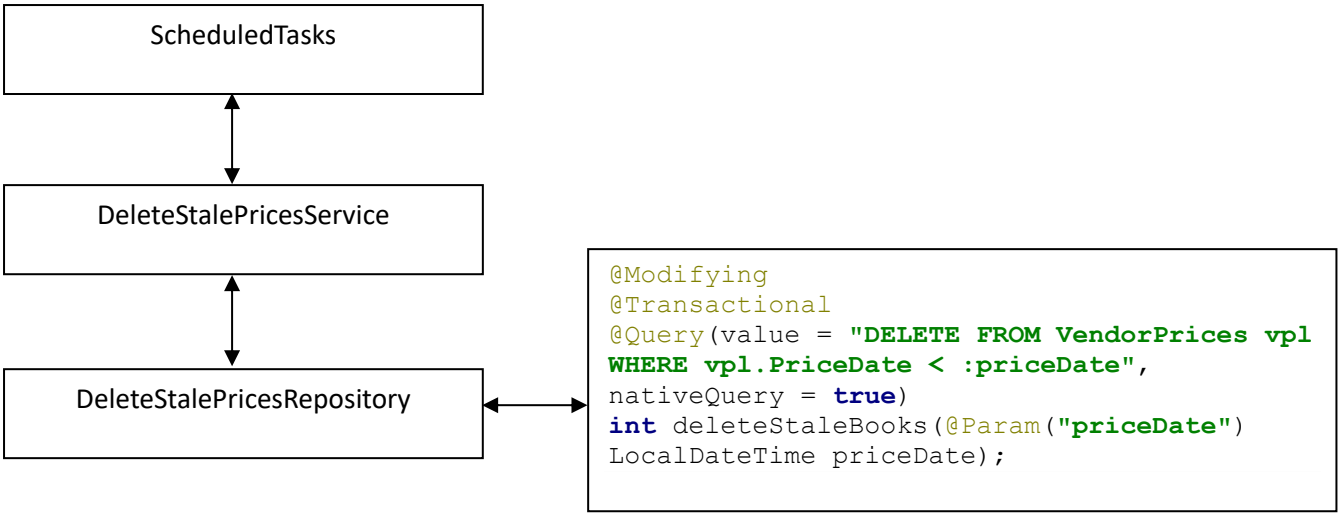
An entry in the logs will show when the process runs and how many rows (if any) have been deleted:

```
Removing stale prices older than 30 days
Removal process complete - stale prices removed: 2
```

The default number of 30 days may be amended to an alternative value by changing the property below in the application.properties file: -

```
## Application properties
no.days.before.price.is.stale=30
```

Deletion of Stale Prices Process Flow



REST Endpoints

There are 5 REST Controller endpoints, accessible via the Swagger UI (or curl, etc): -

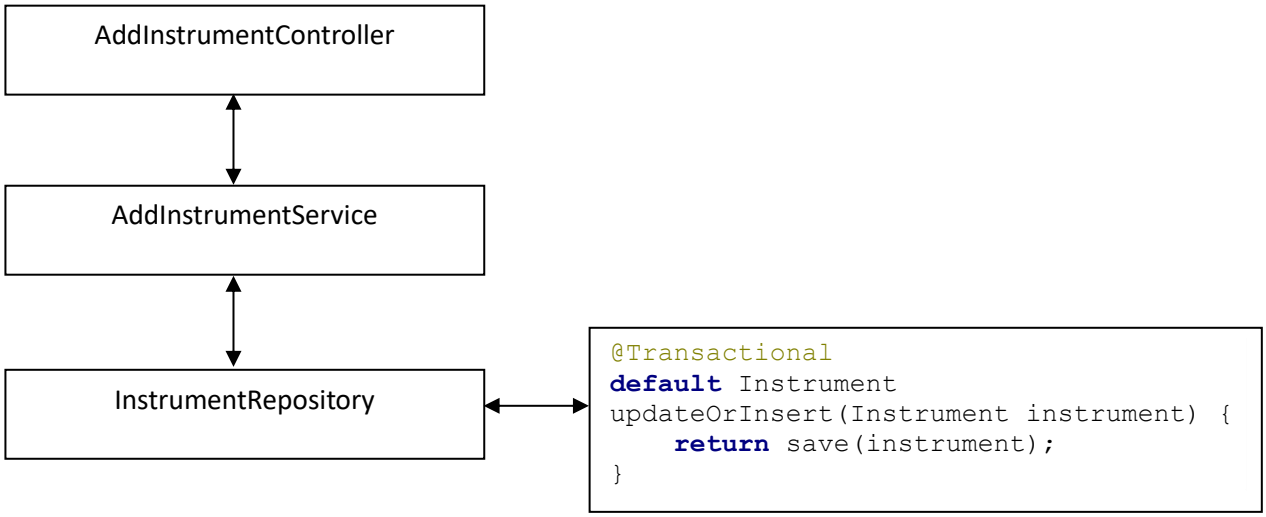
POST Endpoints

Add Instrument (POST)

This endpoint allows you to add an instrument and its description into the **Instrument** table, via the json structure below. The Instrument must exist in the Instrument table if you wish to reference it in the Publish Prices endpoint. Adding an instrument that already exists will simply replace the original contents with the new values provided.

```
{
  "instrumentDescription": "string",
  "instrumentId": "string"
}
```

Add Instrument Process Flow

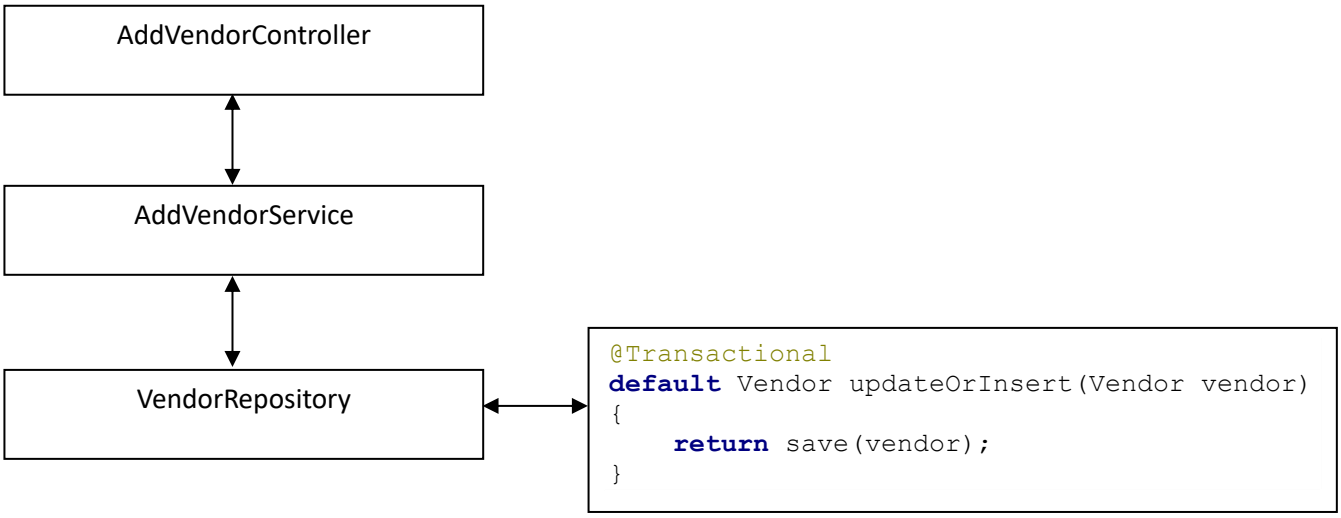


Add Vendor (POST)

This endpoint provides a way for you to add a Vendor and its description into the **Vendor** table, via the json structure below. The Vendor must exist in Vendor table if you wish to reference it in the Publish Prices endpoint. Adding a vendor that already exists will simply replace the original contents with the new values provided.

```
{
  "vendorDescription": "string",
  "vendorId": "string"
}
```

Add Vendor Process Flow



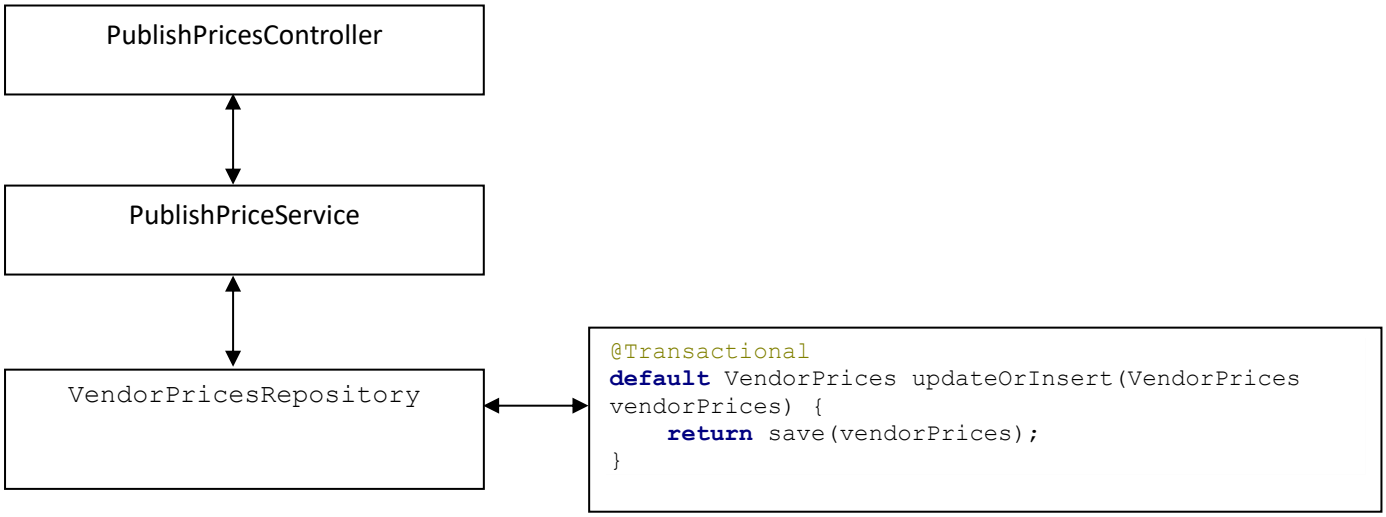
Publish Prices (POST)

This endpoint allows a price to be inserted into the **VendorPrices** table for the supplied Instrument and Vendor. Both the Instrument and Vendor must exist otherwise the publish will fail. A “Price Date” is also required. It defaults to the current Date & Time but any valid DateTime value may be used. The json structure below is required for this endpoint.

```
{
  "instrumentId": "string",
  "price": 0,
  "priceDate": "2024-07-05T19:17:28.951Z",
  "vendorId": "string"
}
```

Publishing a price for an Instrument and Vendor that exists already will simply replace the original contents with the new values provided.

Publish Prices Process Flow

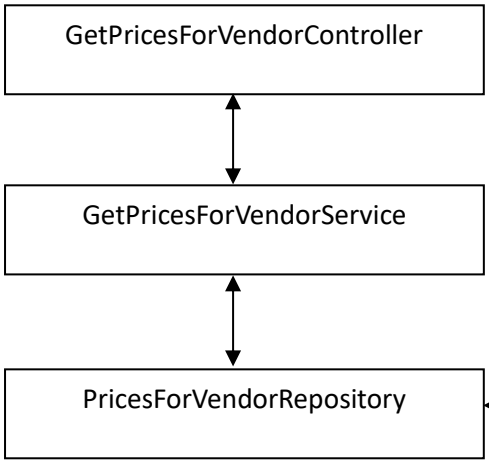


GET Endpoints

Get Prices For Vendor (GET)

This endpoint will display a list of all the instruments and their prices for the provided vendor. A parameter of "vendorId" is required.

Get prices For Vendor Process Flow

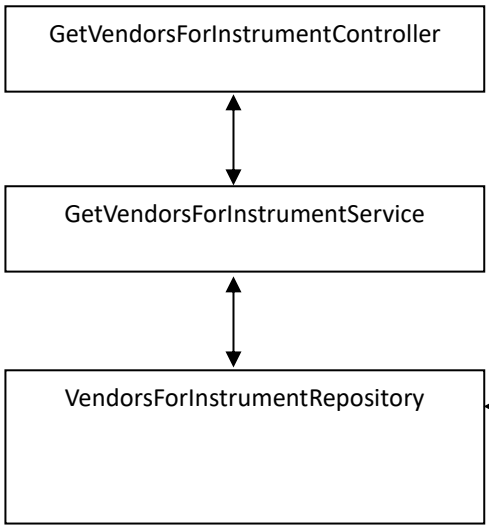



```
@Query(value = "SELECT vpl.InstrumentId,
vpl.VendorId, InstrumentDescription, Price,
PriceDate " +
"FROM VendorPrices vpl " +
"INNER JOIN Instrument i " +
"ON i.InstrumentId = vpl.InstrumentId " +
"WHERE vpl.VendorId = :vendorId",
nativeQuery = true)
List<GetPricesForVendor>
getPricesForVendor(String vendorId);
```

Get Vendors For Instrument (GET)

This endpoint will display a list of all the vendors and their prices for the provided instrument. A parameter of "instrumentId" is required.

Get Vendors For Instrument Process Flow




```
@Query(value = "SELECT vpl.InstrumentId,
vpl.VendorId, VendorDescription, Price,
PriceDate " +
"FROM VendorPrices vpl " +
"INNER JOIN Vendor v ON vpl.VendorId =
v.VendorId " +
"WHERE vpl.InstrumentId = :instrumentId",
nativeQuery = true)
List<GetVendorsForInstrument>
getVendorsForInstrument(String
instrumentId);
```

Recommended Improvements

Persistence Layer

The use of an H2 database for this project was very quick and easy to get up and running. However, it would not be recommended for use in a production environment. Its performance is acceptable for a small number of records but is very slow with larger volumes. Secondly, it would not be ideal for an environment where both updates and querying are happening concurrently, as I understand that H2 locks the entire table during a write operation which could lead to deadlocks.

On a positive note, Spring JPA was used to handle all the database operations within the PriceProvider application making the code reasonably database-agnostic. Therefore, the task of replacing H2 with something better should be a fairly painless experience.

Testing

Although I wrote a lot of unit tests to cover the basic happy/sad paths of each of the classes, more tests could be written to provide better coverage for some of the edge-cases that would undoubtably occur in a production environment.

I would also recommend writing a series of stress-testing scripts to see how the PriceProvider application copes with large numbers of read and write requests being sent simultaneously.

Security

The application currently has no security in place so it would be nice to provide different user roles. For example, some users could have “administrator” access allowing them to create new Vendors and Instruments. Meanwhile, others could have a “publishers” role giving them the ability to publish new prices. Finally, a “read-only” role would allow users to access the “GET” endpoints only.

User-Interface

One other area for improvement would be to provide a better user-interface. The Swagger-UI is fine for testing purposes but with a proper front-end, we could add improved validation and error-handling and a richer UI experience overall.

Enhanced Logging

Finally, given more time, I would like to add some more logging to the process. Although some logging exists for the Deletion of Stale Prices process, I would add logging for all of the endpoints in the process.