Lesson 16: Math and Date Functions

Created by Adriaan Klue, last modified by Jacques Marais on Sep 13, 2018

> *As a Farmer, I want to view a customizable table comparing stock prices at different shops.*

Lesson Outcomes

This lesson demonstrates using a combination of operators on primitives and `Math` and `Date` built-in functions to do calculations more complex compared to that of previous lessons.

Scenario

We want to provide farmers with a price comparison tool. However, we have no guarantee that the stock prices at the various shops are current, because shop owners are under no obligation to do regular updates. To make this tool more useful, we decided to add an *inflation-based estimate* for stock items that have not received recent price updates. To keep things simple the "inflation" will be based on user input and not historic data.

New & Modified App Files

`./model/objects/PriceEstimate.mez`

`./web-app/images/PriceCompare.png`

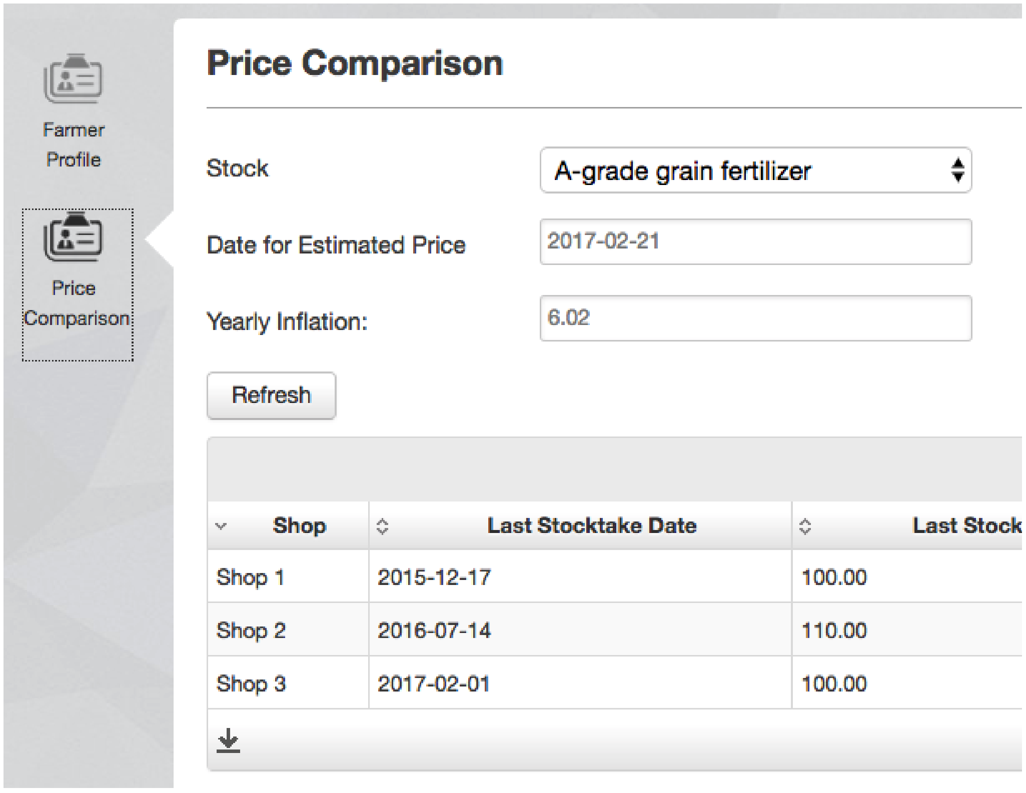
`./web-app/lang/en.lang`

`./web-app/presenters/farmer_ops/PriceComparison.mez`

`./web-app/views/farmer_ops/PriceComparison.vxml`

Required View Elements

Create a view with the following widgets:



Price Comparison

Stock	A-grade grain fertilizer
Date for Estimated Price	2017-02-21
Yearly Inflation:	6.02

Shop	Last Stocktake Date	Last Stock
Shop 1	2015-12-17	100.00
Shop 2	2016-07-14	110.00
Shop 3	2017-02-01	100.00

This is to be accessible by farmers (i.e. add a `<menuitem>` also).

Tutorial Aid

The stock update CSVs below has been added to speed up working through this tutorial. Upload them as per lesson 9. Select a different shop and past date for each.

stockUpdate_shop1.csv

stockUpdate_shop2.csv

stockUpdate_shop3.csv

Calculating the Estimate

The calculation will involve the following built-in functions we haven't used in this tutorial before:

- `Math:pow()`
- `Date:daysBetween()`

For other built-in functions available via the `Math` and `Date` namespaces, see Quick Reference.

Stock item prices will be read from the database and the user will provide the inflation rate and date for which to calculate the estimate. The calculation will involve the following steps.

1. For each shop, find the most recent price update for the selected stock item. (=C)

```

PriceEstimate[] getEstimates() {
   ...
   Shop[] shops = getAllShops();
   for (int i = 0; i < shops.length(); i++) {
      Shop currentShop = shops.get(i);
      StockUpdate lastUpdate = getLastUpdate(currentSh
      ...
}
```

```

StockUpdate getLastUpdate(Shop currentShop) {
   StockUpdate[] updates = StockUpdate:and(
         relationshipIn(shop, currentShop),
         relationshipIn(stock, selectedStock));
   if (updates.length() > 0) {
      updates.sortDesc("stocktakeDate");
      return updates.get(0);
   } else {
      return null;
   }
}
```

2. For each stock item's most recent price update, determine the time duration between the stocktake date and estimate date. (=t)

```
PriceEstimate[] getEstimates() {
   ...
   Shop[] shops = getAllShops();
   for (int i = 0; i < shops.length(); i++) {
      Shop currentShop = shops.get(i);
      StockUpdate lastUpdate = getLastUpdate(currentSh
      if (lastUpdate != null) {
         ...
         int durationDays = Date:daysBetween(lastUpdat
         ...
}
```

3. Calculate using the formula: S=C(1+r)^t , where S is the inflated value after t years and r the inflation rate (r has to be divided by 100 so that we have a decimal representative of the percentage value).

```

PriceEstimate[] getEstimates() {
   PriceEstimate[] result;
   Shop[] shops = getAllShops();
   for (int i = 0; i < shops.length(); i++) {
      Shop currentShop = shops.get(i);
      StockUpdate lastUpdate = getLastUpdate(currentSh
      if (lastUpdate != null) {
         PriceEstimate estimate = PriceEstimate:new();
         estimate.shopName = currentShop.name;
         estimate.lastUpdatedDate = lastUpdate.stockta
         estimate.lastUpdatedPrice = lastUpdate.price;
         int durationDays = Date:daysBetween(lastUpdat
         estimate.estimatedPrice = getEstimatedPrice(l
         result.append(estimate);
      }
   }
   return result;
}
decimal getEstimatedPrice(StockUpdate lastUpdate, int
   decimal d = durationDays; //convert to decimal to r
                             //rounded off before assi
   decimal t = d/365;
   decimal r = yearlyInflation/100;
   decimal latestPrice  = lastUpdate.price;
   return latestPrice * Math:pow(1+r, t);
}
```

The results will go into a table, so we create a new object for its collection source:

```
object PriceEstimate {
   string shopName;
   date lastUpdatedDate;
   decimal lastUpdatedPrice;
   decimal estimatedPrice;
}
```

The steps above will be repeated for each shop, so naturally we add an item to the collection for each shop.

Lesson Source Code

Lesson 16.zip

No labels