Class Diagram and Interface Specification

Comments on the UML

stockdata

It might be surprising that the classes `FailoverStockDB`, `YahooYQLStockDB`, `YahooCSVStockDB`, and `CachingStockDB` do not have any associations with each other. The reason is that, for benefits of testability, the `stockdata` classes are design with dependency injection. This allows each one to be tested individually without requiring any of the others.

When the classes are actually used, they are built into a pipeline, for example (Figure 1):

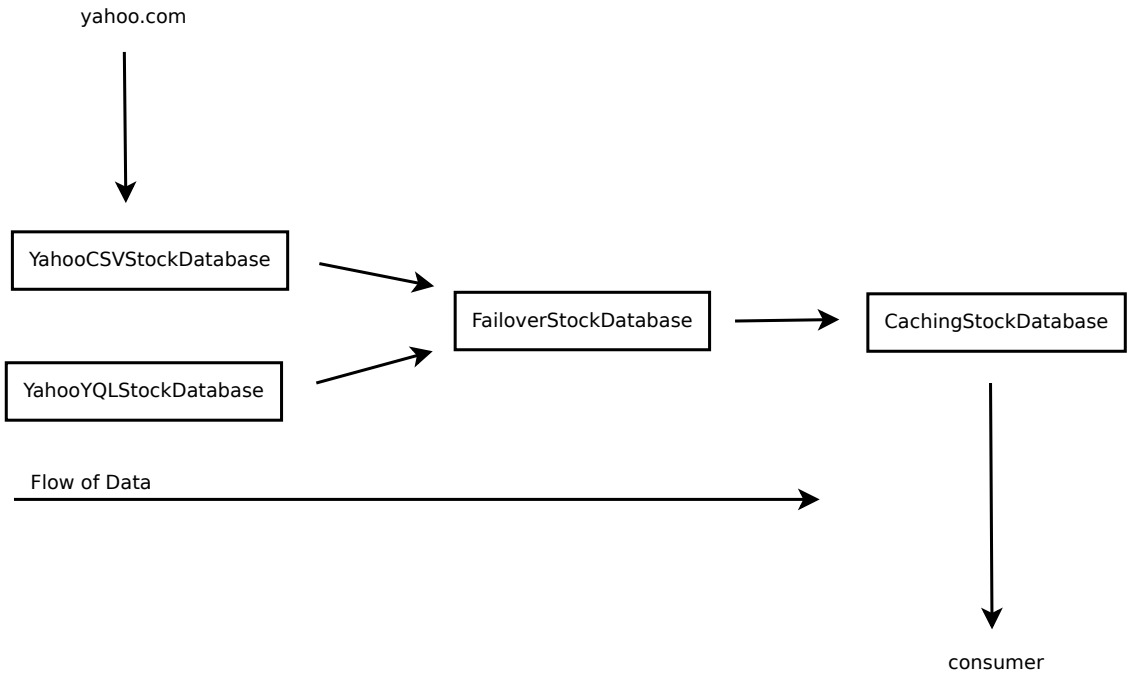


Figure 1: The pipeline architecture of the stock querying classes.

model.schema

These closely resemble the concepts in the Domain Model (See *Domain Model*). The biggest omission is that some of the Domain Concepts do not appear in actual code:

1. The "execution" of a trade was represented as a concept, but does not appear in the code as a class.
2. Likewise the "cancellation" of a trade.

3. A DividendEvent appeared as a concept but is merely procedural in the code.

Also not that some of the arrows in the UML diagram are encapsulated in association classes (See *Domain Model*).

texttrading

The texttrading code should also be viewed as a pipeline (Figure 2):

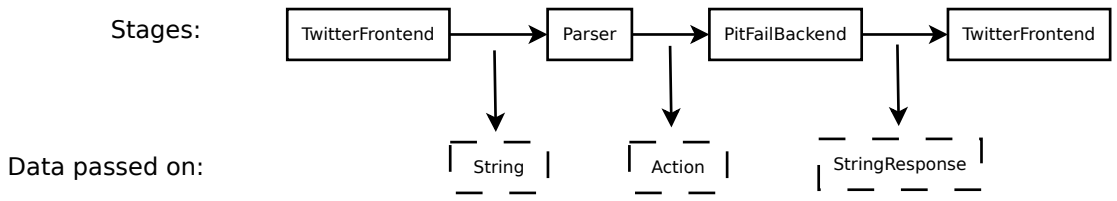


Figure 2: The pipeline architecture of texttrading.

website.control

This is the stateful part of the website: storing the current user, and the portfolio they are currently working with.

website.view

Not all the classes are shown here; however, all of them follow the same structure as the ones shown. A website view class has:

1. Some HTML in the form of Scala inline XML.
2. Some calls into the model to retrieve data.
3. Places in the HTML where data is inserted.
4. Some callbacks for user-input events (submitting a form).
5. Some calls into the model to perform the requested action.

The view is intended to be as thin a layer as possible on top of the real logic in the model.

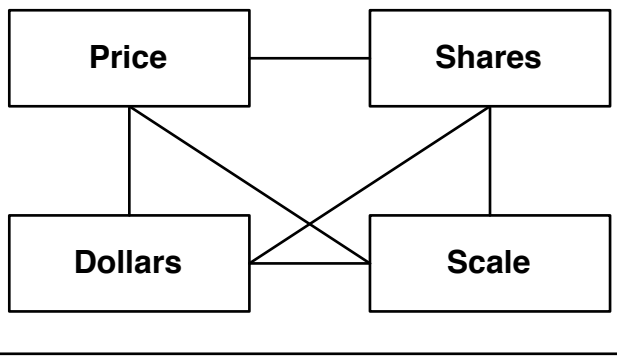
Android view

The Android is another View in the MVC architecture; the classes shown:

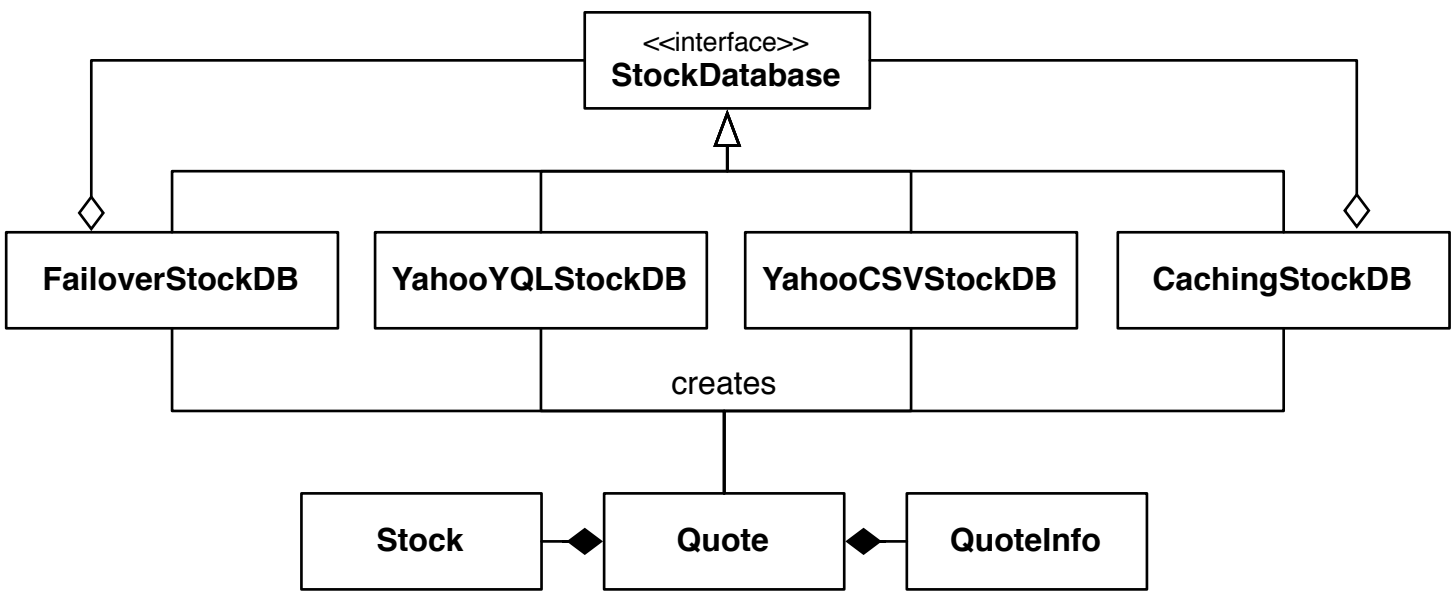
2. Make calls into the model (via servlets) to retrieve data.
1. Take user input (via the Android UI).
3. Make calls into the model to perform the user's actions.

The only real difference between the design of the android and website frontends is that the Android UI has no choice but to run on a separate machine, and so it must retrieve data and perform actions over HTTP.

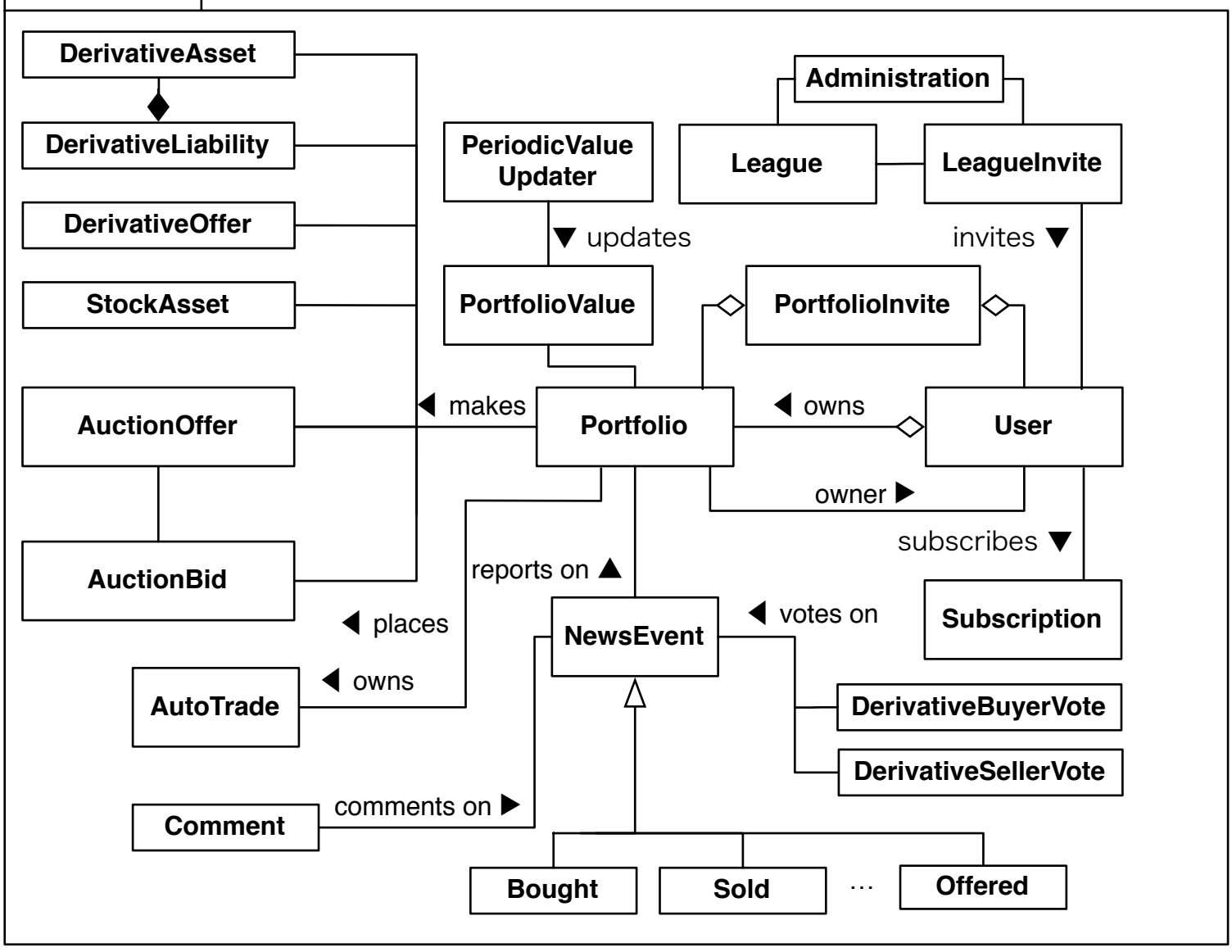
model



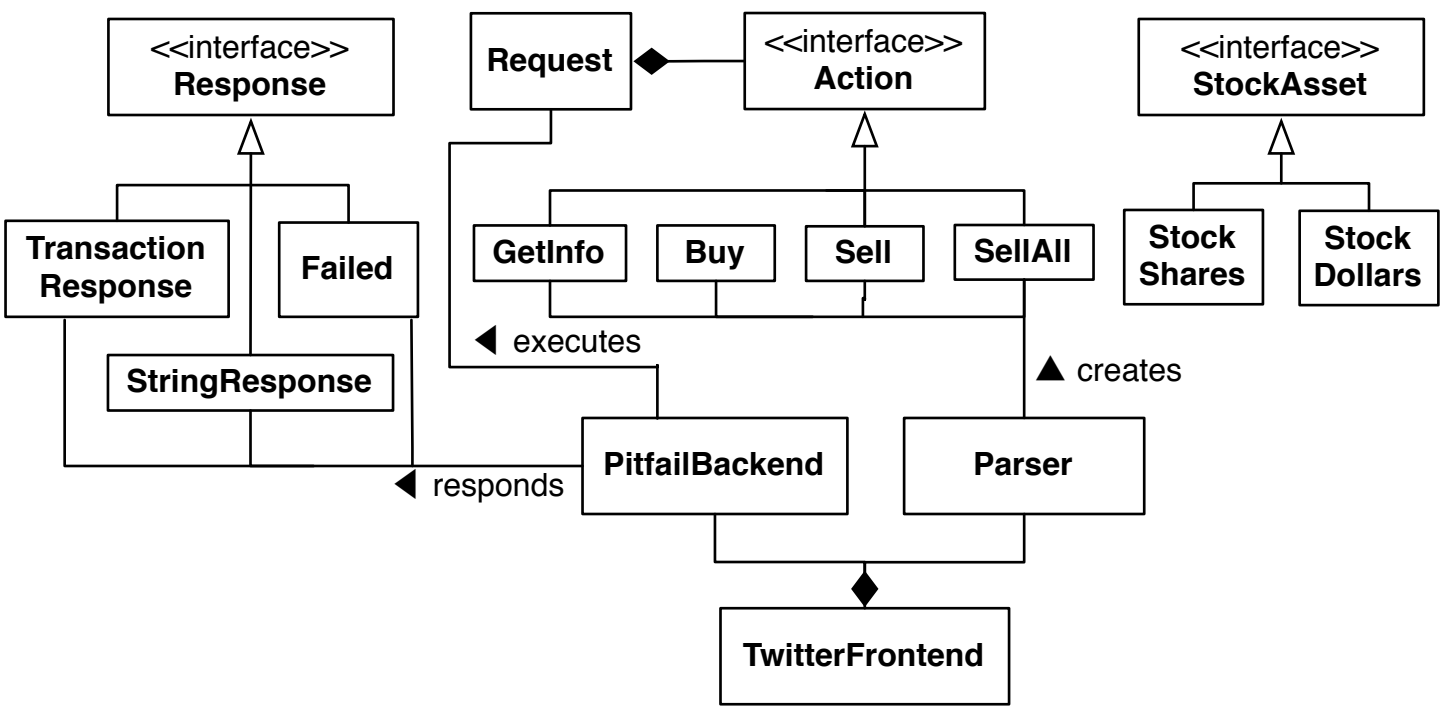
stockdata



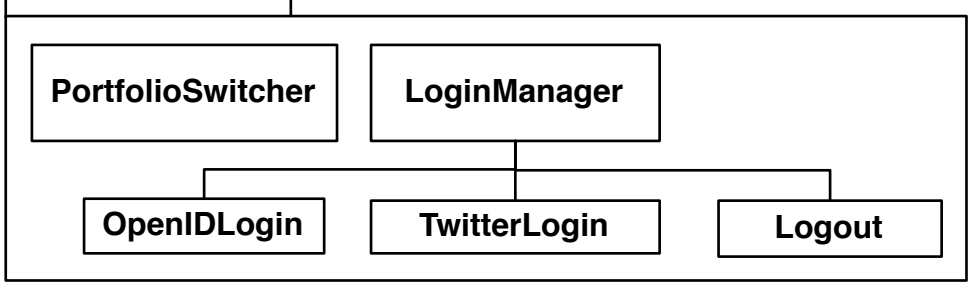
model.schema



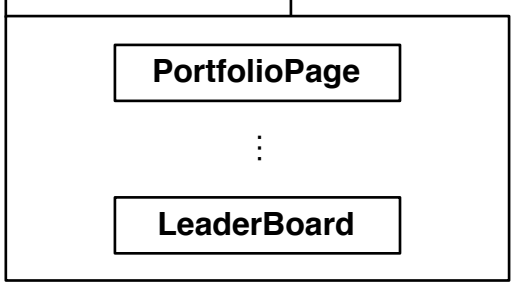
texttrading



website.control



website.view



android