90.308 Agile Software Development Best Practices with Java Project Doc

Document Last Updated - 15 February 2014

Collaborators:	Paul Wallace	Presentation (UI)
	Mike Grissom	Business Logic
	Stuart Connall	Persistence

GitHub Project: https://github.com/mikeyman77/StockTicker

Project: Stock Ticker

Summary

The Stock Ticker will enable users to request and track current stock prices.

User-Interface

The user interface will be implemented using Swing over JSP to simplify application deployment and configuration. It will provide functionality to register and login to the Stock Ticker client in order to view stock data the user is interested in. Upon registering/sign-in, the user will be presented with several options: 1) view and select stocks from a list of all known Nasdaq stock symbols to retrieve stock quotes, 2) type in and request a stock quote for a stock of choice, and 3) select a stock quote from a list of tracked stocks. The third option will not be available until the user designates at least one stock to be tracked. The initial stock(s) will display in a table-like list with the following information:

SYMBOL | TIME | PRICE | CHANGE | CHANGE% | LOW | HIGH | VOLUME

The user can select a stock to view more detail about that stock. The stock detail view will present the list data plus the following additional data:

PREVIOUS CLOSE
OPEN
BID
ASK
DAY'S RANGE
52-WEEK RANGE
AVERAGE VOLUME
MARKET CAP
P/E (ttm)
EPS (ttm)

The user can click the '+' or '-' button to track and untrack a stock. The '+' option will be disabled when the track is currently being tracked and the '-' will be disabled when a stock is not yet being tracked.

The user interface will obtain a reference to the `AuthorizationService` and

`StockTickerService` from the business logic component.

Additional Ideas

1) Implement a graph of some sort using jChart. At this point we are unsure of the complexity or scope of adding this feature.
2) Implement a stock marquee that scrolls up-to-date stock info (just the basics) for tracked stocks. This would update on a user specified interval. Default would be every 15 minutes. This would require an additional thread to implement. At this point we are unsure of the complexity or scope of adding this feature.
3) Implement a stock history feature that would track all stock requests (with different timestamps) for a user over time. This could be filtered by date in the event too much data is collected and stored in the database.
4) Implement an RSS feed to retrieve headline news either for specific stocks or top stocks
5) Implement a configuration option to enable user to customize the UI view, i.e., what they see on the screen, and possibly the actual data retrieved by the business logic. For example, the user could elect to hide the bid and ask fields from the view or possibly tell the business logic they are no longer interested in receiving this data. In this case, the view would automatically hide the field.
6) Implement an option to handle different currencies. This could be based on information, such as the country of origin that the user enters at registration time.

Business Logic

This component handles all of the user requests regarding sign-in and retrieval of stock related information, such as stock quotes, stock tracking data, etc. In addition, the business logic will handle all communication with the persistence layer to store and retrieve business data.

Stock information will be retrieved using Yahoo’s Finance API for CSV. Data will be retrieved upon request as a .csv file, parsed, and passed back to the User Interface for display.

The business logic component will obtain a reference to the `PersistenceService` interface from the persistence component. The interfaces are as follows:

`StockTickerService`

- `public StockQuote getStockQuote(Stock stock)`
- `public List<StockQuote> getStockQuotes(List<Stock> stock)`
- `public List<Stock> getTrackedStocks(User user)`
- `public boolean trackStock(User user, Stock stock, boolean tracked)`
- `public boolean isStockTracked(User user, Stock stock)`

`AuthorizationService`

- `public boolean logIn(User user)`
- `public boolean logOut(User user)`

- `public boolean register(User user)`
- `public boolean unRegister(User user)`
- `public boolean isLoggedIn(User user)`

Persistence

The persistence layer will be implemented using an embedded H2 database engine. H2 is a light-weight, 100% Java implementation, that is ideal for this application. The interface is as follows:

`PersistenceService`

- `public List<Stock> getTrackedStocks(User user)`
- `public boolean trackStock(User user, Stock stock, boolean tracked)`
- `public boolean isStockTracked(User user, Stock stock)`
- `public boolean userExists(User user)`
- `public boolean saveUser(User user)`
- `public User loadUser(User user)`
- `public boolean deleteUser(User user)`

Testing

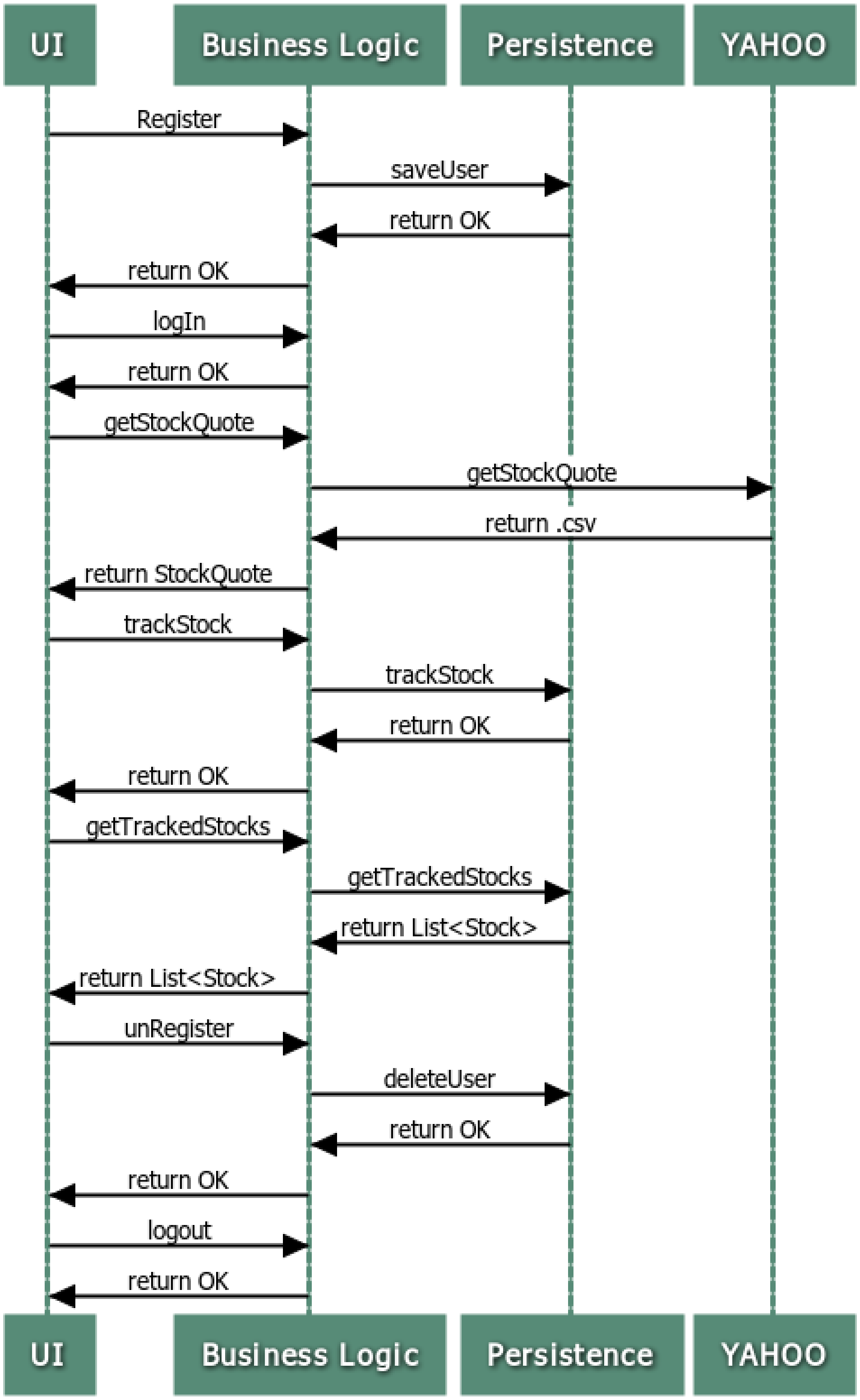
All classes and methods will have JUnit tests and Mockito will be used where mocking is needed. For example, the UI may want to mock the Business Logic interfaces during testing. Likewise, the Business Logic may want to mock its calls to the Persistence Layer. Another way is for each component owner to create stubs and make them available as a separate code branch.

All code after successful unit testing will be pushed to the `testing` branch. When code is deemed production ready, it will be promoted to `master`.

Logic Flow Between Components

The following diagram shows some of the interaction between components when a user performs an action in the user interface. To read this diagram, start on the top left and follow the arrows. Most, but not all calls from the User Interface are show here.

Stock Ticker Logic Flow



www.websequencediagrams.com