Abstract:

The goal of this project is to provide an elegant, simple interface that people can use to compare the performance of different financial instruments.

Background:

Most people who contract with financial managers do not have much understanding of how much benefit paid support provides them in terms of investment return.

Similarly, most people who manage their own portfolios do not know how well they are doing with respect to the rest of the financial market.

The key point is ease of use – the website needs to accessible to users who have little or no financial knowledge.

With the information provided through this project, users can test their long term investment strategies and in turn become better informed investors.

Technologies Used:

- ASP.NET – used to develop the frontend of the application.
- AJAX – used to enhance user experience and avoid full page refreshes.
- MSSQL – used to store the data used by the application.
- C# – used to create a Yahoo! Finance API.
- LINQ – used to add native data querying to the .NET platform.
- .NET 3.5 – Underlying framework used for both the frontend and the backend of the application.

I also set up and maintained the database and web server used by this project.

At times, the use of bleeding edge technologies such as LINQ or .NET 3.5 and the concomitant lack of documentation proved to be a challenge.

Features:

- Create and manage multiple portfolios.
- Graph the performance of your portfolio over varying time periods.
- Show the Industry and Sector Breakdown of each portfolio and show top performers in chosen Industry or Sector.

User Interface

The most important part of this project is a good user interface.

Investment Portfolio Performance Evaluation

Jay Patel
Faculty Advisor: James Gee

Portfolio Performance Evaluation

Abstract:

The goal of this project is to provide an elegant, simple interface that people can use to compare the performance of different financial instruments.

Background:

Most people who contract with financial managers do not have much understanding of how much benefit paid support provides them in terms of investment return.

Similarly, most people who manage their own portfolios do not know how well they are doing with respect to the rest of the financial market.

The key point is ease of use – the website needs to accessible to users who have little or no financial knowledge.

With the information provided through this project, users can test their long term investment strategies and in turn become better informed investors.

Technologies Used:

- ASP.NET – used to develop the frontend of the application.
- AJAX – used to enhance user experience and avoid full page refreshes.
- MSSQL – used to store the data used by the application.
- C# – used to create a Yahoo! Finance API.
- LINQ – used to add native data querying to the .NET platform.
- .NET 3.5 – Underlying framework used for both the frontend and the backend of the application.

I also set up and maintained the database and web server used by this project.

At times, the use of bleeding edge technologies such as LINQ or .NET 3.5 and the concomitant lack of documentation proved to be a challenge.

Features:

- Create and manage multiple portfolios.
- Graph the performance of your portfolio over varying time periods.
- Show the Industry and Sector Breakdown of each portfolio and show top performers in chosen Industry or Sector.

User Interface

The most important part of this project is a good user interface.

Database Schema:

- This project is very data intensive - for each of this website’s members, user information and portfolio contents must be stored on the database.
- Daily price information for each stock will also need to be stored, as well as specific categorical information.
- To retrieve data, I created a simple API to retrieve information from Yahoo! Finance.

Once data is retrieved it is stored in my own database for easier and more efficient access.

Future Improvements:

- Currently, stock information is retrieved using a “pull” architecture – information is updated only when it is requested.
  - Causes unnecessary slowdown for end user.
  - Information can quickly become stale.
- Ideally, the project should move to a “push” architecture – information is updated as soon as it becomes available.
  - Can isolate stock information retrieval logic and website logic.
- Website logic no longer has to take the integrity of stock information into consideration.
- Implement security measures to protect user information.
  - Secure session management/authentication.
  - Cryptographic storage in databases.