InvestiMapp

Software Architecture and Design Pattern Document

Table of Contents

[InvestiMapp Architectural Design](#_5gvaricbhlk0)

[UML package diagram](#_wueph8cu54o1)

[Architecture](#_apsl23j7skmn)

[InvestiMapp Design Patterns](#_idps748b895v)

[Creational: Object Pool](#_izkdu1bbtsqg)

[Why we chose it.](#_fr14ct4pk9vy)

[Structural: Proxy / Facade](#_yfye4rm4d6wn)

[Why we chose them.](#_yii01zhab6w)

[Behavioral: Observer](#_45dm23d1mmgr)

[Why we chose it.](#_wxckjgq5ejjz)

# 

# 

# 

# 

# 

# InvestiMapp Architectural Design

## UML package diagram

# 

## 

## 

## **Architecture**

Client-Server model with a DAO (Data Access Object)

We are using Client-server model, because InvestiMapp is a web server with browsers. In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server.

Why do we choose it:

Because InvestiMapp is a webapp that collects stock data and shows to users. It is using for collecting set of stand-alone servers which provide specific services such as printing, data management, etc. InvestiMapp is a kind of collecting Stock data for users and access client-servers to make use of them.

# InvestiMapp Design Patterns

## Creational: ***Object Pool***

Definition of object pool creational design pattern: "A client with access to a Object pool can avoid creating a new Objects by simply asking the pool for one that has already been instantiated instead.”

### Why we chose it.

Stocks all have the same characteristics and one user will never have access to trading stocks that another user can’t trade. Therefore if one user is looking to track a certain stock, there is a good chance that another user is also looking to track it as well. For this reason, object pool works great. We will never have more than one instance of a stock. We store the user’s details: stock purchase price and quantity, then user the stock ticker to find the rest of that stock’s details in the object pool.

## Structural: Proxy / Facade

Definition of proxy and facade structural design pattern: Proxy: “Add a wrapper and delegation to protect the real component from undue complexity.” and Facade: “Wrap a complicated subsystem with a simpler interface.”

### Why we chose them.

These two patterns go hand in hand when it comes to web applications. We used them both for security and ease of use. We have API endpoints setup which sets up the proxy pattern. Each one will take data or get data from the database and keep SQL injection attacks from happening. In the future we could also use services like [Auth0](https://github.com/auth0-samples/auth0-jquery-samples/tree/master/01-Login) for token checking which can be done at these endpoints as well. The facade pattern will allow us to change which Stock API we are using depending on availability. These APIs like to change their structure or authentication regularly. With the facade pattern we won’t have to redeploy our interface to make these changes.

## Behavioral: Observer

Definition of observer design pattern: “Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”

### Why we chose it.

This patterns makes sense because it echos the use of object pools from creational patterns. When one stock changes and three different users own that stock then three users need to be update on those changes. The way we setup our front end allowed us to separate the object pool and the user data so that you are an observer of the data and on the client side is where the calculation for equity and percent gain or loss are calculated.