Rationale_Design_Patterns_Used

The following include rationales for the design patterns used by Payday.

Server Side Model View Controller (MVC)

- A model view controller design pattern is a system that allows controller layers to serve and manipulate data from model layers, then serve the data and take inputs from view layers. Payday's server side MVC design includes models that contain data objects that are built from the database and controllers serving data to views that are Restful JSON endpoints. The rationale for using a model view controller design pattern within the server side logic for Payday is that it allows for modularization of code, it promotes better maintenance practices, minimizes spaghetti code and promotes DRY (Don't Repeat Yourself) principles.

Client Side Model View Controller (MVC)

- Payday's client side MVC design includes models that contain data objects that are built from GET, PUT, POST, and UPDATE HTTP requests from the server side logic that serves the Restful JSON endpoints as views. Controllers then serve data to views that are contained in the presentation client layer of the application. The rationale for using a model view controller design pattern within the client side logic for Payday is that it is a system that allows controller layers to serve and manipulate data from model layers, then serve the data and take inputs from view layers. It also allows for modularization of code, it promotes better maintenance practices, minimizes spaghetti code and promotes DRY (Don't Repeat Yourself) principles.

Singleton

- A singleton is a design pattern that allows a master object control logic that is required throughout the entire use of an application. The rationale for Payday to use a singleton design pattern is that it is the best implementation for application wide login sessions, interactions, and manipulation of data by a current logged in user object.