System Reference Guide

Tyler Smith

University of Phoenix

Software Architecture

CSS/422

Mohammad Kamali

July 3, 2017

When building a house, or cooking a meal, or developing a piece of software, a proper plan is always the key to success. There are many ways to build a house, just as there are many ways to solve the same problem through software, and the architecture of that software and the type of architecture it is based on determine how powerful that solution will be. There are many different architectures used in computing today, each with their own list of pros and cons. Choosing the right architecture and implementing it properly within your software will ultimately determine how successful your solution will be.

In this system reference document, we will be discussing four of these different architectural styles. Online banking systems use Client/Server architectures to prioritize security and accessibility, while Peer to Peer services like Torrents provide inexpensive and easily maintainable systems. Distributed computing programs like Microsoft’s Office 365 attempt to offer wide-scale accessibility and quick maintenance, for a price of course. Finally, Monolithic programs are easy to develop, easy to deploy, and are the most common type of desktop application.

The client/server architectural style is made up of separate client and server applications that are connected by a linking network. The simplest form of client/server system involves a single server application that can be accessed by multiple clients, also known as a Tier-2 architectural style. Online Banking applications use this architectural style because it prioritizes security and accessibility, by creating a secure network and only allowing secure access to that network.

Whenever you bank online, the client uses an application to access the bank’s database and perform actions on their accounts. The client application first verifies the users identity using an ID and password, and once that is authenticated, access to the server is given. Through the functions in the client app, the server performs functions and updates data that is held in the server’s database. Through this style of communication, security and information accuracy are maintained, but not without some drawbacks. Client/Server applications often have difficulty scaling, due to their reliance on one or a few servers. As more users access the server/servers, the more limited the server’s bandwidth and other resources become. This bottleneck is the most crippling part about the Client/Server model.

Peer to Peer (P2P) applications each device stores its own files, performs its own functions, and communicates with others in the network. In a client/server setting, the servers are relied on to perform critical functions like user authentication and payment processing. Peer to Peer has each computer in the network perform its own functions, while the network also allows access to shared files and directories from other computers with the same application. One of the most popular type of file-sharing formats today is Torrenting, which is widely used to spread pirated software and media, but can also be used to share files with users without having to maintain a server or being bottlenecked by a server’s bandwidth.

When downloading something using a Torrent client, the software first finds other users on the network that have the same file on their computers, also known as seeders. The software then facilitates a connection between those users and the client, and begins downloading the file piece by piece from the seeder’s computers. Once the file has been downloaded, the client is then added to a list of seeders and those files can be accessed from other users in the network to share. This is the power of the P2P architectural style, which can perform powerful functions by connecting computers and combining their strength, instead of relying on the strength of a server.

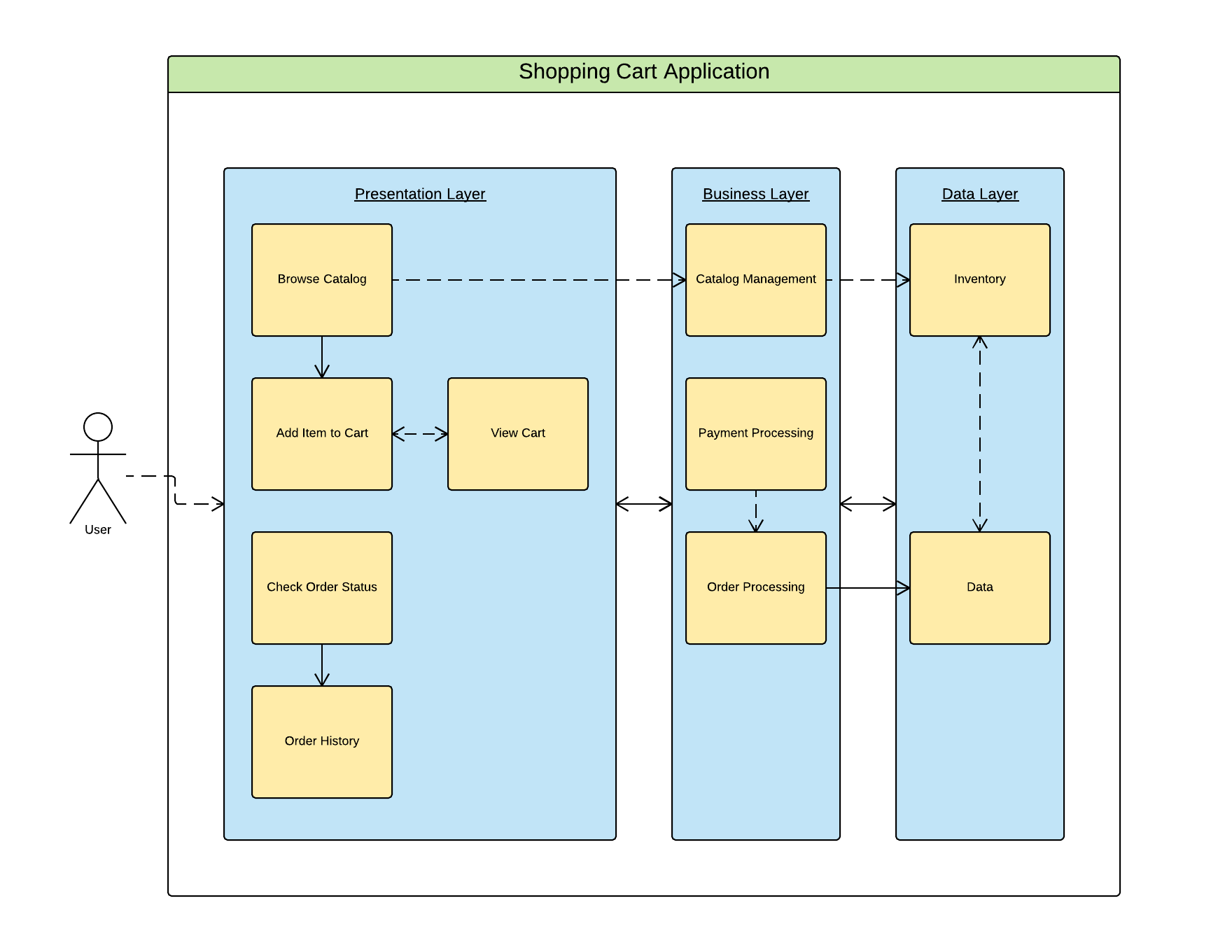
One of the fastest growing trends in the world of online computing is Distributed Computing, also known as Cloud Computing. Cloud computing allows users to access a server that performs most if not all of the functions of the system, and processes all of those functions away from the client’s computer. This means that almost any device with internet capability can have powerful functionality remotely, without needing a powerful computer to perform those tasks.

Microsoft’s Office 365 is a cloud-computing software system that allows user to connect to Microsoft’s servers, and use their entire suite of Office products online, without ever needing to download or update the software they use. User’s first need to connect to (and pay for!) this service on an internet browser, but once they’ve logged in, they can perform any function that the Office suite allows for. This power often comes at a price though, and cloud computing companies often charge high prices for the services they offer. Adobe’s entire creative suite is also available on the cloud now, but at a whopping $50 a month per user.

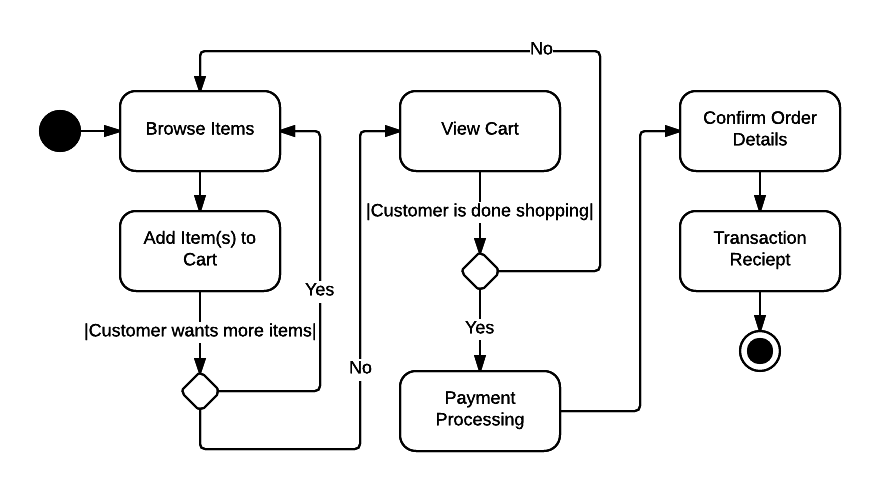
Monolithic software systems operate as their own systems that are isolated and do not need to communicate with other computers to perform functions. Many of the basic functions of computers are intentionally monolithic, because this style is the opposite of modular. Modularity is a programs ability to have parts of it used in other applications. Monolithic applications are non-modular because they often perform specific functions that have detailed and often intimidating code. The calculator function on your computer is monolithic, because it only receives information from the user, and performs functions solely based on the code it contains. Monolithic programs often have a ton of challenges involved with them though, such as a large learning curve for new developers on the project, a tendency to overload inter packages, causing slower loading, and poor scaling.

Remember that software architecture styles are the foundation of the system that you are developing. Without proper planning and understanding of each of the styles you can use, you could run into unexpected uses with scaling, or communication, or deployment, or any number of other factors. Using the resources available to you in this document and online, you and your colleagues should be able to properly plan and develop any piece of software.

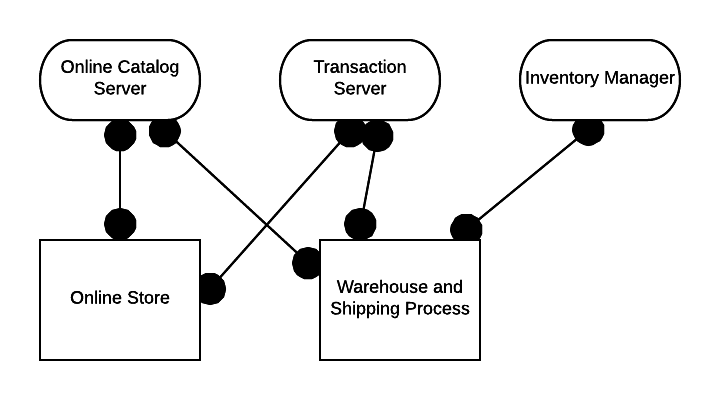
**Layered View:**



**Activity Diagram:**



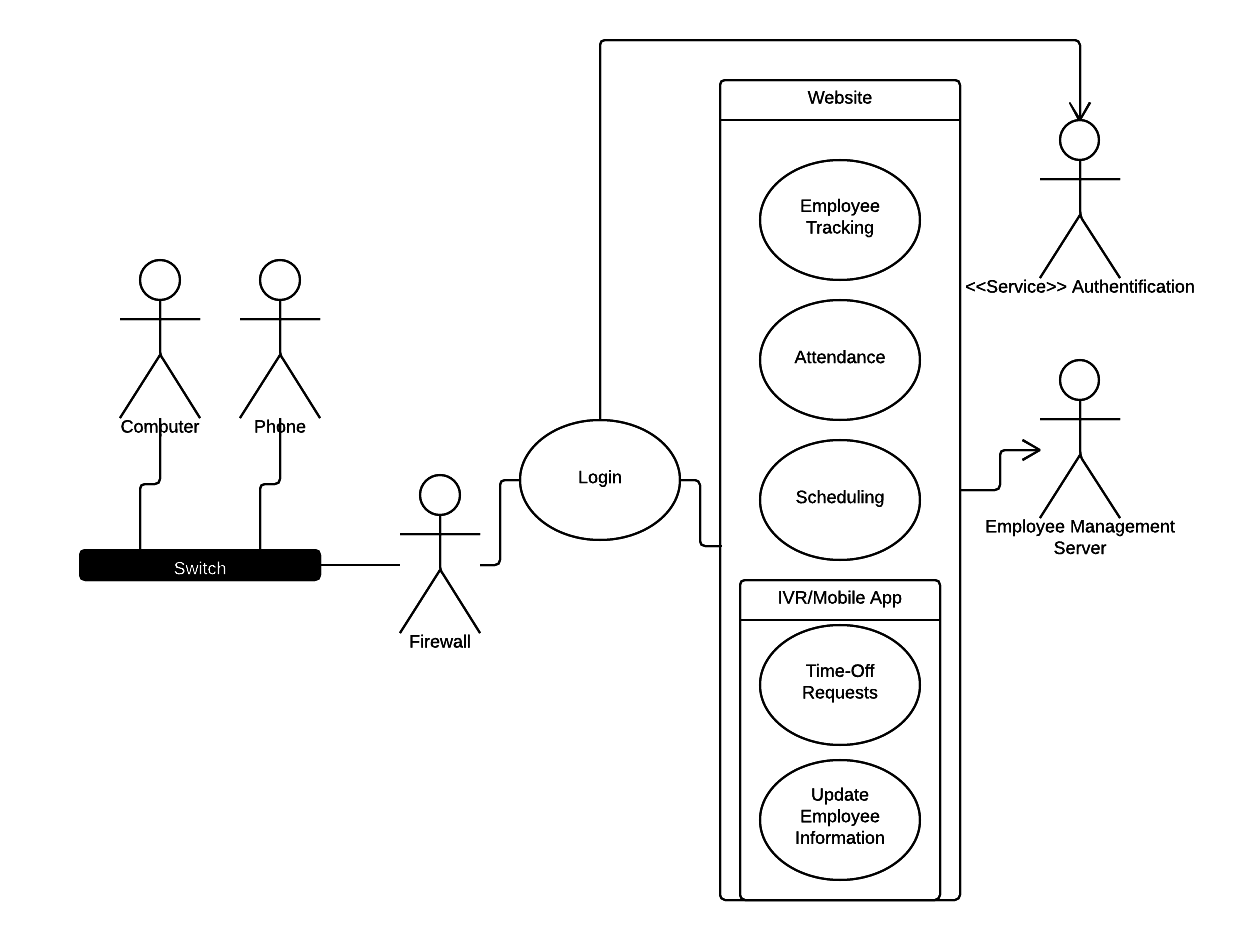
**Client/Server Diagram:**



**Quality Attributes Matrix:**

|  |  |  |
| --- | --- | --- |
| **Functional Requirement** | **Pattern Choice** | **Tradeoffs** |
| Accessibility | Service-Oriented | Sharing of a metadata can confuse the system |
| Auditability | Object-Oriented | These types of systems are difficult to secure |
| Dependability | Service-Oriented | Weakness due to storing all information and functions in the same system, which can corrupt data |
| Efficiency | Client-Server | Decreased dependability, due to a reliance on one or a few servers to complete tasks |
| Portability | Service-Oriented | Only works on systems that the cloud system has support for |
| Precision | Object-Oriented | An unclear plan during developmentcan cause issues with the end product when sharing meta data |
| Reliability | Service-Oriented | Reliant on availability of host servers, and how expensive hosting those servers is |
| Simplicity | Component-Based | Easy to make, but requires a component architecture, or run-time environment in order to work |

**Client-Server Implementation Diagram:**

****