**Week 5 Assignment: Software Architecture and Design**

**Final Project**

Shaun Hoadley

CST307: Software Architecture & Design

Professor Amr Elchouemi

December 20, 2021

**Software Architecture and Design Final Project**

**Introduction**

The purpose of this design document is to outline the software architecture and design for the new e-commerce site for the ABC retail stores owner by Enterprise X. The project team will consist of the software application architect, a product manager, a database administrator, and three software engineers. This new system involves two architectural patterns, as well as a number of design patterns. The architectural patterns being used on this project are the layered and MVC (Model-Veiw-Controller) patterns. The design patterns include the singleton, prototype, iterator, state, observer, transaction, session, and memento patterns. A use case diagram, class diagram, activity diagram, sequence diagram, and deployment diagram will represent the business and application views. This document also shows the object-oriented design for the system.

The aim of this project is to fulfill a request from the CEO to build an e-commerce site with the goal of recovering business from other e-commerce sites like Amazon, and eBay. Since the early 2000’s, when e-commerce sites began to emerge, allowing customers to purchase items from the comfort of their homes twenty-fours a day, seven days a week, ABC store sales and growth have significantly dropped. Giving ABC customers the ability to purchase ABC store items online with the option for home delivery or in-store pickup, the CEO envisions the growth rates ABC stores were experiencing in the 1980’s would be restored. Additionally, the e-commerce site will enable ABC stores to expand business, allowing third party vendors to sell similar items and collect a percentage of the profits.

**Team Roles**

\* Responsibilities are modified from various job postings on Indeed.com (n.d.)

**Application Architect –** The key responsibilities of the Application Architect include the following:

* They keep up with the ever-changing technical landscape.
* They work closely with product manager and stakeholders.
* They contribute to the system architecture, design patterns, non-functional requirements, and the tools/framework selection.
* They enable rapid product development and create innovative solutions for solving business challenges.
* They work closely with all teams, both on-site and off-site.
* They will develop and design a scalable enterprise e-commerce application.
* They serve as a mentor to the development teams.
* They develop and deploy microservices for use in a cloud environment.
* They communicate with the customers and the C-level executives.

**Product Manager -** The key responsibilities of the Product Manager include the following:

* They create the roadmap for the project addressing the project deadline, and provide accountability for unexpected delays with the development process.
* They review and support the architecture, design, and development to ensuring the business objectives and project goals are being met.
* They assist in the gathering and refinement of requirements.
* They represent the clients’ voice throughout the project.
* They identify possible future features and new opportunities.
* They ensure the product teams make and keep the business commitments.
* They communicate issues, dependencies, trade-offs, and status of the project with the stakeholders.
* They ensure the rapid and appropriate escalation of critical issues.
* They facilitate the SCRUM ceremonies.

**Database Administrator –** The key responsibilies of the Database Administrator include the following:

* They will design, implement, and administrate the databases, their access levels, and security.
* They perform the day-to-day database administration activities ensuring the integrity, security, performance, and availability of the production and development environments.
* They make any recommendations concerning database improvement and application design standards.
* They aid the other development team members with support of project design, implementation and deliverables.
* They create the scheduled processes used for automating routine functions.
* They build SQL Server Integration Services packages and SQL Server Reporting Services reports.
* They resolve database issues with support from the application architect and the software engineers.

**Software Engineers -** The key responsibilies of the three Software Engineers include the following:

* They develop and maintain the new e-commerce application.
* They perform unit testing throughout the entire SDLC.
* They communicate with the rest of the development team to ensure all requirements are met.
* They participate in daily SCRUM meetings and other SCRUM ceremonies.
* They work with the project manager and application architect on new feature planning.
* They develop a broad code base understanding.
* They maintain version control through the use of GitHub.

Throughout the lifecycle of the project, these roles will work together and support each other. The process requires close communication to refine the project requirements, architect the system for the e-commerce and inventory application, designing the system, developing and testing the software, and fulfilling the functional and non-functional requirements. They work collaboratively, involving stakeholders throughout the process, to employ agile methods, and utilize the scrum framework, to achieve the ultimate project goal of client satisfaction.

**Requirements**

**Goals**

There are several goals that the project is to be designed to reach. First, the development of an online presence for the ABC stores. Next, the ability to leverage the advantage of having physical stores as a place for pickups and returns in addition to the way other e-commerce sites operate. The third goal is to link the e-commerce and physical stores to key vendors to reduce inventory costs and pass on the savings to their customers. Next, linking the e-commerce site to a number of carriers as shipping choices. Another goal is to provide a portal for verified vendors of similar products to use as a marketplace for a percentage of the profits. Finally, to provide ABC customers an always-up availability, 24/7 access to ordering the products they want.

**Functional Requirements**

* ABC store items should be searchable for customers through an online e-commerce website
* ABC store items should be able to be purchased by customers from ABC stores online
* The choice of delivery or in-store pickup
* Inventory availability visible for all stores for specified items
* Shipping carrier choice selection at checkout for delivery orders
* Automated inventory notifications to vendors
* Marketplace for the sale of goods for verified vendors of similar goods

**Non-Functional Requirements**

* Culturally sensitive so user base can grow outside of the United States.
* 24/7 availability without interruptions
* New customer account creation in under a minute
* Customers should be able to find and purchase items they have searched for in under two minutes
* Customer data stored securely with critical data being encrypted
* The website should be accessible in all major browsers
* The website should be accessible and conform to ADA best practices

**Software Qualities**

**External Qualities**

***Availability***

* The e-commerce site should be available 24/7 without any interruptions

*Impact*

This is a project goal presented by the CEO. Downtime with the system directly affects the revenue generated by the online stores. Downtime also affects the user experience and might deter users from returning in the future. All scheduled maintenance should be done outside of peak hours.

***Usability***

* New customer account creation should take less than one minute
* Searched items should be found and purchased in under two minutes
* The website should be accessible and conform to ADA best practices

*Impact*

Users may or may not return to the site depending on usability. If the usability does not compare to other e-commerce competitors (eBay, Amazon, etc.), users may elect to use the competitor e-commerce sites instead.

***Portability***

* Culturally sensitive so user base can grow outside of the United States
* The website should be accessible in all major browsers

*Impact*

While ABC stores currently only have physical stores in the United States, having an online presence may allow growth to extend to international markets, thus it should be culturally sensitive. Having a bad experience with browser support may have an affect on the size of the user base.

**Internal Qualities**

***Maintainability***

* The e-commerce system should have low lines of code (LOC), high cohesion, and low coupling
* The system should be extensible, flexible, and modifiable
* The system should support perfective, preventative, adaptive, and corrective maintenance

*Impact*

The growth of the business is a major goal for the new e-commerce system. With growth, comes the need for scalability, which may be impeded by poor maintainability of the system.

***Interoperability***

* Employment of common standards in the system
* The use of interfaces in the system

*Impact*

The interoperability of a system increases from the implementation of common standards and interfaces. Parts of the system may interpret specifications differently if interoperability is not considered during implementation, making future integration of new systems difficult in the future. Integration tests may prove difficult. Integration issues may exist during a new release of the software. As new versions of different parts of the system are released, integrations may be difficult to maintain (Ingeo, 2018).

***Testability***

* Unit testing should be supported by the system
* The system tests should be self-documenting
* Tests should have high observability, high controllability, be highly isolatable, be highly automatable, and hve low complexity

*Impact*

Systems that are architected well are testable. Having the ability to do system tests throughout implementation allows developers to locate and correct errors in the code more rapidly, reducing development time and costs, and leading to more reliable code.

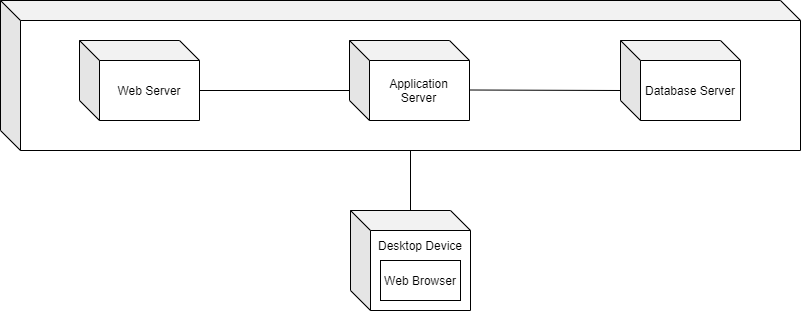
**Architecture Descriptions**

***Stakeholders***

* System Administrators, System Managers, and Operators
* Business Managers and Users
* Database Designers and Administrators
* System & Software Engineers

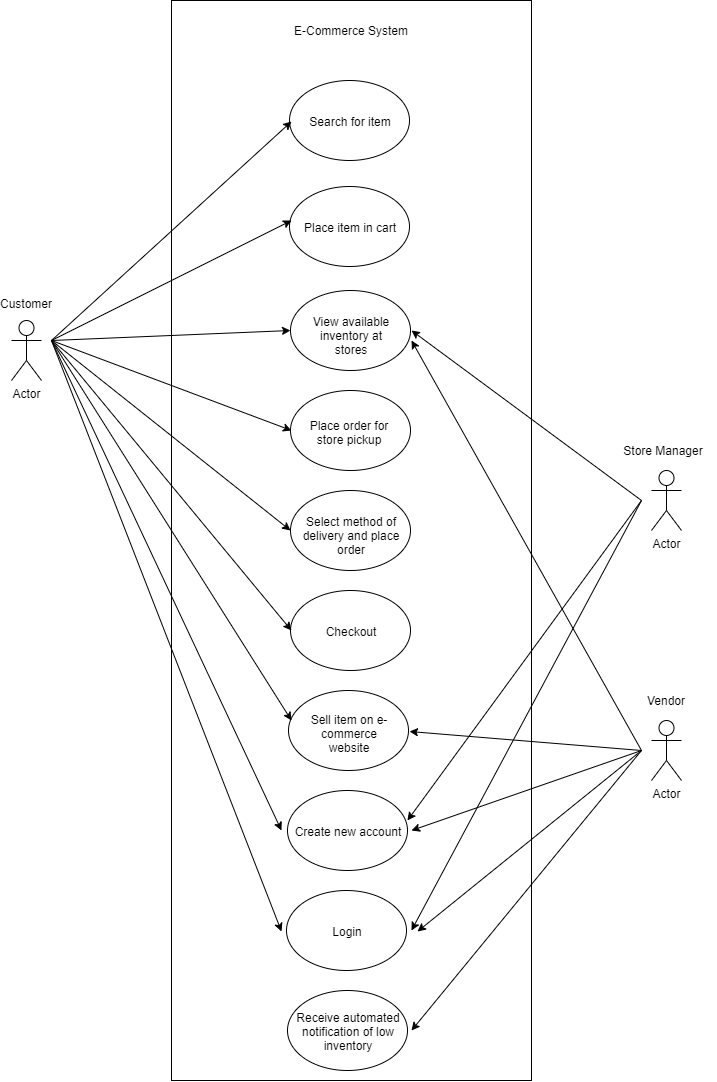
***UML Business Model Views***

**Deployment Diagram**



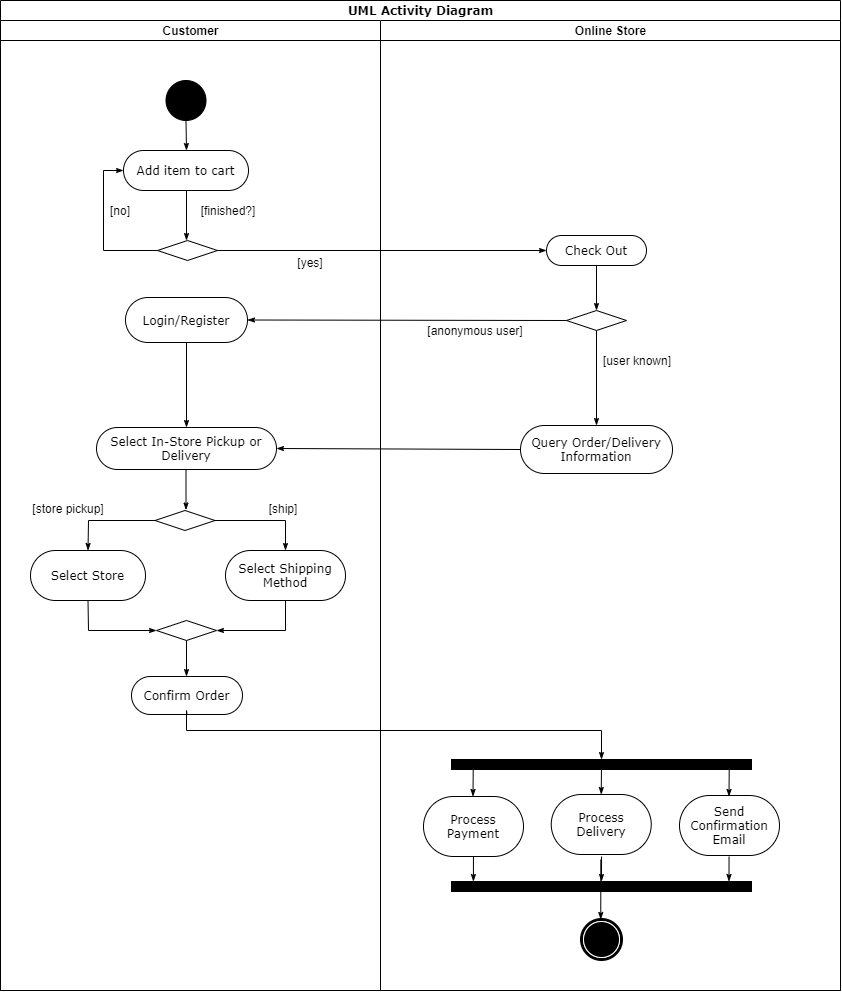
**Figure 1: Deployment Diagram for ABC stores**

**Use Case Diagram**

****

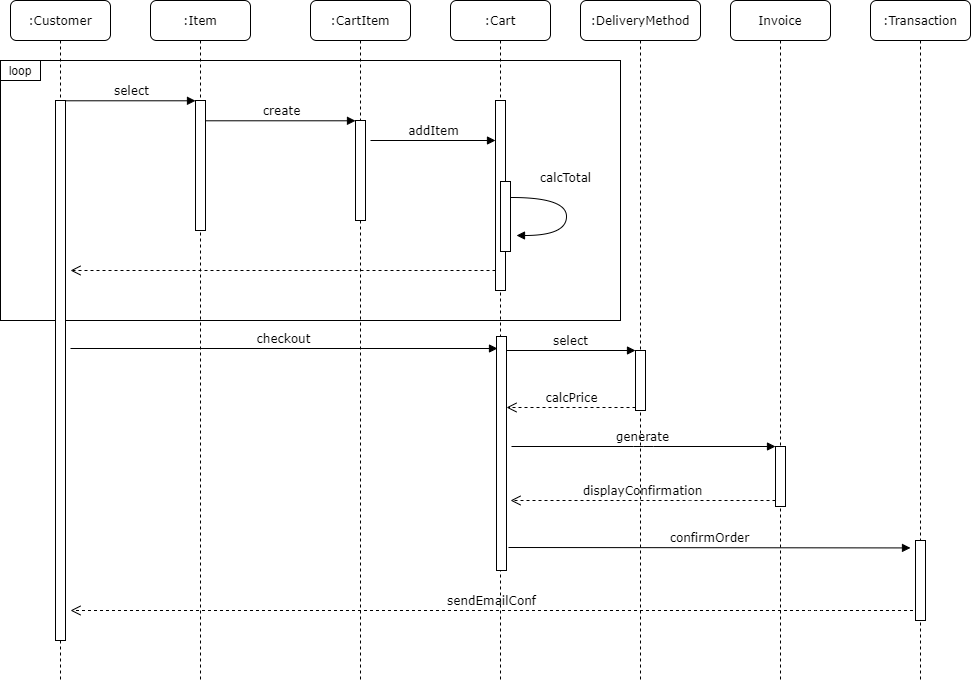
**Figure 2: Use Case Diagram for ABC stores**

**Activity Diagram**

****

**Figure 3: Activity Diagram for ABC stores**

**Sequence Diagram**

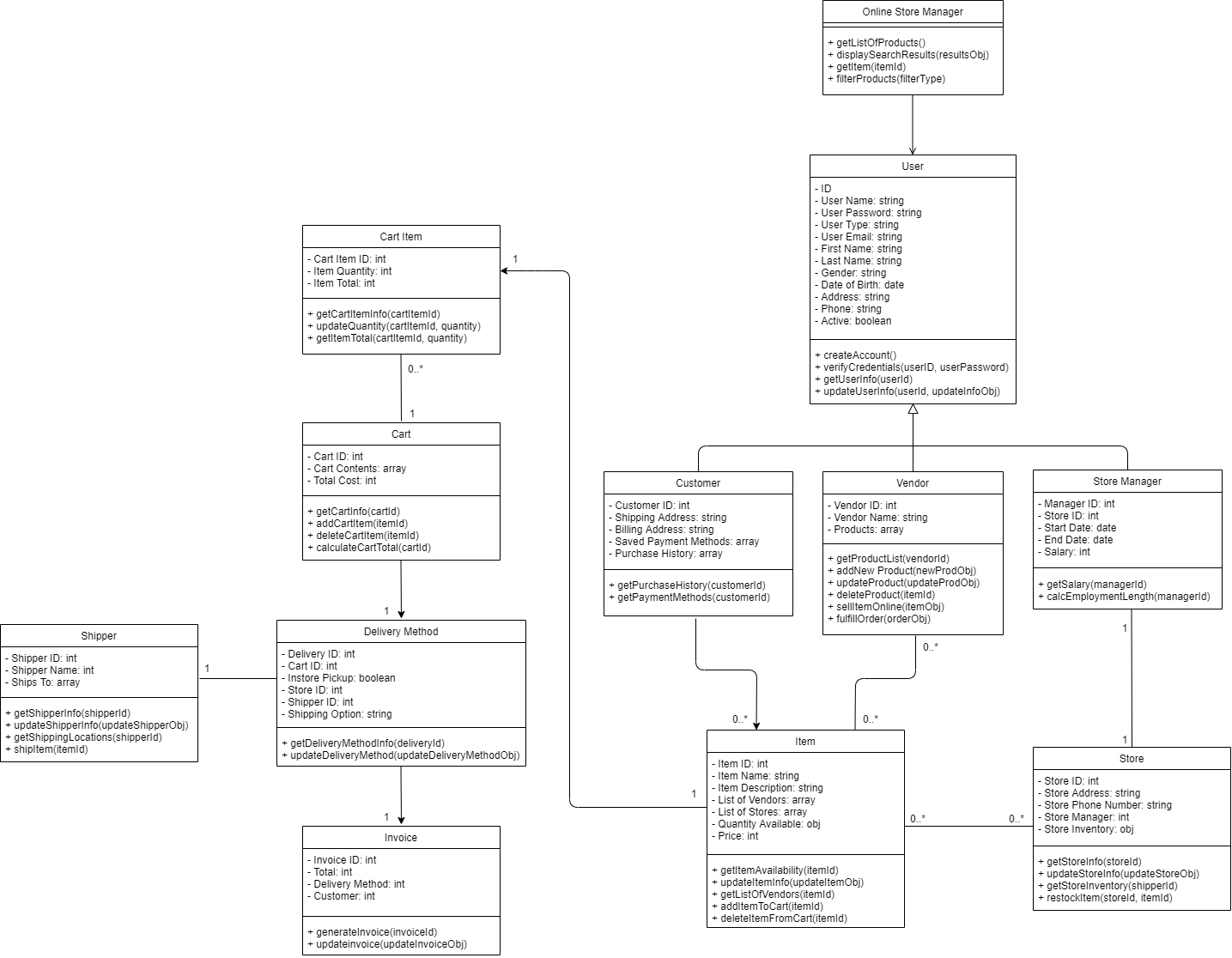
****

**Figure 4: Sequence Diagram for ABC stores**

**Class Diagram**

The class diagram is located in the next section, Object-Oriented Design.

**Object-Oriented Design**

****

**Figure 5: Class Diagram for ABC stores**

**Use of Design Best Practices**

***Encapsulation***

Separating the features and methods in a class that are likely to change from those that will not is encapsulation (Dooley, 2017). By using this design principle one can prevent the stable parts of a design from being affected by changes. Figure 5 above shows that the attributes related to instances of a User will not be changed. For example, there will always be an ID, user name, password, type, email, first name, last name, etc. for users of the ABC Online Store. Over time, however, the ABC Stores might add new types of users encompassing new attributes; To clarify, the class User has the sub-classes of Customer, Vendor, and Store Manager, each having unique attributes. ABC Stores could eventually add, or extend, new subclasses from User, like Administrator, Regional Manager, etc. Because the subclasses may change, they are separated from the User class, which will not change over time and this is an example of how encapsulation works.

***Open-Closed Principle (OCP)***

Figure 5 also shows the open-closed principle (OCP). The open-closed principle is closely related to encapsulation. In *Extending the “Open-Closed Principle” to Automated Algorithm Configuration (2019)*, it states that in the open-closed principle, “frameworks should be ‘open’ in the sense of being extensible via the addition of new components, but ‘closed’ in that they do not require framework modification to achieve this.” What this means is that attributes and methods that are not going to vary, should be abstracted into a super-class. Thus, when new features are requested, they can be added to the code, instead of requiring the modification of existing code. Again with figure 5, the commonalities that the Customer, Vendor, and Store Manager classes share are abstracted into the User class is an example of OCP. The User class is open to extension but closed to modification. When new subclasses of User are added (Administrator, Regional Manager, etc.), this can be achieved rather easily without modifying the existing code.

***Don’t Repeat Yourself (DRY)***

Don’t Repeat Yourself (DRY) attempts to avoid the unnecessary duplication of code. Whenever the same behavior can be observed in more than one place throughout a system, that behavior should be abstracted into a separate class (Dooley, 2017). In doing so, that behavior can be inherited and reused by instances of that class. In Figure 5, the User class is an abstraction of the Customer, Vendor, and Store Manager classes. Because the Customer, Vendor, and Store Manager classes share similar behaviors (i.e. createAccount() and verifyCredentials()), these behaviors have been abstracted out into a separate class called User. This prevents repeating the createAccount() and verifyCredentials() methods in all the of the subclasses.

***Single-Responsibility Principle (SRP)***

The single responsibility principle (SRP) is also being utilized in the design of the e-commerce site. Dooley (2017) states, “A cohesive class does one thing well, and doesn’t try to do anything else”. What is meant is that classes should have a single functionality that it executes well, and each class should only have one reason to change. A class that follows the SRP are thus marked by high levels of cohesion. In *Refactoring - Improving Coupling and Cohesion of Existing Code* (2004), cohesion is “the degree to which elements of a class belong together” (Du Bois, et al., 2004). For example, cohesion and the SRP is seen in the CartItem class in Figure 5. This class has a single responsibility, tracking a particular type of item in a customer’s cart and the item’s desired quantity. This class lets users update the quantity of individual items in the cart before checking out. The CartItem class does not need all of the data of a particular item (such as item name, description, vendors, etc.), that information is abstracted within the Item class. The CartItem class encompasses a single responsibility, and everything within the class belongs together demonstrating it is cohesive.

***Liskov Substitution Principle (LSP)***

The Liskov substitution principle (LSP) means, “When you derive one object from another, the base-level semantics should not change. If you find yourself writing branching logic so that your function does one thing if provided with a base class, but something else for a derived class, you have violated this principle.” (Spencer and Richards, 2015). Breaking this down, inheritance in object-oriented design should be clear and purposeful. Never should there be an instance of a class that does not use an behavior or attribute of the superclass from which it inherits. Suppose that in Figure 5, the billing address, store ID, and vendor name were listed as attributes under the User class instead of in the Customer, Store Manager, and Vendor classes respectively. This would mean that the billing address would be inherited but unused when instantiating a Store Manager or Vendor object. Similarly, the vendor name attribute in User would be inherited but unused when instantiating a Customer or Store Manager object. Again, the store ID would be inherited but unused when instantiating a Vendor or Customer object. This would be a violation of LSP; Therfore, vendor name is placed in the Vendor class because it is unique to Vendor, billing address is placed in the Customer class because it is unique to Customer, and store ID is placed in the Store Manager class because it is unique to Store Manager; Thus, making all of the attributes inherited from the User class applicable to the subclasses.

***Principle of Least Knowledge (PLK)***

Objects should be loosely coupled, this is the basis of the principle of least knowledge (PLK), also known as the Law of Demeter. *An Empirical Study on the Developers’ Perception of Software Coupling* (2013) describes coupling as, “the measure of the strength of association established by a connection from one module to another”. Meaning that modules, or classes, should function independently of each other. The design for the ABC site holds this true, as there are very few instances where a class possesses a method that relies on information from other classes (the addCartItem(itemId) method in the Cart class, which takes the argument itemId). PLK can be enforced by implementing the single-responsibility principle and making classes more independent from one another.

***Dependency Inversion Principle (DIP)***

The foundation for the dependency inversion principle (DIP) is, “dependency modules are inverted for the purpose of rendering high-level modules independent of the low-level module implementation details” (Haoyu & Haili, 2012). In a top-down design approach, typicaly, high-level modules are dependant on the implementation of code at lower levels. The inverse of that concept is DIP. Dooley (2017) explains further, “The modules that contain the high-level business rules should take precedence over, and be independent of, the modules that contain the implementation details.” In Figure 5, the User class exhibits DIP, because it still functions without the lower level classes like Vendor, Customer, and Store Manager. This means, if the Vendor, Customer, and Store Manager classes did not exist, a generic User could still be instantiated using the attributes and behaviors within the User class.

**Architecture Design Process**

***Attribute-Driven Design (ADD)***

The attribute-driven design method was selected for the design because the success of the store is dependant upon customer and client satisfaction. Frequently, functional requirements are prioritized more heavily in software development projects. It is vital to point out, however, non-functional requirements, such as quality attributes, are every bit as important to consider. Quality attributes can directly and indirectly affect a users experience. In *A Prescriptive Approach to Quality-Focused System Architecture* (2017) it states, “A system is considered successful if it meets stakeholder needs.” Since the success of the ABC e-commerce site is heavily dependent upon the satisfaction of its, the ADD method seemed to be the most practical.

According to Gielen (n.d.), ADD is a decomposition process focusing on quality attributes to design the software architecture. ADD, in comparison to the other architectural design processes, Zimeo, et al, (2013) states, “The ADD method, to design complex software systems, suggests to identify the architectural drivers from the software requirements with the aim of selecting the proper architectural patterns to adopt as a reference guide for the whole design phase. This allows software architects to reuse the consolidated experience as an existing knowledge base to reduce the design effort”. Because it is a step-by-step, iterative process that allows trade-offs between attributes to be revealed early in the development process, ADD is beneficial to users of the ABC e-commerce site, as it is likely that they will see increased usability, performance, security, and availability.

Diagram

Description automatically generated

**Figure 6: ADD process (Ingeno, 2018)**

***Architecture Patterns***

The architecture patterns selected for implementation for the ABC e-commerce site are the layered pattern and the model-view-controller (MVC) patterns. This decision was made after researching for effective architecture patterns for e-commerce systems. That research resulted in the following:

“A widespread architectural pattern that satisfies self-service interaction model is Model View Controller. This pattern, with its main variants (Model View Presenter and Presentation Abstraction Controller are the most known), suggests the organization of the presentation layer and its decoupling from the model of enterprise systems. However, additional patterns are needed to help the design of the other layers proposed by multi-layer and multi-tier decompositions. The layered organization allows for a clear separation of presentation, application logic and data management that multi-tiered architectures exploit in order to partition the different functional components onto dedicated resources, so ensuring important nonfunctional attributes, such as security, scalability, efficiency and data persistence” (Zimeo, et al., 2013).

Showing that MVC is commonly used in the e-commerce space because of it’s support of self-service interaction models (such as the UI of the ABC e-commerce site). However, the research also showed that the use of the MVC pattern alone is insufficiant, because it only addresses the UI. Therefore, a layered architecture pattern can be applied to compliment the MVC model selected for user interactions and address the full needs of the system. Together these patterns will support scalability, efficiency, security, and data persistence.

***Layered Architecture***

*Rationale*

One of the architecture patterns selected for the e-commerce site is a layered architecture. Mark Richards (2015) stated, “The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern”. Layered architecture describes software that is organized into horizontal layers stacked upon one another, in which information can pass through. The three most common layers are presentation layer (user interface), business layer (the business logic for the application), and data layer (handling interactions with data sources and operations within tables). This pattern was selected because layered architectures are often implemented when splitting a system into smaller components or layers that can be spread across servers is beneficial, and when increased reliability is a priority (Packt Editorial Staff, 2018). In our case, reliability is a vital quality attribute, making a layered architecture is the most desirable.

***Model-View-Controller (MVC)***

*Rationale*

The Model-view-controller (MVC) was also selected as an architectural pattern because it is a pattern that is common in World Wide Web applications. The MVC pattern divides an application into three parts that attempts to present information to users differently than it is represented internally. This allows a better user interface (UI) and user experience (UX): “MVC mostly relates to the UI / interaction layer of an application” (GeeksforGeeks, n.d.). The first part of MVC is the Model, which holds the application data, without the logic of how the data gets presented. The second part of MVC is View, which presents the data to the user. Lastly, the Controller is the part of MVC that serves as an intermediary between the Model and View components. The Controller listens for event triggers from the View to call the correct associated methods on the Model. Events can be as simple as a mouse click from the user. This pattern organizes the presentation layer and makes for an engaging UI and is thus beneficial in e-commerce applications.

**Design Patterns**

The ABC e-commerce site leverages eight design patterns: the prototype, singleton, observer, iterator, session, state, transaction, and momento design patterns.

***Prototype***

The application can “gather that information by instantiating the objects at predefined intervals and keep them in a cache, when an object is requested, it is retrieved from cache and cloned. When the legacy database is updated, discard the content of the cache and re-load with new object” (Kulkarni, et al., 2012) when users ask for more specific information about an item. This pattern is based on the ability to copy an object, rather than creating a new one from scratch, saving time and leading to better user experiences.

***Singleton***

Database connections are an example of a singleton in the e-commerce system. A singleton pattern is used when a class will only be instantiated once and only once. It is a simple pattern allowing a class to be defined. According to GeeksforGeeks (n.d.), “A singleton class shouldn’t have multiple instances in any case and at any cost. Singleton classes are used for logging, driver objects, caching and thread pool, database connections.”

***Observer***

In the ABC e-commerce system, the observer design pattern can be used to notify users as their items move through the different stages of the shipping process (Kulkarni, et al., 2012). The pattern may also be used for notifying users of item availability. The observer pattern defines a dependency between objects so that the dependents are notified when an object changes state (GeeksforGeeks, n.d.).

**References**

Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., & De Lucia, A. (2013). *An empirical study on the developers’ perception of software coupling*. 5th International Conference on Software Engineering (ICSE), Software Engineering (ICSE), 2013 35th International Conference On, 692–701. https://doi-org.proxy-library.ashford.edu/10.1109/ICSE.2013.6606615

Dooley, J. F. (2017). *Software development, design, and coding: With patterns, debugging, unit testing, and refactoring* (2nd ed.). Retrieved from https://www.vitalsource.com/

Du Bois, B., Demeyer, S., & Verelst, J. (2004). *Refactoring - improving coupling and cohesion of existing code*. 11th Working Conference on Reverse Engineering, Reverse Engineering, 2004. Proceedings. 11th Working Conference on, Reverse Engineering, 144–151. https://doi- org.proxy-library.ashford.edu/10.1109/WCRE.2004.33

GeeksforGeeks. (n.d.). *Design Patterns*. Retrieved from https://www.geeksforgeeks.org/mvc-design-pattern/

Gielen, F. (n.d.). *What is the ADD Process?* Coursera. Retrieved from https://www.coursera.org/lecture/iot-software-architecture/what-is-the-add-process-kkO5T

Haoyu, W., & Haili, Z. (2012). *Basic Design Principles in Software Engineering*. 2012 Fourth International Conference on Computational & Information Sciences, 1251. Retrieved from http://search.ebscohost.com.proxylibrary.ashford.edu/login.aspx?direct=true&db=edb&AN=86 532662&site=eds-live&scope=site

Indeed.com. (n.d.) Find Jobs. Retrieved December 18, 2021 from indeed.com.

Ingeno, J. (2018). *Software architect’s handbook: Become a successful software architect by implementing effective architecture concepts*. Retrieved from https://www.vitalsource.com/

Kulkarni, P. S., Patil, S. P., Rane, P. B., & Meshram, B. B. (2012). Use of Design Patterns in E- commerce Application: Survey. *International Journal of Advanced Research in Computer Science, 3(1), 447*. Retrieved from http://search.ebscohost.com.proxy-library.ashford.edu/login.aspx?direct=true&db=edb&AN=91876394&site=eds-live&scope=site

Neill, C. J., Sangwan, R. S., & Kilicay-Ergin, N. H. (2017). A Prescriptive Approach to Quality- Focused System Architecture. *IEEE Systems Journal, Systems Journal, IEEE, 11(4), 1994–2005*. <https://doi-org.proxy-library.ashford.edu/10.1109/JSYST.2015.2423259>

Packt Editorial Staff. (2018). *What is a multi-layered software architecture?* Retrieved from https://hub.packtpub.com/what-is-multi-layered-software-architecture/

Richards, M. (2015). *Software Architecture Patterns*. O'Reilly Media, Inc. Retrieved from https://www.oreilly.com/library/view/software-architecture- patterns/9781491971437/ch01.html

Spencer, L. D., and Richards, S. H. (2015). *Reliable JavaScript: How to Code Safely in the World's Most Dangerous Language*, John Wiley & Sons, Incorporated. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ashford- ebooks/detail.action?docID=4040657.

Swan, J., Adriænsen, S., Barwell, A. D., Hammond, K., & White, D. R. (2019). Extending the “Open-Closed Principle” to Automated Algorithm Configuration. *Evolutionary Computation, 27(1), 173–193*. https://doi-org.proxy-library.ashford.edu/10.1162/evco\_a\_00245

Zhang, A. (2018). *How to write a good software design doc*. Retrieved from https://medium.freecodecamp.org/how-to-write-a-good-software-design-document-66fcf019569c

Zimeo, E., Oliva, G., Baldi, F., & Caracciolo, A. (2013). Designing a Scalable Social E- Commerce Application. *Scalable Computing: Practice & Experience, 14(2), 131*. Retrieved from http://search.ebscohost.com.proxy- library.ashford.edu/login.aspx?direct=true&db=edb&AN=93425282&site=eds-live&scope=site