Caringal Kllen Dalemar M.

3F4

When developing our e-commerce it's important to add the functional design to ensure maintenance, scalability and a clear understanding of the systems. We include concept such, as Separation of Concerns, Single Responsibility, Modularity and avoiding redundancy (DRY) which help us maintain a structured and effective codebase.

Separation of Concerns involves dividing the application into sections where each part's responsible, for a specific task or functionality within the system structure. Like in the login process is handled separately from the registration flow and product display functions to make development and debugging easier without causing any problems, between the other. The Single Responsibility Principle takes this a step further by emphasizing that each file or module should focus only on one function. Like in User_info.js handles user data management this we're the structure of the database model is in. while sma_controller.js is use in managing product and session related tasks. This division makes it easier to update or repair a section of the application without affecting the other code.

DRY principle (Don't Repeat Yourself) it avoid writing the code to prevent duplication of code snippets. Of rewriting code, for tasks like database connections or session handling in sections of your applications codebase it's better for the file be recyclable (just like the database.js) and use it throughout the application. This practice helps minimize mistakes maintain a codebase and smooth maintenance efforts by saving time.

When building the e-commerce architecture is organized using the MVC (Model–View–Controller). In this setup the Controller (like sma_controller.js file) manages requests. Retrieves information, from the database (Model) then forwards it to the View for display, on the user interface screen. This system is straightforward to grasp and effectively handles user engagement within the application.

References:

E Store Free Website Template | Free CSS Templates | Free CSS. (n.d.). Retrieved October 29, 2024, from https://www.free-css.com/free-css-templates/page260/e-store?fbclid=IwY2xjawGN06RleHRuA2FlbQIxMAABHSNIAppketTSLAPXhnXiXoMnmtdqM-UTiqCAo6OUnD3jLE3xGZOT2mwS_A_aem_-1T7thf036wffdGWbeH5iw

Tolentino, Kirk Adison M.

3-F4

In creating our e-commerce, we made sure to follow some essential software design principles to keep everything organized and easy to maintain. With features like product browsing, order management, and payment processing. We use SOLID principles and the DRY (Don't Repeat Yourself) approach to guide our development.

One of the key principles we used is the Single Responsibility Principle (SRP). This means that every part of the system does only one job. For instance, our inventory system focuses solely on tracking stock levels, while the payment module handles just the transactions. This makes it easier to update or fix one part without worrying about breaking something else. It also helps us keep the system modular, meaning we can scale or improve individual components as needed. We also made sure to apply the DRY principle to avoid writing the same code multiple times. Instead of repeating code for things like product filtering or discount calculations, we created shared functions that we can use throughout the platform. This saves us time, reduces bugs, and makes our code easier to maintain.

To make sure everything stays well-organized, we followed the Model-View-Controller (MVC) pattern. The Model takes care of data (like product and customer information), the View handles what the user sees (like product listings or the shopping cart), and the Controller processes actions (like adding items to the cart). This clear separation of roles makes development smoother and allows us to work on different parts of the system without stepping on each other's toes.

Reference

T Wulfert, R Woroch, G Strobel, S Seufert, F Möller - Electronic markets, 2022 - Springer

https://link.springer.com/article/10.1007/s12525-022-00558-8

Aheizer B. Rodriguez

BSIT 3-F4

Several software design principles and patterns are essential to creating an organized and effective application when developing an e-commerce platform with essential features like user accounts, product management, shopping carts, order management, checkout procedures, and inventory management. Particularly pertinent are important design tenets like Dependency Inversion Principle (DIP), Open-Closed Principle (OCP), and Single Responsibility Principle (SRP). SRP, for example, makes it easier to maintain and grow the code by guaranteeing that every class in the system has a specific purpose. By separating various activities into distinct, manageable classes, this technique is particularly helpful for handling tasks like user authentication and product catalog management. In this situation, OCP is advantageous because it makes it possible to add new features without changing the existing code, increasing the system's adaptability. DIP encourages loosely linked components, which are crucial for e-commerce systems that depend on third-party APIs, inventory management, and payment gateway integration.

Several software design patterns are helpful when these ideas are applied. A popular pattern called ModelView-Controller (MVC) divides an application's logic (Model), user interface (View), and control flow (Controller). A more structured codebase is made possible by this division, which permits independent changes to each layer. For example, the shopping cart can be used as a model, its contents can be rendered by various views, and user activities like adding or removing things are managed by controllers. The Repository design, which separates data access and manipulation to facilitate code testing and maintenance, is another pertinent design. Product data handling and inventory management benefit greatly from this arrangement.

Since it centralizes the instantiation logic, the Factory Pattern is useful for producing instances of complicated objects, such as various user account kinds or payment methods. Without making direct modifications to the client code, this centralization helps manage possible changes to object generation. These patterns streamline testing and monitoring while preserving performance, dependability, and usability.

References:

- Martin, R. C. (2002). Agile Software Development, Principles, Patterns, and Practices. Prentice Hall.
- Freeman, E., & Freeman, E. (2004). Head First Design Patterns. O'Reilly Media.