UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ZUMPANGO

LICENCIATURA EN INGENIERÍA EN COMPUTACIÓN

INGENIERIA DE SOFTWARE

Breve reflexión escrita

AUTORES:

ANGEL REYES CABRERA

JONATHAN MORALES JASSO

JOSÉ ERICK VARGAS RANGEL

FECHA: 22/05/24

La separación en capas dentro del desarrollo de software es un principio fundamental que busca mejorar la organización del código, la claridad de las responsabilidades y la capacidad de mantener, escalar y reutilizar el sistema a lo largo del tiempo. En nuestro proyecto de préstamo de bicicletas comunitarias, esta estructura resulta clave para lograr una implementación ordenada, donde cada parte del sistema cumple una función clara y delimitada.

La arquitectura por capas divide la aplicación en distintos niveles o módulos, generalmente definidos como: Capa de Presentación (UI), Capa de Lógica de Negocio (Servicio o Service), Capa de Acceso a Datos (DAO) y Modelo de Datos (Modelo). Cada una de estas capas tiene un propósito específico y no debería depender directamente de detalles internos de las otras, salvo mediante interfaces o servicios bien definidos.

Por ejemplo, la Capa de Presentación en nuestro proyecto está representada por la clase AppBicicletas, la cual ofrece un menú interactivo en consola para que el usuario pueda registrar bicicletas, registrar usuarios, solicitar préstamos o devoluciones, y consultar el estado del sistema. Esta capa se comunica con la capa de servicio para ejecutar acciones, pero no sabe cómo se almacenan los datos ni cuál es la lógica interna exacta para validar operaciones. Esto permite que cualquier cambio en la lógica del negocio o en la forma de almacenar los datos no afecte directamente la interfaz del usuario.

La Capa de Lógica de Negocio, compuesta por clases como BicicletaService, UsuarioService y PrestamoService, es la encargada de implementar todas las reglas que rigen el funcionamiento del sistema. Aquí es donde se determina si un usuario puede tomar prestada una bicicleta, si ya tiene un préstamo activo o si la bicicleta que quiere devolver está efectivamente prestada. Estas clases actúan como intermediarias entre la UI y la capa de datos, asegurando que todas las operaciones cumplan con las reglas definidas. Gracias a esta capa, podemos tener control total del comportamiento del sistema y asegurar que no se hagan operaciones inválidas.

Por su parte, la Capa de Acceso a Datos (DAO) se encarga de simular la persistencia de los datos. En este caso, usamos estructuras de datos como ArrayList para guardar objetos de tipo Usuario, Bicicleta o Prestamo. Las clases DAO como BicicletaDAO, UsuarioDAO y PrestamoDAO se encargan de agregar, buscar, listar y modificar los datos sin que el resto del sistema necesite saber cómo están organizados. Este aislamiento facilita que en el futuro podamos reemplazar la lógica de almacenamiento en memoria por una base de datos real, sin necesidad de modificar la lógica del negocio ni la interfaz del usuario.

El Modelo de Datos representa las entidades principales del sistema. En nuestro caso, son Usuario, Bicicleta y Prestamo. Cada una tiene atributos y métodos get y set que permiten acceder y modificar su información. Estas clases son compartidas entre las distintas capas, pero no contienen lógica compleja: solo representan datos. Esto hace que el modelo sea reutilizable y coherente a lo largo de toda la aplicación.

Una de las ventajas más evidentes de la separación en capas es que permite trabajar de forma colaborativa sin interferencias. Por ejemplo, mientras un integrante del equipo diseña la interfaz, otro puede estar desarrollando los servicios o escribiendo el código para acceder a los datos. Esto mejora la productividad y reduce conflictos, ya que cada integrante puede enfocarse en una responsabilidad distinta sin tener que entender todo el sistema de una vez.

Otra ventaja importante es la modularidad. Si más adelante se desea modificar una parte específica del sistema (como añadir un nuevo criterio de búsqueda en los DAO o implementar nuevas validaciones en los servicios), es posible hacerlo sin reescribir todo el proyecto. Esta capacidad de aislar los cambios facilita el mantenimiento del software y reduce el riesgo de introducir errores.

También es importante considerar que la arquitectura en capas promueve el uso de buenas prácticas de programación orientada a objetos, como la encapsulación y la abstracción. Cada clase tiene una función clara, los datos están protegidos mediante atributos privados y accesores, y las responsabilidades están distribuidas de manera lógica. Esto no solo mejora la calidad del código, sino que también lo hace más fácil de leer, probar y depurar.

En el ámbito educativo y de desarrollo académico, este enfoque también permite que los estudiantes y desarrolladores en formación entiendan mejor cómo funciona una aplicación completa, diferenciando entre los datos, la lógica y la interacción con el usuario. Facilita además la realización de pruebas unitarias en cada capa por separado, lo cual es esencial para garantizar la calidad del software.