Justificación de la implementación de PublicBusinessSystem (Opción b)

Para la consecución del Ejercicio 1 se optó por la opción b, es decir, se creó la clase PublicBusinessSystem que extiende BusinessSystem e implementa los interfaces ClientGateway y AdminGateway. Esta decisión se tomó con el objetivo de separar la lógica de negocio de las funcionalidades de acceso remoto.

La clase BusinessSystem contiene toda la lógica central de la aplicación, incluyendo gestión de usuarios, locales y reviews, así como operaciones de negocio internas. Si hubiéramos implementado los interfaces directamente en BusinessSystem (opción a), se habría mezclado la lógica de negocio con los detalles de comunicación remota, generando un acoplamiento innecesario que dificultaría la mantenibilidad y posibles ampliaciones del sistema.

Al crear PublicBusinessSystem como clase derivada, se consigue:

1. Encapsulación de la lógica de negocio: BusinessSystem sigue siendo independiente de RMI, manteniendo su diseño limpio y reutilizable.
2. Facilidad de publicación remota: PublicBusinessSystem expone únicamente los métodos definidos en los interfaces remotos (ClientGateway y AdminGateway), actuando como puente entre la aplicación interna y los clientes RMI.
3. Flexibilidad y escalabilidad: En el futuro, otros tipos de “gateways” o interfaces remotas podrían implementarse sin modificar la lógica central.

Informe de cambios en BModel y justificación

Para permitir la correcta utilización de RMI en el proyecto, fue necesario asegurar que las clases cuyos objetos son transferidos entre cliente y servidor fueran serializables. En concreto, se realizaron los siguientes cambios:

1. Implementación de la interfaz java.io.Serializable en todas las clases
2. Añadido de serialVersionUID en dichas clases para garantizar la compatibilidad de versiones durante la serialización y deserialización

Estos cambios son necesarios porque los objetos que se envían desde el servidor al cliente deben ser serializables para poder transferirse a través de la red. Inicialmente, algunas clases como Bar no implementaban Serializable, lo que provocaba excepciones java.io.NotSerializableException al invocar métodos remotos que retornaban instancias de estas clases. La adición de serialVersionUID no es obligatoria, pero es recomendable para asegurar que cambios futuros en la clase no rompan la compatibilidad con objetos serializados previamente.