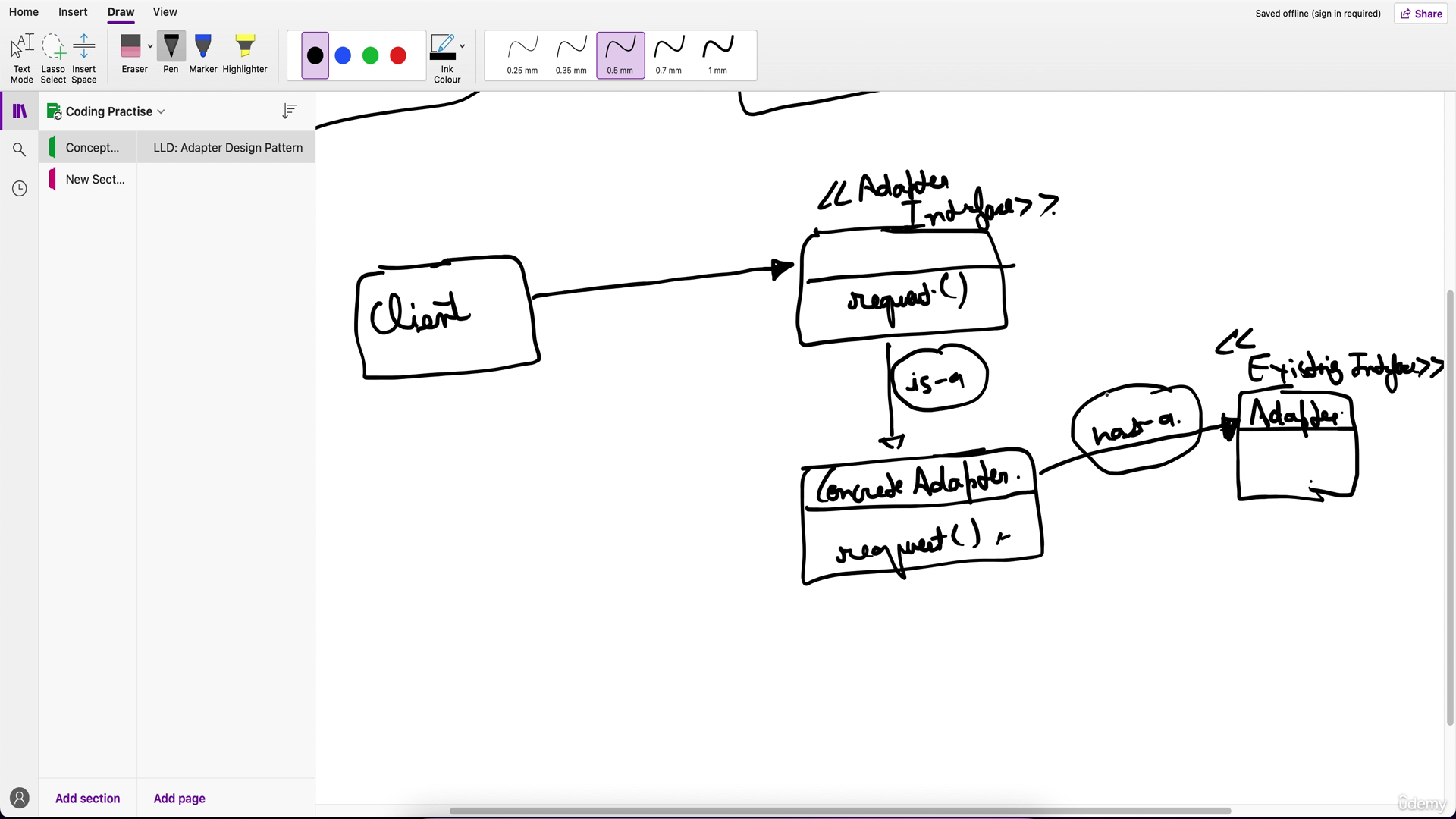
**Adapter Design Pattern**

The Adapter design pattern is a structural pattern that allows objects with incompatible interfaces to work together. It acts as a bridge between two incompatible interfaces by converting the interface of a class into another interface that a client expects. This pattern is often used to make existing classes work with others without modifying their source code.

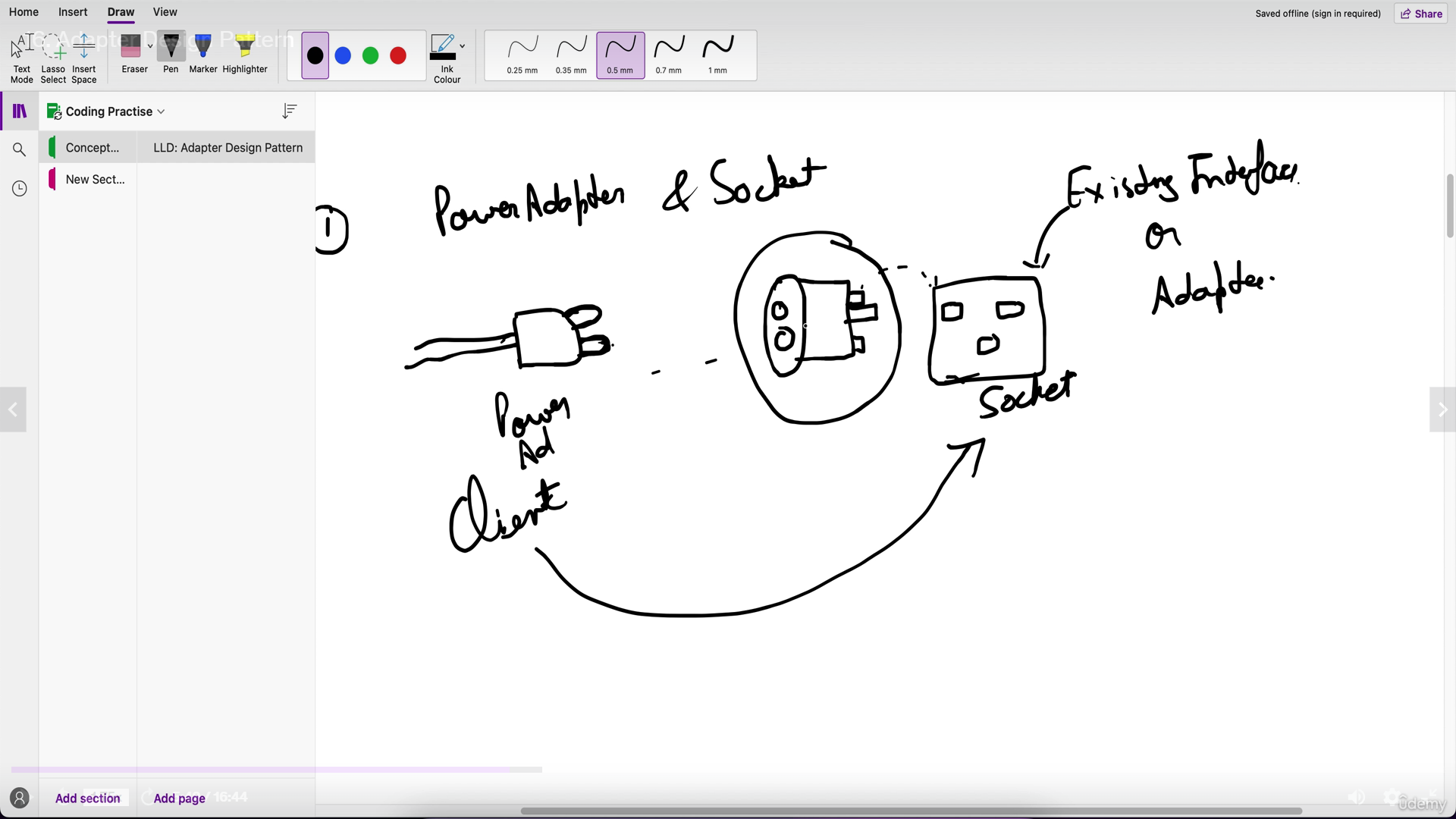
**Definition**

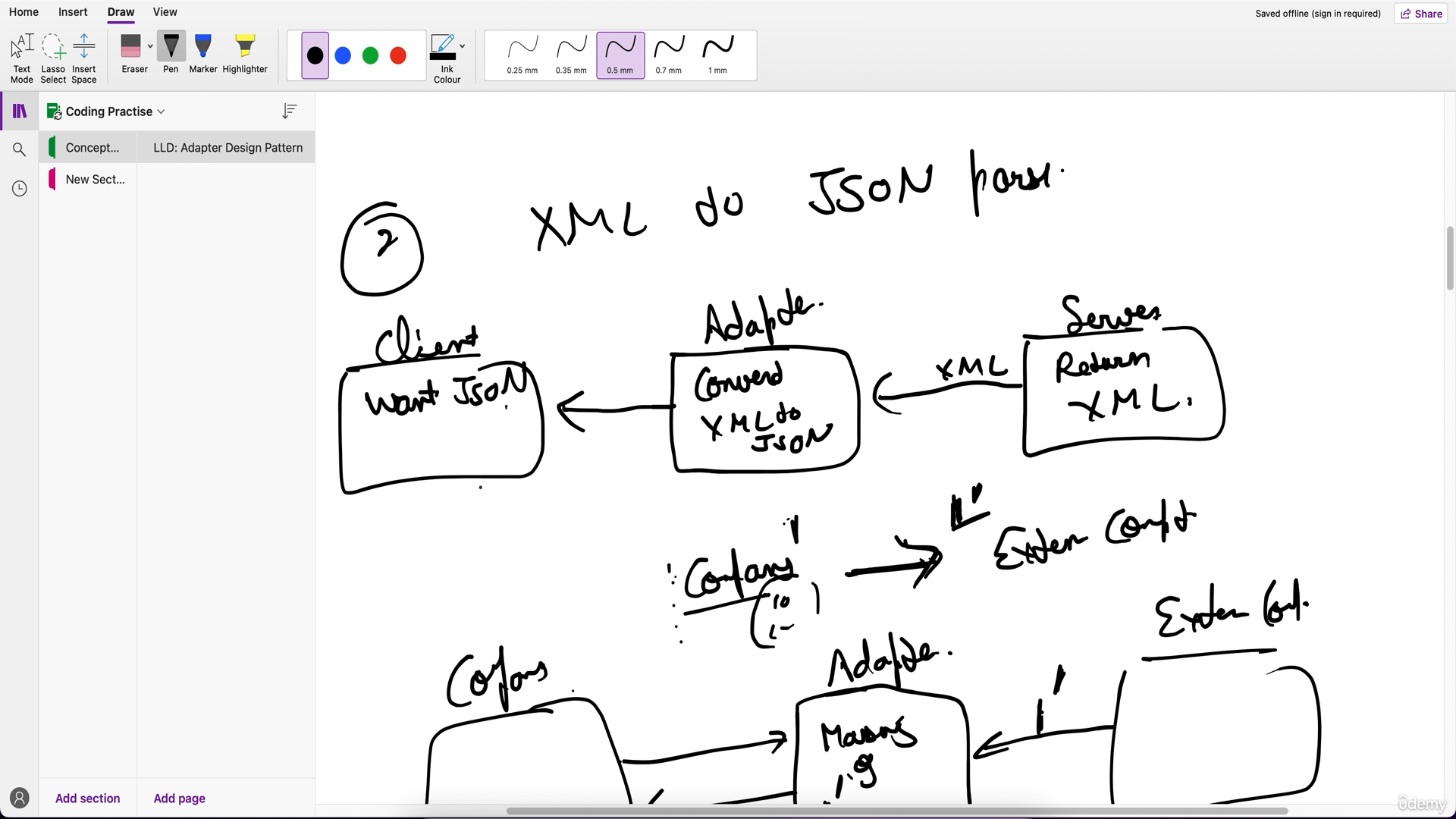
The Adapter pattern enables the integration of two incompatible interfaces by creating an intermediary adapter class that translates the requests and responses between them. This allows systems to use classes that would otherwise be incompatible, promoting reusability and flexibility.

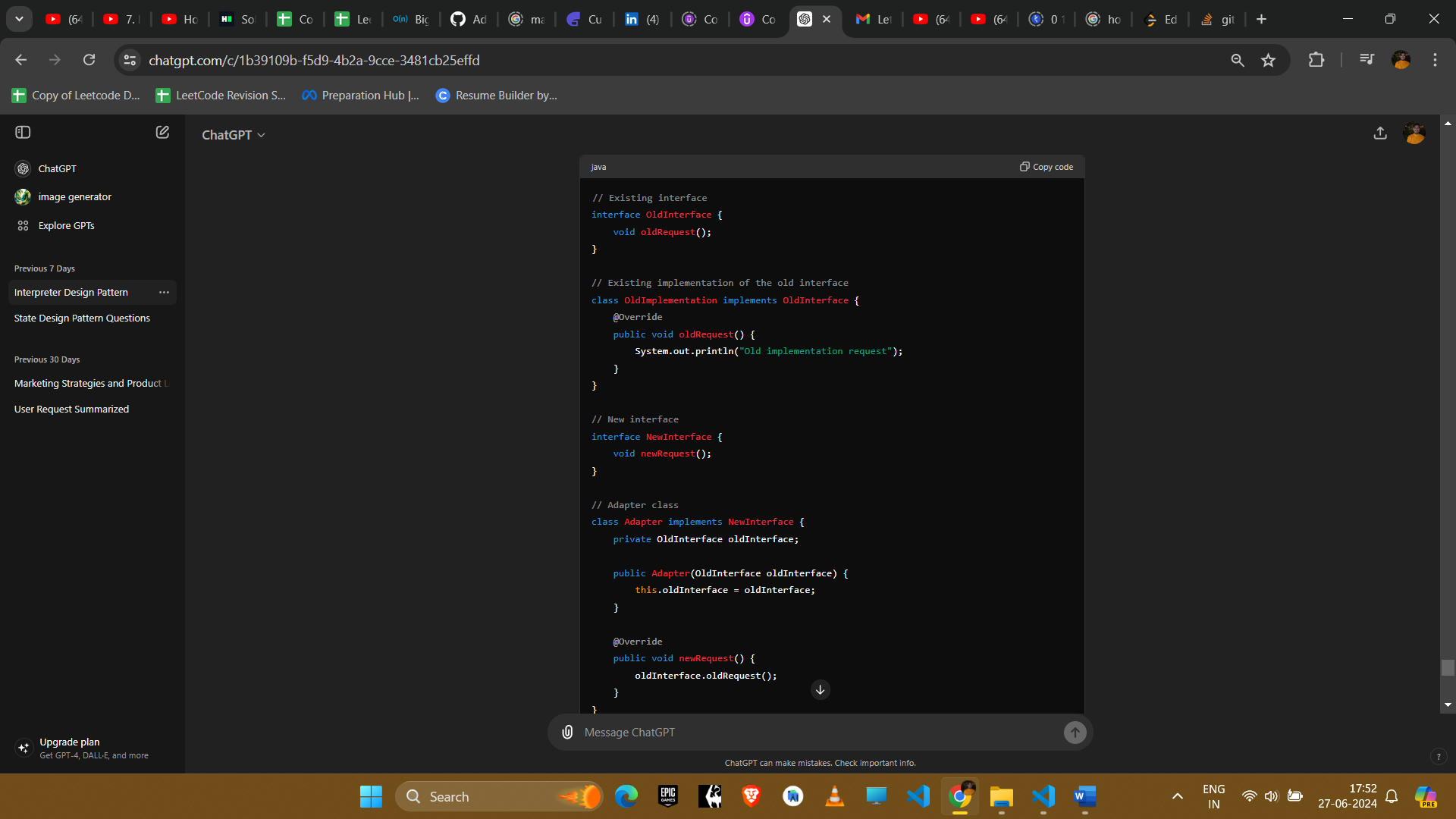


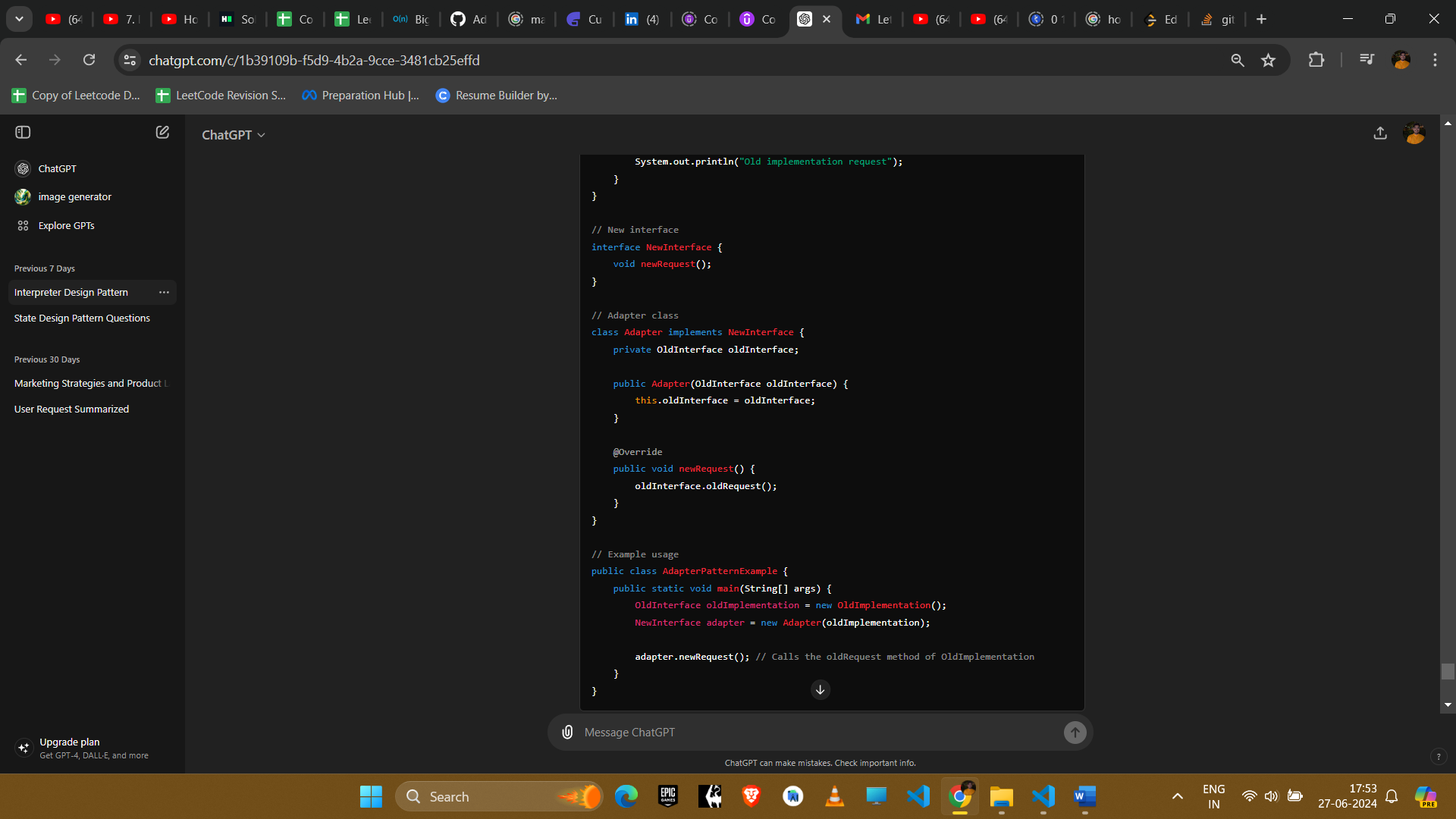
**Example**

Consider a scenario where you have a legacy system that uses an old interface, and you want to integrate it with a new system that uses a different interface.









**Explanation**

1. **Old Interface**: OldInterface defines the interface of the legacy system.
2. **Old Implementation**: OldImplementation is the concrete implementation of the OldInterface.
3. **New Interface**: NewInterface defines the interface expected by the new system.
4. **Adapter Class**: Adapter implements the NewInterface and holds a reference to an object of the OldInterface. It translates the newRequest call into an oldRequest call.

**Example Uses in Amazon Interviews**

1. **Integrating Legacy Systems**
   * **Scenario**: A company wants to integrate a legacy system with a new system, but the interfaces are incompatible.
   * **Implementation**: Use an adapter to translate between the legacy and new interfaces, allowing seamless integration.
2. **Third-Party Libraries**
   * **Scenario**: Using a third-party library with an interface that does not match the required interface of the application.
   * **Implementation**: Create an adapter to wrap the third-party library and present the expected interface to the application.
3. **Data Conversion**
   * **Scenario**: Converting data from one format to another while maintaining compatibility with existing systems.
   * **Implementation**: Use an adapter to convert data formats on-the-fly, ensuring compatibility without changing existing code.
4. **UI Components**
   * **Scenario**: Integrating new UI components with an existing UI framework that uses different interfaces.
   * **Implementation**: Create adapters to wrap new UI components and make them compatible with the existing framework.

**Conclusion**

The Adapter pattern is essential for integrating systems with incompatible interfaces, enhancing reusability and flexibility. By introducing an intermediary adapter class, this pattern allows seamless communication between different components without modifying their existing code, making it a valuable tool in software development and system integration.