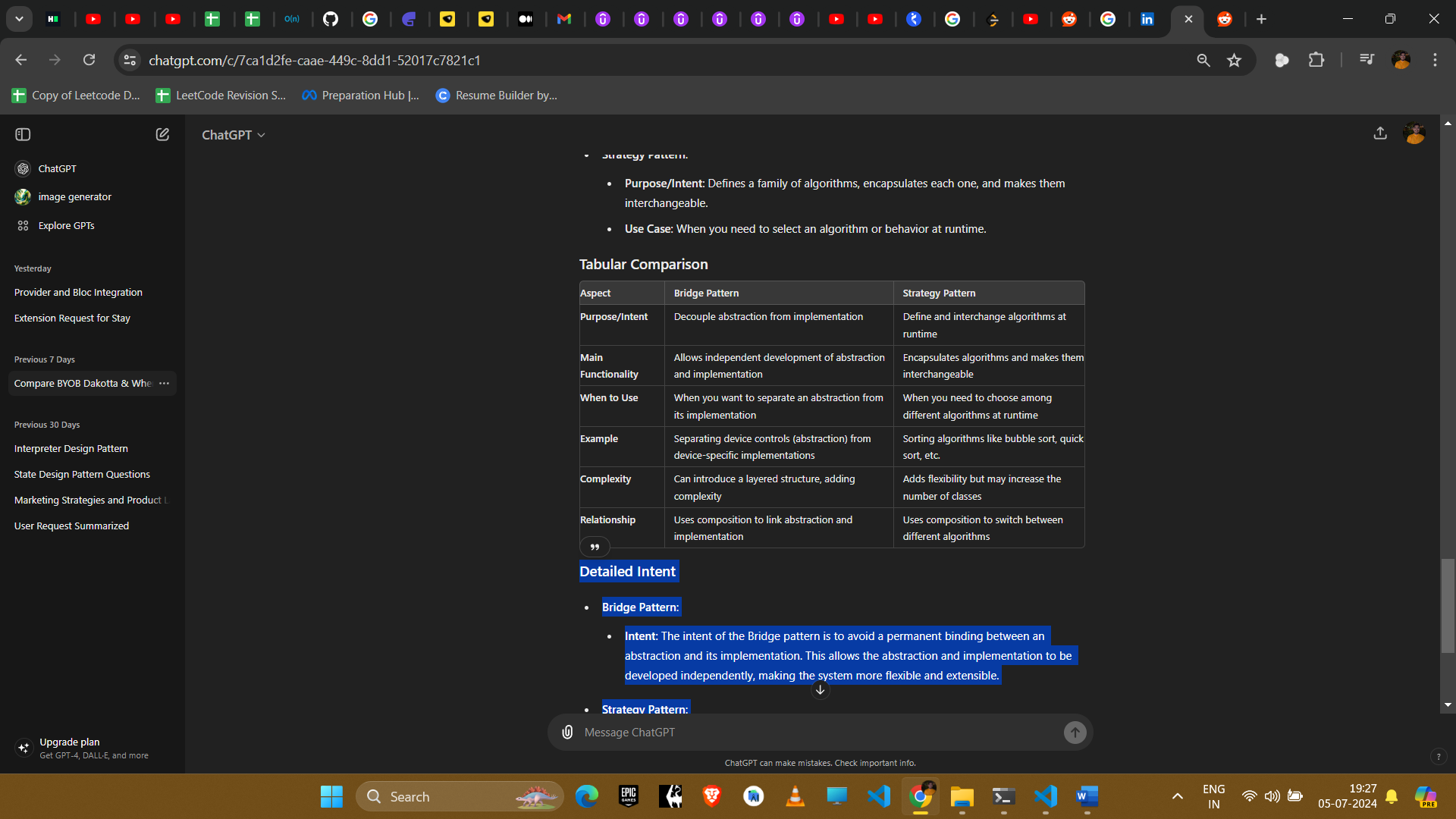
 **Bridge Pattern**:

* **Purpose/Intent**: Decouples an abstraction from its implementation so that the two can vary independently.
* **Use Case**: When you need to separate the abstraction and its implementation to enable independent changes.

 **Strategy Pattern**:

* **Purpose/Intent**: Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
* **Use Case**: When you need to select an algorithm or behavior at runtime.



**Detailed Intent**

* **Bridge Pattern**:
  + **Intent**: The intent of the Bridge pattern is to avoid a permanent binding between an abstraction and its implementation. This allows the abstraction and implementation to be developed independently, making the system more flexible and extensible.
* **Strategy Pattern**:
  + **Intent**: The intent of the Strategy pattern is to define a family of algorithms, encapsulate each one, and make them interchangeable. This pattern lets the algorithm vary independently from clients that use it, enabling dynamic selection of the algorithm based on the context.

In summary, the Bridge pattern focuses on decoupling the interface from the implementation to promote independent development and scalability. The Strategy pattern, however, focuses on defining a family of algorithms and making them interchangeable, providing flexibility in choosing the appropriate algorithm at runtime. Both patterns enhance the flexibility and maintainability of the system but address different design needs.