A design pattern in javascript is a reusable solution to a common problem in software design. It provides a structural way to address frequent coding challenges, improving code readability, scalability and maintainability.

Design pattern act as a templates / blueprint that developers can use to solve recurring issues in a consistent way, ensuring the application is flexible and easier to extend / refactor.

**Why use Design pattern in javascript?**

1. **Reusability**Pattern help reuse proven solutions to common problems.
2. **Code consistency**

Ensures a standard approach to solving problems across the project.

1. **Scalability**facilitates building applications that can grow without major code refactoring.
2. **Readability and maintainability**

Makes the codebase easier to understand and maintain.

1. **Improved collaboration**

Developers familiar with design pattern can collaborate more effectively by speaking the same language.

1. **Flexibility**

Promotes loosely coupled designs, making changes in one part of the application less likely to impact others.

**When to use design pattern?**

1. **Recurring problems**

When we encounter the same type of problems repeatedly.

1. **Complex systems**

For organizing code in large, modular applications.

1. **Team collaboration**

To ensures consistent coding practices across teams.

1. **Code maintainability**

to future-proof our application against changes.

Design patterns are generally divided into 3 main categories

1. **Creational patterns**

Creational pattern focuses on the process of object creation. These patterns provide mechanism to create objects in a way that ensures flexibility and reusability without exposing the complex instantiation logic to the user. They help make the system independent of how its objects are created, composed, and represented.

1. **Factory Pattern**
2. **Singleton Pattern**
3. **Builder Pattern**
4. **Prototype Pattern**
5. **Abstract Pattern**

Advantages

Promotes code reusability by standardizing object creation

Provides scalability for managing object creation complexity

Decouples object creation logic from the main application code.

Encourages flexible design, allowing changes without affecting existing code.

Disadvantages

1. **Structural patterns**

Structural design pattern deal with how objects and classes composed to form larger structures. These patterns focus on simplifying the design by ensuring that components work together efficiently, enabling flexibility, scalability, and easier maintenance.

1. **Adapter**
2. **Decorator**
3. **Façade**
4. **Proxy**
5. **Composite**
6. **Bridge**
7. **Flyweight**

**Advantages**

* Makes the code modular and flexible.
* Simplifies complex relationships between objects.
* Improves maintainability and reduces coupling.
* Encourages reuse of existing code components.

**Disadvantages**

Adds overhead when patterns are unnecessarily applied to simple scenarios.

Can increase complexity in small applications.

1. **Behavioral patterns**

Behavioral patterns focus on how objects and classes interact with each other and how responsibilities are distributed. They defines patterns for communication between object to make the system more flexible and efficient in carrying out its responsibilities.

Behavioral patterns deal with runtime behavior of objects and emphasize the delegation of responsibility, avoiding tight coupling between components.

1. **Observer Pattern**
2. **Strategy Pattern**
3. **Command Pattern**
4. **Iterator Pattern**
5. **Mediator Pattern**
6. **State Pattern**