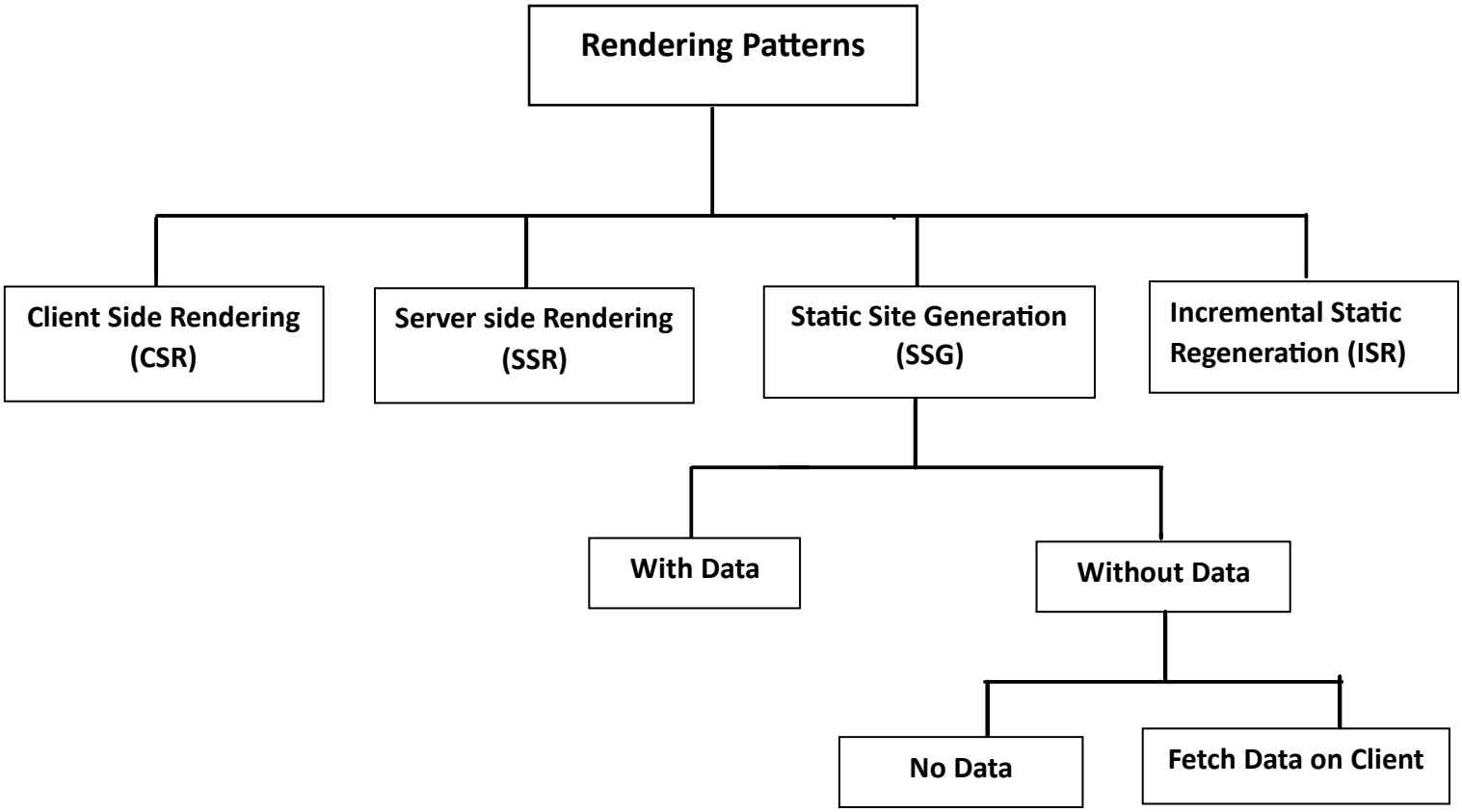
Rendering and Design Patterns

❖ Rendering Patterns:



Client-Side Rendering (CSR):

CSR, also known as client-side rendering, is a technique where the rendering process occurs on the client-side, typically in the user's web browser.

In CSR, the server sends a minimal HTML document along with JavaScript files to the client. The JavaScript code is responsible for fetching data from an API and manipulating the Document Object Model (DOM) to update the user interface.

- Advantages of CSR:

- Enhanced interactivity: CSR enables highly interactive web applications as the client can update the UI without making additional requests to the server. This results in a smooth and responsive user experience.
- Improved performance: Once the initial page load is complete, subsequent interactions can be faster since the client-side rendering avoids full page refreshes. Only the necessary components or data are fetched, reducing the server load and improving overall performance.

- Disadvantage client-side rendering:

- It requires more performant client devices. Clients have to execute JavaScript code and not just display static HTML.
- It slows down the first render because you must deliver the whole rendering machinery to the client.
- Caching is an issue—clients can cache for themselves and not other users. This means every user has to render HTML that has already been generated on multiple other clients.
- If you rely on search engine traffic for your application, optimizing for search engines with a client-rendered application can be a challenge.

➢ Use case of CSR:
Consider a social media feed that displays a list of posts. With CSR, the initial HTML and JavaScript are sent to the client. The JavaScript code then retrieves the post data from an API and dynamically renders the posts in the user's browser. As the user scrolls, additional posts can be fetched asynchronously, allowing for seamless pagination without reloading the entire page.

Server-Side Rendering (SSR):

SSR, or server-side rendering, is an alternative rendering technique where the server generates a fully rendered HTML page for each request and sends it to the client. In SSR, the server handles both the rendering and data fetching processes, ensuring that the client receives a complete and ready-to-display page.

➢ Advantages of SSR:
- Improved SEO: Search engine crawlers can easily parse the fully rendered HTML page, making SSR favourable for websites that heavily rely on organic search traffic. This approach ensures that search engines index the content accurately and comprehensively.
- Initial load performance: SSR can provide a better initial load performance compared to CSR because the server sends a fully rendered page. This can be particularly beneficial for content-heavy websites or applications where fast page load times are crucial.

➢ Disadvantages of server-side rendering:
- The server needs to be more performant than client-side rendering; the client does almost nothing.
- You can't use static web hosting services, as you'll need one that can run your server-side programming language.
- User interactions are limited with server-side rendering: Only standard HTML elements are available, and everything that needs additional JavaScript in the browser can make the application unwieldy over time.

- In case you want to avoid client-side JavaScript altogether, each change to the page will need a round trip to the server. If the client has an unreliable internet connection, this can quickly turn into a problem.

➢ Use case of SSR:
Imagine an e-commerce website with various product listings. With SSR, when a user visits a product page, the server renders the complete HTML, including the product information, images, and relevant data. This ensures that the user sees the entire page immediately, regardless of their device's processing power or internet speed.

Static Site Generation (SSG):

With static site generation, our third and final technique, the HTML is generated at compile time, for example, on a continuous delivery system. The generated HTML files then get uploaded to a static web server.

While this is the most straightforward out of the three techniques, it only works for straightforward interactions. Static site generation is mainly suitable for websites with content that isn't frequently changed.

The ideal use case scenarios for this approach are documentation or media galleries—websites where one party creates all the content, and everyone else consumes it without actively interacting with it.

There are also numerous static site generators for blogs. If you don't plan on frequently updating your content and don't need a fancy comment system to connect with your audience, this approach can be a good fit for your project.

➢ Advantages static site generation:
- This is another technique suitable for search engine optimization because the content search engines receive is the finished HTML.
- It offers quick delivery, as the web server only has to send the HTML without JavaScript overhead.
- The website is rendered once and can then be delivered to all clients, which allows for free caching.
- You can use a simple static web server to host the website.
- Because the build server is the one accessing the database and not the web server hosting the website, an attack on the web server doesn't give access to the database.

➢ disadvantages of static site generation
- It carries a high risk of stale content. Using client- or server-side rendering, content is rendered on demand for every request, but with static site generation you have to create a version explicitly.

- There’s a higher performance demand for the machine that renders the page. You will generate all pages in one go, so with heavier pages updating the rendering can take some time.

- Use case of SSG:
  Static Site Generation (SSG) rendering pattern is commonly used in web development for creating websites where the content is generated at build time rather than runtime.
  Blog or Content Management System (CMS):
  Imagine you're building a blog or a CMS where content is added or updated frequently. By using SSG, you can pre-generate HTML files for each page or post during the build process. This means that when a user requests a page, the server can simply serve the pre-generated HTML file without needing to dynamically generate it on the fly. This greatly reduces server load and response times, making the website faster and more scalable.

Incremental Static Regeneration (ISR):

Incremental Static Regeneration, or ISR, is a concept primarily associated with Next.js, a React-based framework used for building web applications. This technique allows developers to update static pages after they have been generated, eliminating the need to rebuild the entire site. ISR is especially beneficial for large sites with numerous static pages that don’t frequently change, but where some pages might need more regular updates.

What is Incremental Static Regeneration?

During the initial build process, static pages are generated and served to users. Developers can specify a revalidation period for each static page. Once this period elapses, the next request to the page triggers a regeneration of the page in the background. While the page is being regenerated, the old (stale) version of the page continues to be served to users.

Upon completion of the regeneration, the old version of the page is replaced with the new one for subsequent requests. This method combines the advantages of static generation, such as performance and reliability, with the ability to update content dynamically when necessary. It is particularly useful for sites with content that changes periodically but not constantly, like e-commerce sites, blogs, or news sites.

❖ Design Patterns:

What is Pattern Designing?

In software development, pattern designing refers to the application of design patterns, which are reusable and proven solutions to common problems encountered during the design and implementation of software systems. Design patterns are general solutions to common problems that arise during the design and implementation of software systems. They provide a set of proven solutions to design challenges and promote best practices in software development.

Characteristics of Pattern Designing:

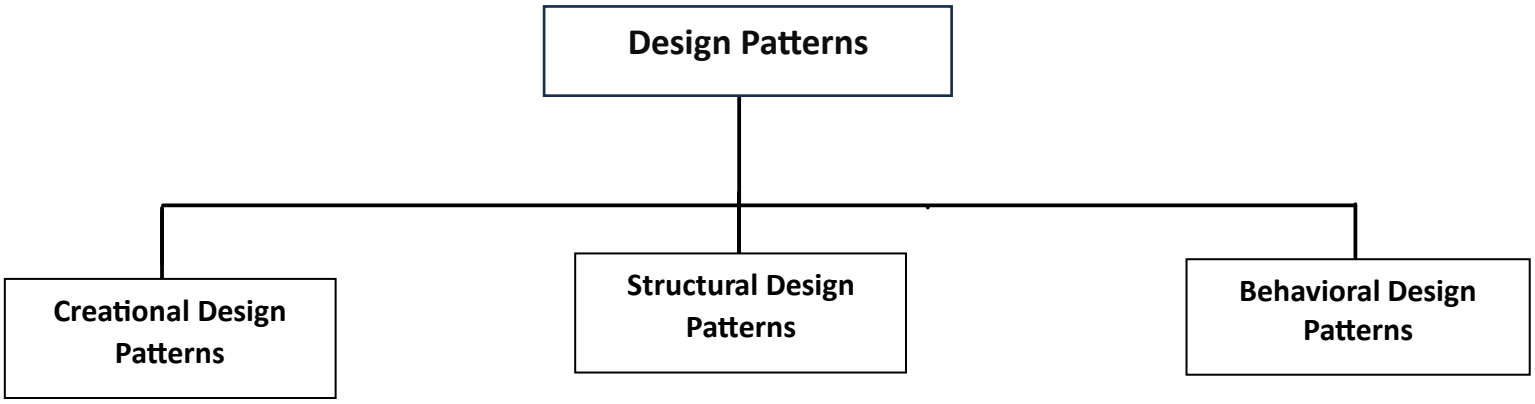
- Problem-Solution Approach: Design patterns follow a problem-solution approach. They identify common design problems and provide well-defined solutions that have been proven effective in similar contexts.
- Reusability: Design patterns promote code reusability by encapsulating successful design practices. Once a design pattern is established, it can be applied to various projects, saving time and effort.
- Abstraction: Patterns abstract away specific implementation details and focus on high-level design concepts. They provide a way to communicate and document design decisions in a standardized and understandable manner.
- Common Vocabulary: Design patterns establish a common vocabulary and set of terms that developers can use to discuss and communicate design concepts. This helps in fostering a shared understanding among team members.
- Proven Solutions: Design patterns are not arbitrary solutions; they are based on the collective experience of the software development community. They represent solutions that have been tried and tested in real-world scenarios.

What are Design Patterns?

- A design pattern provides a general reusable solution for the common problems that occur in software design. The pattern typically shows relationships and interactions between classes or objects.
- The idea is to speed up the development process by providing well-tested, proven development/design paradigms.
- Design patterns are programming language-independent strategies for solving a common problem.
- That means a design pattern represents an idea, not a particular implementation. By using design patterns, you can make your code more flexible, reusable, and maintainable.
- It’s not mandatory to always implement design patterns in your project. Design patterns are not meant for project development. Design patterns are meant for common problem-solving. Whenever there is a need, you have to implement a suitable pattern to avoid such problems in the future.

- To find out which pattern to use, you just have to try to understand the design patterns and their purposes. Only by doing that, you will be able to pick the right one.

Types of Design Patterns:



1.Creational Design Patterns:

Creational design patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented. A class creational pattern uses inheritance to vary the class that's instantiated, whereas an object creational pattern will delegate instantiation to another object. Creational patterns give a lot of flexibility in what gets created, who creates it, how it gets created, and, when.

There are two recurring themes in these patterns:

- They all encapsulate knowledge about which concrete class the system uses.
- They hide how instances of these classes are created and put together.

Example:

Consider an example, a drawing editor that lets users draw and arrange graphical elements (lines, polygons, text, etc.) into pictures and diagrams. The drawing editor's key abstraction is the graphical object, which has an editable shape and can draw itself.

The interface for graphical objects is defined by an abstract class called Shape. The editor defines a subclass of the Shape for each kind of graphical object: a **LineShape** class for lines, a **PolygonShape** class for polygons, and so forth.

Types:

- ✓ Factory Method Design Pattern
- ✓ Abstract Factory Method Design Pattern
- ✓ Singleton Method Design Pattern

- ✓ Prototype Method Design Pattern.
- ✓ Builder Method Design Pattern

2.Structural Design Patterns:

Structural Design Patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations. Consider how multiple inheritances mix two or more classes into one. The result is a class that combines the properties of its parent classes.

There are two recurring themes in these patterns:

- This pattern is particularly useful for making independently developed class libraries work together.
- Structural Design Patterns describe ways to compose objects to realize new functionality.
- The added flexibility of object composition comes from the ability to change the composition at run-time, which is impossible with static class composition.

Types:

- ✓ Adapter Method Design Patterns:
- ✓ Bridge Method Design Patterns
- ✓ Composite Method Design Patterns.
- ✓ Decorator Method Design Patterns
- ✓ Facade Method Design Patterns
- ✓ Flyweight Method Design Patterns
- ✓ Proxy Method Design Patterns

Example:

One example of a structural design pattern is the "Adapter" pattern. This pattern allows incompatible interfaces to work together by creating a bridge between them. For instance, imagine you have an old library that uses a legacy API, and you want to integrate it with a new system that uses a different interface. You can use the Adapter pattern to create a wrapper class that translates the calls from the new system into calls that the old library understands, thus allowing them to interact seamlessly.

3.Behavioral Design Patterns:

Behavioral Patterns are concerned with algorithms and the assignment of responsibilities between objects. Behavioral patterns describe not just patterns of objects or classes but also the patterns of communication between them. These patterns characterize complex control flow that’s difficult to follow at run-time.

There are three recurring themes in these patterns:

- Behavioral class patterns use inheritance to distribute behavior between classes.
- Behavioral object patterns use object composition rather than inheritance.
- Behavioral object patterns are concerned with encapsulating behavior in an object and delegating requests to it.

Types:

- ✓ Chain of Responsibility Method Design Pattern
- ✓ Command Method Design Pattern
- ✓ Interpreter Method Design Patterns
- ✓ Mediator Method Design Pattern
- ✓ Memento Method Design Patterns
- ✓ Observer Method Design Pattern
- ✓ State Method Design Pattern
- ✓ Strategy Method Design Pattern
- ✓ Template Method Design Pattern
- ✓ Visitor Method Design Pattern

Example:

This pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. For example, in a weather application, multiple displays (observers) might be interested in changes to the weather data (subject). The Observer pattern allows these displays to be notified whenever the weather data changes, ensuring they always display the most up-to-date information.

Advantages of Pattern Designing

- Reusable Solutions:
- Scalability
- Abstraction and Communication
- Maintainability
- Speeds Up Development
- Tool for Problem Solving

Disadvantages of Pattern Design:

- Learning Curve
- Overengineering.
- Rigidity
- Applicability Concerns
- Maintenance Challenges