**A REPORT ON**

**“RENDERING AND DESIGN PATTERN”**

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE INTERNSHIP PROVIDED BY

**SANKEY SOLUTIONS AS SOLUTION ANALYST**

**SUBMITTED BY**

**Ms. Manasi Gopal Narkhede**

OF

**ABMSP’S ANANTRAO PAWAR COLLEGE OF ENGINEERING & RESEARCH, PARVATI , PUNE 411009**

**ABSTRACT**

*Rendering and design patterns play crucial roles in the development of interactive applications, providing a framework for creating visually appealing and efficient user interfaces. The fundamentals of rendering and various design patterns commonly employed in the field are explained. Rendering, the process of generating visual output from data, is a critical aspect of creating engaging and responsive user interfaces. Different rendering patterns offer unique solutions to address specific use cases and challenges.* *Rendering patterns tell how web content is generated, processed, and presented to users, influencing aspects such as performance, interactivity, and user experience. Rendering patterns such as Client-side Rendering, Incremental Static Generation, Progressive Hydration, Selective Hydration, React Server Components, Server-side Rendering, Static Rendering, and Streaming Server-Side Rendering are explored and elaborated along with their applicability and suitability for different scenarios. Design patterns offer reusable and proven solutions to recurring challenges, promoting maintainability, scalability, and flexibility in software design. Design patterns such as Compound Pattern, Higher Order Component (HOC) Pattern, Hooks Pattern, Container/Presentational Pattern, and Render Props Pattern are explored and elaborated with their characteristics, applications, and best practices.*

**TABLE OF CONTENTS**

**Sr. No. Title of Chapter Page No.**

1. **Title 3**
2. **Introduction 4**

2.1. Rendering Pattern 4

2.1.1. Client-side Rendering (CSR) 4

2.1.2. Incremental Static Generation 4

2.1.3. Progressive Hydration 4

2.1.4. Selective Hydration 5

2.1.5. React Server Components 5

2.1.6. Server-side Rendering (SSR) 5

2.1.7. Static Rendering 5

2.1.8. Streaming Server-Side Rendering 6

2.2. Design Pattern 6

2.2.1. Compound Pattern 6

2.2.2. Higher Order Component (HOC) Pattern 6

2.2.3. Hooks Pattern 7

2.2.4. Container/Presentational Pattern 7

2.2.5. Render Props Pattern 7

**3. Conclusion 8**

1. **TITLE**

Rendering and Design Patterns

1. **INTRODUCTION**
   1. **Rendering Pattern:**

Rendering is the process of generating images or visual outputs from a set of data, often in the context of computer graphics and user interface development. The primary goal is to create a representation of data that is easily interpretable by users. Rendering patterns tell how web content is generated, processed, and presented to users, influencing aspects such as performance, interactivity, and user experience. Different rendering patterns offer different solutions to address specific use cases and challenges.

Rendering Patterns can be broadly categorized into:

**2.1.1. Client-side Rendering (CSR):**

Client-side Rendering is well-suited for dynamic web applications where interactivity is crucial. It is commonly used in Single Page Applications (SPAs), allowing the client's browser to handle rendering. CSR is suitable for applications requiring real-time updates, dynamic content loading, and interactive user interfaces.

**2.1.2. Incremental Static Generation:**

Incremental Static Generation is suitable for content-heavy websites with frequently changing data. It allows pre-rendering pages at build time and incrementally updating them in response to data changes. This pattern is beneficial for maintaining performance while ensuring that the content remains up-to-date.

**2.1.3. Progressive Hydration:**

Progressive Hydration is applicable in scenarios where a balance between initial page load performance and interactivity is required. It involves rendering a basic version of the page on the server and progressively enhancing it on the client side. This approach is useful for providing a fast initial user experience while allowing for more complex interactions as resources become available.

**2.1.4. Selective Hydration:**

Selective Hydration is suitable for optimizing performance in large-scale applications. It involves loading only the necessary parts of the application on the client side, minimizing the initial load time. This pattern is beneficial for improving perceived performance and reducing the amount of JavaScript that needs to be loaded initially.

**2.1.5. React Server Components:**

React Server Components are designed for applications with a need for server-rendered components that can be seamlessly updated on the client side. This pattern is suitable for creating highly interactive applications while maintaining server-side rendering capabilities. React Server Components offer a balance between server and client responsibilities.

**2.1.6. Server-side Rendering (SSR):**

Server-side Rendering is applicable when search engine optimization (SEO) is a priority or when initial page load performance is crucial. It involves rendering pages on the server and sending fully-formed HTML to the client. SSR is beneficial for content-focused websites and applications where fast initial rendering is a priority.

**2.1.7. Static Rendering:**

Static Rendering is well-suited for content that doesn't change frequently. It involves pre-rendering entire pages at build time and serving them as static assets. This pattern is efficient for websites with relatively stable content, reducing the need for server processing during runtime.

**2.1.8. Streaming Server-Side Rendering:**

Streaming Server-Side Rendering is beneficial for optimizing the time-to-first-byte (TTFB) and providing a faster initial loading experience. It involves streaming HTML content progressively to the client as it's generated on the server. This pattern is suitable for applications prioritizing quick initial rendering over interactivity.

* 1. **Design Patterns:**

Design patterns are recurring solutions to common problems faced during software design. Design patterns offer reusable and proven solutions to recurring challenges, promoting maintainability, scalability, and flexibility in software design.

Design Patterns are as follows:

**2.2.1. Compound Pattern:**

The Compound Pattern involves combining multiple design patterns to address complex challenges. This approach allows developers to leverage the strengths of different patterns to create a comprehensive solution. It is particularly useful in large-scale applications where diverse functionalities and architectural concerns need to be addressed.

**2.2.2. Higher Order Component (HOC) Pattern:**

The HOC pattern involves creating a higher-order component that encapsulates the logic or behaviour shared across multiple components. This pattern promotes code reuse, abstraction, and separation of concerns. HOCs are useful in scenarios where cross-cutting concerns such as authentication, logging, or state management need to be applied consistently across components.

**2.2.3. Hooks Pattern:**

The Hooks Pattern is a paradigm introduced in React that allows functional components to use state and lifecycle features. Hooks, such as useState and useEffect, enable functional components to manage state and side effects, reducing the reliance on class components. The Hooks Pattern simplifies component logic and encourages the development of more concise and readable code.

**2.2.4. Container/Presentational Pattern:**

The Container/Presentational Pattern emphasizes a clear separation between components responsible for managing state (Containers) and components focused on rendering UI (Presentational). Containers are responsible for data fetching, state management, and business logic, while Presentational components focus on rendering the UI. This pattern enhances maintainability and testability by isolating concerns.

**2.2.5. Render Props Pattern:**

The Render Props Pattern involves passing a function as a prop to a component, allowing the component to render content based on the provided function. This pattern promotes component reusability and flexibility. It is commonly used when components need to share logic or behaviours dynamically without resorting to higher-order components.

1. **CONCLUSION**

Rendering and design patterns are integral components of software development, especially in the realm for user interface design and computer graphics. Understanding different rendering patterns is crucial for web developers to make informed decisions based on the specific requirements of their projects. The choice of a rendering pattern depends on factors such as the nature of the content, desired user experience, and performance considerations. By selecting the most suitable rendering pattern, developers can optimize both initial page load times and subsequent interactions, ultimately enhancing the overall quality of web applications. Understanding and applying design patterns is crucial for building maintainable and scalable web applications. The design patterns offer developers proven solutions to common challenges, enabling the creation of robust and flexible software architectures. As the web development landscape evolves, staying informed about these patterns and their best practices empowers developers to make informed design decisions, resulting in more efficient and maintainable codebases. By understanding the principles behind rendering and incorporating appropriate design patterns, developers can create efficient, scalable, and visually appealing applications tailored to specific use cases. The combination of these concepts contributes to the development of robust and maintainable software systems in the rapidly evolving landscape of interactive computing.