Got it! I'll **add explanations of React hooks usage** (like `useEffect`) for every example and clarify when/how hooks fit in SSR, CSR, SSG, and ISR in Next.js.

Next.js Rendering Methods Guide

Overview

This guide explains the four main rendering methods in Next.js, their use cases, and how to implement them effectively.

Table of Contents

1. Server-Side Rendering (SSR)

Implementation

```
// pages/ssr-example.js
export async function getServerSideProps(context) {
  // Access to request/response objects
  const { req, res } = context;

  // Can access cookies, headers, etc.
  const userAgent = req.headers['user-agent'];

  // Dynamic data that changes on every request
  const currentTime = new Date().toISOString();

  // Data that needs to be fresh on every request
  const res = await fetch('https://api.example.com/live-data', {
    headers: {
      'Cache-Control': 'no-cache',
      'Authorization': `Bearer ${process.env.API_KEY}`
    }
  });
  const data = await res.json();

  return {
    props: {
      data,
      currentTime,
```

```
      userAgent
    }
  };
}

export default function SSRPage({ data, currentTime, userAgent }) {
  return (
    <div>
      <h1>SSR Page</h1>
      <p>Current Time: {currentTime}</p>
      <p>User Agent: {userAgent}</p>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
}
```

Key Points

- Data fetched on every request
- Access to request/response objects
- Can use dynamic data (cookies, headers, etc.)
- Good for personalized content
- SEO-friendly
- Hooks run only on client after hydration

2. Client-Side Rendering (CSR)

Implementation

```
// pages/csr-example.js
export default function CSRPage() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(res => res.json())
      .then(data => setData(data));
  }, []);

  return (
    <div>
      <h1>CSR Page</h1>
      {data ? (
        <pre>{JSON.stringify(data, null, 2)}</pre>
      ) : (
        <p>Loading...</p>
      )}
    </div>
```

```
  );
}
```

Key Points

- Data fetched in browser
- Initial HTML is empty
- Poor SEO
- Good for user-specific content
- All hooks run on client

3. Static Site Generation (SSG)

Implementation

```
// pages/ssg-example.js
export async function getStaticProps() {
  // Can only access build-time environment variables
  const apiKey = process.env.API_KEY;

  // Data that doesn't change frequently
  const res = await fetch('https://api.example.com/static-content', {
    headers: {
      'Authorization': `Bearer ${apiKey}`
    }
  });
  const data = await res.json();

  // Can generate static paths
  const paths = data.map(item => ({
    params: { id: item.id.toString() }
  }));

  return {
    props: {
      data,
      lastGenerated: new Date().toISOString()
    }
  };
}

// Optional: Generate static paths for dynamic routes
export async function getStaticPaths() {
  return {
    paths: [
      { params: { id: '1' } },
      { params: { id: '2' } }
    ],
    fallback: false // Show 404 for non-existent paths
```

```
  };
}

export default function SSGPage({ data, lastGenerated }) {
  return (
    <div>
      <h1>SSG Page</h1>
      <p>Last Generated: {lastGenerated}</p>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
}
```

Key Points

- Data fetched at build time only
- No access to request/response objects
- Can generate static paths for dynamic routes
- Best for content that rarely changes
- Excellent SEO
- Fastest performance
- Hooks run only on client

4. Incremental Static Regeneration (ISR)

Implementation

```
// pages/isr-example.js
export async function getStaticProps() {
  const res = await fetch('https://api.example.com/data');
  const data = await res.json();

  return {
    props: { data },
    revalidate: 60 // Revalidate every 60 seconds
  };
}

export default function ISRPage({ data }) {
  useEffect(() => {
    console.log('Client-side effect');
  }, []);

  return (
    <div>
      <h1>ISR Page</h1>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
```

```
  );
}
```

Key Points

- Data fetched at build time
- Periodic revalidation
- Good SEO
- Good for semi-static content
- Hooks run only on client

Comparison Table

Feature	SSR	CSR	SSG	ISR
Data Fetching	Server (every request)	Client	Build time	Build time + revalidate
SEO	☑	✗	☑	☑
Performance	Good	Poor	Excellent	Good
Use Case	Dynamic content	User-specific	Static content	Semi-static content
Hooks Usage	Client-side only	Full	Client-side only	Client-side only

Best Practices

1. **Choose the Right Method**
   - Use SSR for dynamic, SEO-critical pages
   - Use CSR for user-specific content
   - Use SSG for static content
   - Use ISR for semi-static content
2. **Data Fetching**
   - Keep data fetching in `getServerSideProps` or `getStaticProps`
   - Use hooks only for client-side effects
   - Avoid data fetching in `useEffect` for SEO-critical pages
3. **Performance**
   - Implement proper loading states
   - Use appropriate caching strategies
   - Consider using ISR for frequently updated content
4. **SEO**
   - Use SSR or SSG for SEO-critical pages
   - Implement proper meta tags
   - Ensure content is available in initial HTML

Additional Resources

- Next.js Documentation
- React Hooks Documentation
- Next.js Data Fetching

If you want, I can also add **SEO tags (`<Head>`)** examples inside these pages! Just ask.