SYSTEM DESIGN QUESTIONS

1. How would you optimize a React application for performance?

Answer:

Optimization Techniques:

1. **Code Splitting:**

jsx
```jsx
import React, { Suspense, lazy } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}
```

2. **Memoization:**

jsx
```jsx
import React, { memo, useMemo, useCallback } from 'react';

const ExpensiveComponent = memo(({ data, onItemClick }) => {
  const processedData = useMemo(() => {
    return data.map(item => ({
      ...item,
      processed: heavyCalculation(item)
    }));
  }, [data]);

  return (
    <div>
      {processedData.map(item => (
        <Item
          key={item.id}
          item={item}
          onClick={onItemClick}
        />
      ))}
    </div>
```

```
  );
});

function Parent() {
  const [items, setItems] = useState([]);

  const handleItemClick = useCallback((id) => {
    setItems(prev => prev.map(item =>
      item.id === id ? { ...item, selected: !item.selected } :
item
    ));
  }, []);

  return <ExpensiveComponent data={items}
onItemClick={handleItemClick} />;
}
```

3. **Virtual Scrolling:**

jsx

```
import React, { useState, useMemo } from 'react';

function VirtualList({ items, itemHeight = 50, containerHeight =
400 }) {
  const [scrollTop, setScrollTop] = useState(0);

  const visibleItems = useMemo(() => {
    const startIndex = Math.floor(scrollTop / itemHeight);
    const endIndex = Math.min(
      startIndex + Math.ceil(containerHeight / itemHeight) + 1,
      items.length
    );

    return items.slice(startIndex, endIndex).map((item, index) =>
({
      ...item,
      index: startIndex + index
    }));
  }, [items, scrollTop, itemHeight, containerHeight]);

  return (
    <div
      style={{ height: containerHeight, overflow: 'auto' }}
      onScroll={(e) => setScrollTop(e.target.scrollTop)}
    >
      <div style={{ height: items.length * itemHeight, position:
'relative' }}>
        {visibleItems.map(item => (
```

```
            <div
              key={item.id}
              style={{
                position: 'absolute',
                top: item.index * itemHeight,
                height: itemHeight,
                width: '100%'
              }}
            >
              {item.content}
            </div>
          ))}
        </div>
      </div>
    );
}
```

2. Design a scalable component architecture.

Answer:

Folder Structure:

```
src/
├── components/
│   ├── common/
│   │   ├── Button/
│   │   │   ├── Button.jsx
│   │   │   ├── Button.test.js
│   │   │   ├── Button.module.css
│   │   │   └── index.js
│   │   └── Modal/
│   ├── features/
│   │   ├── UserProfile/
│   │   └── Dashboard/
├── hooks/
├── utils/
├── services/
└── contexts/
```

Component Design Pattern:

jsx

```jsx
// components/common/Button/Button.jsx
import React from 'react';
import PropTypes from 'prop-types';
import styles from './Button.module.css';
```

```jsx
const Button = ({
  children,
  variant = 'primary',
  size = 'medium',
  disabled = false,
  loading = false,
  onClick,
  type = 'button',
  ...rest
}) => {
  const buttonClasses = [
    styles.button,
    styles[variant],
    styles[size],
    disabled && styles.disabled,
    loading && styles.loading
  ].filter(Boolean).join(' ');

  return (
    <button
      type={type}
      className={buttonClasses}
      disabled={disabled || loading}
      onClick={onClick}
      {...rest}
    >
      {loading ? <Spinner size="small" /> : children}
    </button>
  );
};

Button.propTypes = {
  children: PropTypes.node.isRequired,
  variant: PropTypes.oneOf(['primary', 'secondary', 'danger']),
  size: PropTypes.oneOf(['small', 'medium', 'large']),
  disabled: PropTypes.bool,
  loading: PropTypes.bool,
  onClick: PropTypes.func,
  type: PropTypes.oneOf(['button', 'submit', 'reset'])
};

export default Button;
```

Custom Hooks:

jsx

```jsx
// hooks/useApi.js
```

```
import { useState, useEffect } from 'react';

export function useApi(url, options = {}) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true);
        setError(null);

        const response = await fetch(url, options);
        if (!response.ok) {
          throw new Error(`HTTP error! status:
${response.status}`);
        }

        const result = await response.json();
        setData(result);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error };
}

// hooks/useLocalStorage.js
import { useState, useEffect } from 'react';

export function useLocalStorage(key, initialValue) {
  const [storedValue, setStoredValue] = useState(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      console.error('Error reading from localStorage:', error);
      return initialValue;
    }
  });
```

```
  const setValue = (value) => {
    try {
      setStoredValue(value);
      window.localStorage.setItem(key, JSON.stringify(value));
    } catch (error) {
      console.error('Error writing to localStorage:', error);
    }
  };

  return [storedValue, setValue];
}
```