React Coding Challenges

Interview Questions and Answers

Core Concepts

This section focuses on fundamental principles and advanced concepts that an experienced developer should master.

1. Implement a custom useDebounce hook that delays the execution of a function

Solution:

```
const useDebounce = (value, delay) => {
  const [debouncedValue, setDebouncedValue] = useState(value);
  useEffect(() => {
    const timer = setTimeout(() => setDebouncedValue(value), delay);
    return () => clearTimeout(timer);
  }, [value, delay]);
  return debouncedValue;
}
```

Key points:

- Uses setTimeout for delayed execution
- Cleans up previous timeouts
- Returns memoized value

2. Create a custom usePrevious hook that returns the previous value of a state

Solution:

```
const usePrevious = (value) => {
  const ref = useRef();
  useEffect(() => {
    ref.current = value;
  }, [value]);
  return ref.current;
}
```

Explanation:

- Uses useRef to store previous value
- Updates ref in useEffect after render
- Returns previous value from ref

3. Implement a reusable Modal component with portal and accessibility features

Solution:

```
const Modal = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;
  return createPortal(




{children}



,
    document.body
  );
```

Features:

- Uses React Portal
- ARIA attributes for accessibility
- Backdrop click handling

4. Create a custom useLocalStorage hook for persistent state management

Solution:

```
const useLocalStorage = (key, initialValue) => {
  const [storedValue, setStoredValue] = useState(() => {
    try {
      return JSON.parse(localStorage.getItem(key)) ?? initialValue;
    } catch (error) {
      return initialValue;
    }
  });
```

Features:

- Lazy initial state
- Error handling
- Automatic JSON parsing

5. Implement a custom useIntersectionObserver hook for infinite scrolling

Solution:

```
const useIntersectionObserver = (callback, options = {}) => {
  const targetRef = useRef(null);
  useEffect(() => {
    const observer = new IntersectionObserver(([entry]) => {
      if (entry.isIntersecting) callback();
    }, options);
    observer.observe(targetRef.current);
    return () => observer.disconnect();
  }, []);
```

Features:

- Intersection Observer API
- Cleanup on unmount
- Configurable options

6. Create a custom useMediaQuery hook for responsive design

Solution:

```
const useMediaQuery = (query) => {
  const [matches, setMatches] = useState(window.matchMedia(query).matches);
  useEffect(() => {
    const media = window.matchMedia(query);
    const listener = (e) => setMatches(e.matches);
    media.addListener(listener);
    return () => media.removeListener(listener);
  }, [query]);
```

Features:

- Window matchMedia API
- Event listener cleanup
- Dynamic query support

7. Implement a custom useThrottle hook for rate limiting

Solution:

```
const useThrottle = (value, limit) => {
  const [throttledValue, setThrottledValue] = useState(value);
  const lastRan = useRef(Date.now());
  useEffect(() => {
    if (Date.now() >= lastRan.current + limit) {
      setThrottledValue(value);
      lastRan.current = Date.now();
    }
  }, [value, limit]);
```

Features:

- Time-based throttling
- Reference tracking
- Configurable limit

8. Create a custom useAsync hook for handling async operations

Solution:

```
const useAsync = (asyncFn) => {
  const [state, setState] = useState({ loading: false, error: null, data: null });
  const execute = useCallback(async () => {
    setState({ loading: true, error: null, data: null });
    try {
      const data = await asyncFn();
      setState({ loading: false, error: null, data });
    } catch (error) {
      setState({ loading: false, error, data: null });
    }
  }, [asyncFn]);
```

Features:

- Loading state management
- Error handling
- Data storage

9. Implement a custom useEventListener hook for global event handling

Solution:

```
const useEventListener = (eventName, handler, element = window) => {
  const savedHandler = useRef();
  useEffect(() => {
    savedHandler.current = handler;
  }, [handler]);
  useEffect(() => {
    element.addEventListener(eventName, savedHandler.current);
    return () => element.removeEventListener(eventName, savedHandler.current);
  }, [eventName, element]);
```

Features:

- Event delegation
- Handler memoization
- Cleanup handling

10. Create a custom useClickOutside hook for detecting outside clicks

Solution:

```
const useClickOutside = (ref, callback) => {
  useEffect(() => {
    const handleClick = (event) => {
      if (ref.current && !ref.current.contains(event.target)) {
        callback();
      }
    };
```

```
    document.addEventListener('mousedown', handleClick);
    return () => document.removeEventListener('mousedown', handleClick);
  }, [ref, callback]);
```

Features:

- Ref-based detection
- Event cleanup
- Callback execution

Data Structures and Algorithms

Questions in this section test your understanding of how to work with and manipulate data efficiently.

1. Implement a custom LRU Cache in React with a specified capacity

Solution:

We can implement an LRU Cache using a Map for O(1) access and maintaining order:

```
class LRUCache {
  constructor(capacity) {
    this.cache = new Map();
    this.capacity = capacity;
  }
  get(key) {
    if (!this.cache.has(key)) return -1;
    const value = this.cache.get(key);
    this.cache.delete(key);
    this.cache.set(key, value);
    return value;
  }
  put(key, value) {
    if (this.cache.has(key)) this.cache.delete(key);
    else if (this.cache.size >= this.capacity) {
      this.cache.delete(this.cache.keys().next().value);
    }
    this.cache.set(key, value);
  }}
```

2. Implement a function that finds all pairs of numbers in an array that sum to a target value

Solution:

Using a hash map provides O(n) time complexity:

```
const findPairs = (arr, target) => {
  const seen = new Set();
  const pairs = [];
  arr.forEach(num => {
    const complement = target - num;
    if (seen.has(complement)) pairs.push([num, complement]);
    seen.add(num);
  });
  return pairs;
}
```

3. Implement a sliding window maximum function for a given array and window size

Solution:

Using a deque approach for O(n) time complexity:

```
const maxSlidingWindow = (nums, k) => {
  const result = [];
  const deque = [];
  for (let i = 0; i < nums.length; i++) {
    while (deque.length && nums[deque[deque.length-1]] <= nums[i])
      deque.pop();
```

```
    deque.push(i);
    if (deque[0] <= i - k) deque.shift();
    if (i >= k - 1) result.push(nums[deque[0]]);
  }
  return result;
}
```

4. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time

Solution:

Using two stacks to track minimums:

```
class MinStack {
  constructor() {
    this.stack = [];
    this.minStack = [];
  }
  push(x) {
    this.stack.push(x);
    if (!this.minStack.length || x <= this.minStack[this.minStack.length-1])
      this.minStack.push(x);
  }
  pop() {
    if (this.stack.pop() === this.minStack[this.minStack.length-1])
      this.minStack.pop();
  }
  top() { return this.stack[this.stack.length-1]; }
  getMin() { return this.minStack[this.minStack.length-1]; }
}
```

5. Implement a function to detect if a linked list has a cycle using constant extra space

Solution:

Using Floyd's Cycle-Finding Algorithm (tortoise and hare):

```
const hasCycle = head => {
  let slow = head, fast = head;
  while (fast && fast.next) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow === fast) return true;
  }
  return false;
}
```

6. Implement a throttle function that limits the rate at which a function can fire

Solution:

Implementation with specified delay:

```
const throttle = (func, limit) => {
  let inThrottle;
  return function(...args) {
    if (!inThrottle) {
      func.apply(this, args);
      inThrottle = true;
      setTimeout(() => inThrottle = false, limit);
    }
  }
```

7. Implement a function to flatten a nested array structure

Solution:

Using recursion for deep flattening:

```
const flatten = arr => {
  return arr.reduce((flat, item) =>
    flat.concat(Array.isArray(item) ? flatten(item) : item),
  []);
}
```

8. Implement a debounce function with immediate option

Solution:

Advanced debounce with immediate execution option:

```
const debounce = (func, wait, immediate = false) => {
  let timeout;
  return function(...args) {
    const later = () => {
      timeout = null;
      if (!immediate) func.apply(this, args);
    };
    const callNow = immediate && !timeout;
    clearTimeout(timeout);
    timeout = setTimeout(later, wait);
    if (callNow) func.apply(this, args);
  };
```

9. Implement a deep clone function without using JSON methods

Solution:

Handling objects, arrays, and primitive types:

```
const deepClone = obj => {
  if (obj === null || typeof obj !== 'object') return obj;
  const clone = Array.isArray(obj) ? [] : {};
  for (let key in obj) {
    if (Object.prototype.hasOwnProperty.call(obj, key)) {
      clone[key] = deepClone(obj[key]);
    }
  }
  return clone;
}
```

10. Implement a function to find the longest substring without repeating characters

Solution:

Using sliding window technique:

```
const lengthOfLongestSubstring = s => {
  let max = 0, start = 0;
  const seen = new Map();
  for (let i = 0; i < s.length; i++) {
    if (seen.has(s[i])) start = Math.max(start, seen.get(s[i]) + 1);
    seen.set(s[i], i);
    max = Math.max(max, i - start + 1);
  }
  return max;
}
```

System Design

These questions evaluate your ability to think about the bigger picture, including architecture, scalability, and performance.

1. Design a scalable URL shortener service like bit.ly

Key Requirements:

- Generate unique short URLs
- Redirect to original URL
- High availability
- Low latency

System Design:

- **API Gateway**: Handle incoming requests, rate limiting
- **URL Generation Service**: Create short URLs using base62 encoding or counter-based approach
- **Storage**: NoSQL (like DynamoDB) for URL mappings
- **Cache Layer**: Redis for frequently accessed URLs
- **Analytics Service**: Track clicks and usage patterns

Code Example (URL Generation):

```
const generateShortURL = (counter) => {
  const base62chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
  let shortURL = '';
  while (counter > 0) {
    shortURL = base62chars[counter % 62] + shortURL;
    counter = Math.floor(counter / 62);
  }
  return shortURL;
}
```

2. Design a real-time chat system like Slack

Core Components:

- **WebSocket Server**: Handle real-time connections
- **Message Queue**: RabbitMQ/Kafka for message delivery
- **User Service**: Authentication and user management
- **Chat Service**: Message handling and persistence

Key Considerations:

- Message ordering and delivery guarantees
- Presence management
- Message persistence
- Offline message handling

WebSocket Connection Example:

```
const ws = new WebSocket('wss://chat.example.com');
ws.onmessage = (event) => {
  const message = JSON.parse(event.data);
  updateChatUI(message);
};
ws.send(JSON.stringify({ type: 'message', content: 'Hello!' }));
```

3. Design a social media feed system like Instagram

System Components:

- **Feed Service**: Generate personalized feeds
- **Post Service**: Handle post creation/storage
- **CDN**: Media content delivery
- **Cache Layer**: Redis for feed caching

Feed Generation Approaches:

- Push model: Fan-out on write
- Pull model: Compile on read
- Hybrid approach for optimal performance

Feed Caching Example:

```
const getFeed = async (userId) => {
  const cachedFeed = await redis.get(`feed:${userId}`);
  if (cachedFeed) return JSON.parse(cachedFeed);
  const feed = await generateFeed(userId);
  await redis.setex(`feed:${userId}`, 3600, JSON.stringify(feed));
  return feed;
}
```

4. Design a distributed rate limiter

Requirements:

- Limit requests per user/IP
- Distributed system support
- High availability
- Low latency

Implementation Approaches:

- **Token Bucket**: Flexible rate limiting
- **Sliding Window**: More precise control
- **Redis**: Distributed counter storage

Rate Limiter Implementation:

```
const checkRateLimit = async (key, limit, window) => {
  const current = Date.now();
  const clearBefore = current - window;
  await redis.zremrangebyscore(key, 0, clearBefore);
  const count = await redis.zcard(key);
  return count < limit;
}
```

5. Design a distributed caching system

Key Components:

- **Cache Nodes**: Store data in memory
- **Consistent Hashing**: Data distribution
- **Replication**: Data redundancy
- **Cache Client**: Handle cache operations

Features:

- Eviction policies (LRU/LFU)
- Cache coherence
- Failure handling

Consistent Hashing Example:

```
const getNode = (key, nodes) => {
  const hash = createHash(key);
  const sortedNodes = [...nodes].sort();
  const nodeIndex = binarySearch(sortedNodes, hash);
  return sortedNodes[nodeIndex % sortedNodes.length];
}
```

6. Design a notification service

System Components:

- **Notification Service**: Handle notification logic
- **Queue System**: Message buffering
- **Push Providers**: FCM, APNS integration
- **Template Service**: Message templates

Features:

- Multiple channels (push, email, SMS)
- Rate limiting
- Delivery tracking

Notification Dispatch Example:

```
const sendNotification = async (userId, template, data) => {
  const userChannels = await getUserChannels(userId);
  const message = await renderTemplate(template, data);
  return Promise.all(userChannels.map(channel =>
    dispatch(channel, message)));
}
```

7. Design a job scheduling system

Core Components:

- **Scheduler Service**: Job scheduling logic
- **Worker Pool**: Job execution
- **Queue System**: Job queue management
- **State Store**: Job status tracking

Features:

- Cron-based scheduling
- Retry mechanisms
- Priority queues

Job Scheduler Example:

```
const scheduleJob = async (jobConfig) => {
  const jobId = generateJobId();
  await redis.zadd('scheduled_jobs', jobConfig.timestamp,
    JSON.stringify({ id: jobId, ...jobConfig }));
  return jobId;
}
```

8. Design a distributed search system

System Components:

- **Indexer Service**: Document processing
- **Search Service**: Query processing
- **Data Sharding**: Index distribution
- **Query Coordinator**: Result aggregation

Features:

- Inverted index

- Relevance scoring
- Faceted search

Search Query Example:

```
const search = async (query, filters) => {
  const shards = getRelevantShards(query);
  const results = await Promise.all(shards.map(shard =>
    queryIndex(shard, query, filters)));
  return mergeAndRankResults(results);
}
```

9. Design a distributed logging system

System Components:

- **Log Collector**: Gather logs
- **Stream Processor**: Real-time processing
- **Storage Service**: Log persistence
- **Query Service**: Log analysis

Features:

- Log aggregation
- Real-time processing
- Search capabilities

Log Processing Example:

```
const processLog = async (logEntry) => {
  const enriched = await enrichLogData(logEntry);
  await kafka.produce('logs', enriched);
  if (isErrorLog(enriched)) {
    await alertSystem.notify(enriched);
  }
}
```

10. Design a content delivery network (CDN)

System Components:

- **Edge Servers**: Content caching
- **Origin Servers**: Source content
- **Load Balancer**: Request distribution
- **DNS Service**: Server selection

Features:

- Geographic distribution
- Cache invalidation
- SSL termination

Cache Control Example:

```
const setCachePolicy = (response, ttl) => {
  response.headers.set('Cache-Control', `public, max-age=${ttl}`);
  response.headers.set('Edge-Control', `cache-maxage=${ttl}`);
  return response;
}
```

Coding and Debugging

This section presents practical coding challenges and questions about debugging techniques.

1. Write a custom hook to implement debouncing in React

Solution:

Here's a custom useDebounce hook implementation:

```
const useDebounce = (value, delay) => {
  const [debouncedValue, setDebouncedValue] = useState(value);
  useEffect(() => {
    const timer = setTimeout(() => setDebouncedValue(value), delay);
    return () => clearTimeout(timer);
  }, [value, delay]);
  return debouncedValue;
}
```

Usage:

```
const searchTerm = useDebounce(inputValue, 500);
```

2. Implement a function to flatten a deeply nested array in React

Solution:

```
const flattenArray = (arr) => {
  return arr.reduce((flat, item) => {
    return flat.concat(Array.isArray(item) ? flattenArray(item) : item);
  }, []);
}
```

Example usage:

```
const nested = [1, [2, 3, [4, 5]], 6];
const flat = flattenArray(nested); // [1, 2, 3, 4, 5, 6]
```

3. Create a custom hook for handling infinite scroll in React

Solution:

```
const useInfiniteScroll = (callback) => {
  const observer = useRef();
  const lastElementRef = useCallback(node => {
    if (observer.current) observer.current.disconnect();
    observer.current = new IntersectionObserver(entries => {
      if (entries[0].isIntersecting) callback();
    });
    if (node) observer.current.observe(node);
  }, [callback]);
  return lastElementRef;
}
```

4. Implement a memoized selector function similar to Redux's createSelector

Solution:

```
const createSelector = (selectors, resultFn) => {
  let lastArgs = null;
  let lastResult = null;
```

```
  return (...args) => {
    const selectorResults = selectors.map(fn => fn(...args));
    if (lastArgs && selectorResults.every((val, i) => val === lastArgs[i])) {
      return lastResult;
    }
    lastArgs = selectorResults;
    lastResult = resultFn(...selectorResults);
    return lastResult;
  };
}
```

5. Create a custom hook for handling form validation in React

Solution:

```
const useFormValidation = (initialState, validate) => {
  const [values, setValues] = useState(initialState);
  const [errors, setErrors] = useState({});
  const handleChange = (event) => {
    const { name, value } = event.target;
    setValues(values => ({ ...values, [name]: value }));
    setErrors(validate({ ...values, [name]: value }));
  };
  return { values, errors, handleChange };
}
```

6. Implement a function to deep clone a React component's props

Solution:

```
const deepCloneProps = (props) => {
  if (typeof props !== 'object' || props === null) return props;
  const clone = Array.isArray(props) ? [] : {};
  for (let key in props) {
    clone[key] = deepCloneProps(props[key]);
  }
  return clone;
}
```

Note: This handles nested objects and arrays but not functions or React elements.

7. Create a custom hook for managing async state with loading and error handling

Solution:

```
const useAsync = (asyncFn) => {
  const [state, setState] = useState({ data: null, loading: true, error: null });
  useEffect(() => {
    asyncFn()
      .then(data => setState({ data, loading: false, error: null }))
      .catch(error => setState({ data: null, loading: false, error }));
  }, [asyncFn]);
  return state;
}
```

8. Implement a function to check if two React elements are equal

Solution:

```
const areElementsEqual = (elem1, elem2) => {
  return elem1?.type === elem2?.type &&
    JSON.stringify(elem1?.props) === JSON.stringify(elem2?.props) &&
    elem1?.key === elem2?.key;
}
```

Note: This is a simplified version. Real implementations should handle more edge cases.

9. Create a custom hook for managing local storage with React

Solution:

```
const useLocalStorage = (key, initialValue) => {
  const [storedValue, setStoredValue] = useState(() => {
    try {
      return JSON.parse(localStorage.getItem(key)) ?? initialValue;
    } catch (error) {
      return initialValue;
    }
  });
  useEffect(() => localStorage.setItem(key, JSON.stringify(storedValue)), [key, storedValue]);
  return [storedValue, setStoredValue];
}
```

10. Implement a function to create a throttled version of a React event handler

Solution:

```
const throttle = (func, limit) => {
  let inThrottle;
  return function(...args) {
    if (!inThrottle) {
      func.apply(this, args);
      inThrottle = true;
      setTimeout(() => inThrottle = false, limit);
    }
  };
}
```

Usage:

```
const throttledHandler = throttle(handleScroll, 100);
```