Gas Tracker Dashboard - Interview Guide 📚

Table of Contents

1. Project Presentation Structure

Opening (30 seconds)

"I developed a Gas Tracker Dashboard that monitors blockchain gas prices in real-time. This project helps traders and developers optimize their transaction costs on networks like Ethereum through real-time monitoring, price predictions, and transaction simulation."

Technical Overview (1 minute)

```
Tech Stack:
- Frontend: React 18 + TypeScript
- Build Tool: Vite
- State Management: Zustand
- Real-time Updates: WebSocket
- Blockchain: Ethereum providers + Uniswap SDK
```

Key Features (1 minute)

1. Real-time Gas Price Monitoring
2. Multi-Chain Support
3. Transaction Simulation
4. Interactive Charts
5. Alert System

2. Technical Deep Dive

Architecture Highlights

1. **Frontend Architecture**

   - Component-based structure
   - Custom hooks for business logic
   - Centralized state management
   - TypeScript for type safety

2. **Real-time Data Flow**

   - WebSocket connection management
   - Data normalization
   - State updates optimization
   - Error handling and reconnection logic

3. **Performance Optimizations**

   - Memoization strategies
   - Virtual scrolling
   - Lazy loading
   - Bundle size optimization

3. Live Demo Guide

Demo Preparation

1. Local environment setup
2. Test data ready
3. Common scenarios prepared
4. Backup plan for technical issues

Demo Flow

1. **Start with Homepage**

   - Show real-time price updates
   - Explain visual indicators
   - Demonstrate responsiveness

2. **Feature Walkthrough**

   - Gas price monitoring
   - Chain selection
   - Transaction simulation
   - Alert configuration

3. **Technical Showcase**

   - Show code organization
   - Demonstrate error handling
   - Display performance metrics

4. Common Interview Questions

Architecture Questions

Q: "Why did you choose Zustand over Redux?" A: "Zustand offers:

- Simpler boilerplate
- Built-in TypeScript support
- Smaller bundle size
- Perfect for our scale"

Q: "Why WebSocket instead of REST API?" A: "WebSocket provides:

- Real-time updates
- Lower latency
- Reduced server load
- Better user experience"

Technical Implementation

Q: "How do you handle WebSocket failures?" A: "Implemented:

- Automatic reconnection
- Exponential backoff
- Fallback to HTTP polling
- User notifications"

Q: "Explain your performance optimization strategy" A: "Used:

- Code splitting
- Lazy loading
- Memoization
- Virtual scrolling for large datasets"

5. Technical Decision Explanations

State Management

```
// Example of Zustand store structure
interface GasStore {
  prices: GasPrices;
  updatePrices: (newPrices: GasPrices) => void;
  selectedChain: Chain;
  setChain: (chain: Chain) => void;
}
```

Real-time Updates

```
// WebSocket implementation highlight
const useWebSocket = () => {
  // Connection management
  // Data handling
  // Error recovery
};
```

Performance Solutions

```
// Example of optimization
const MemoizedChart = React.memo(({ data }) => {
  // Efficient rendering logic
});
```

Key Talking Points

Technical Excellence

1. Type safety with TypeScript
2. Modern React practices
3. Performance optimization

4. Clean code architecture

Problem Solving

1. Real-time data challenges
2. Cross-chain compatibility
3. User experience optimization
4. Error handling strategies

Best Practices

1. Code organization
2. Testing strategy
3. Documentation
4. Performance monitoring

Interview Tips

DO's

- Start with high-level overview
- Use specific examples
- Show enthusiasm
- Connect features to user benefits
- Be prepared for deep technical questions

DON'Ts

- Don't memorize scripts
- Avoid over-technical jargon
- Don't hide knowledge gaps
- Don't rush through explanations

Conclusion

Remember to:

1. Stay confident but humble
2. Show technical depth
3. Focus on real solutions
4. Maintain clear communication
5. Demonstrate passion for the project

Note: This guide can be converted to PDF for better readability and portability.