USE CASE

Procurement Platform Interface Modernization from JSF to React

CLIENT PROFILE

The HRS eRFP platform, also known as the HRS procurement or sourcing platform, is a comprehensive solution developed by HRS Group to streamline corporate hotel procurement processes. This platform enables efficient communication between corporate clients and hotel suppliers, facilitating tasks such as submitting bids, managing negotiations, and selecting suitable accommodations.

INITIAL CHALLENGE

The customer-facing application was built using legacy JSF framework, which resulted in:

- Limited flexibility in UI/UX design
- Performance bottlenecks
- Poor maintainability
- Difficulty in implementing modern features

PERFORMANCE ISSUES

Dense data rendering issues

- The interface shows a large grid with multiple columns and complex data
- Each row contains numerous data points, calculations, and icons

UI/UX LIMITATIONS

Information Overload

- Multiple metrics, percentages, and values crammed into single rows
- Small, hard-to-read icons and numbers
- Complex tooltip system (as shown in Image 2) that overlays critical information

Poor Information Hierarchy

- All data points appear with equal visual weight
- Critical business metrics blend with secondary information
- Difficult to quickly scan and find important information
- Status indicators (green/red dots) are small and easy to miss

INTERACTION DESIGN ISSUES

- Multiple nested levels of information
- Complex bookmark/flag system
- Many columns requiring horizontal scrolling
- Small click targets for interactive elements
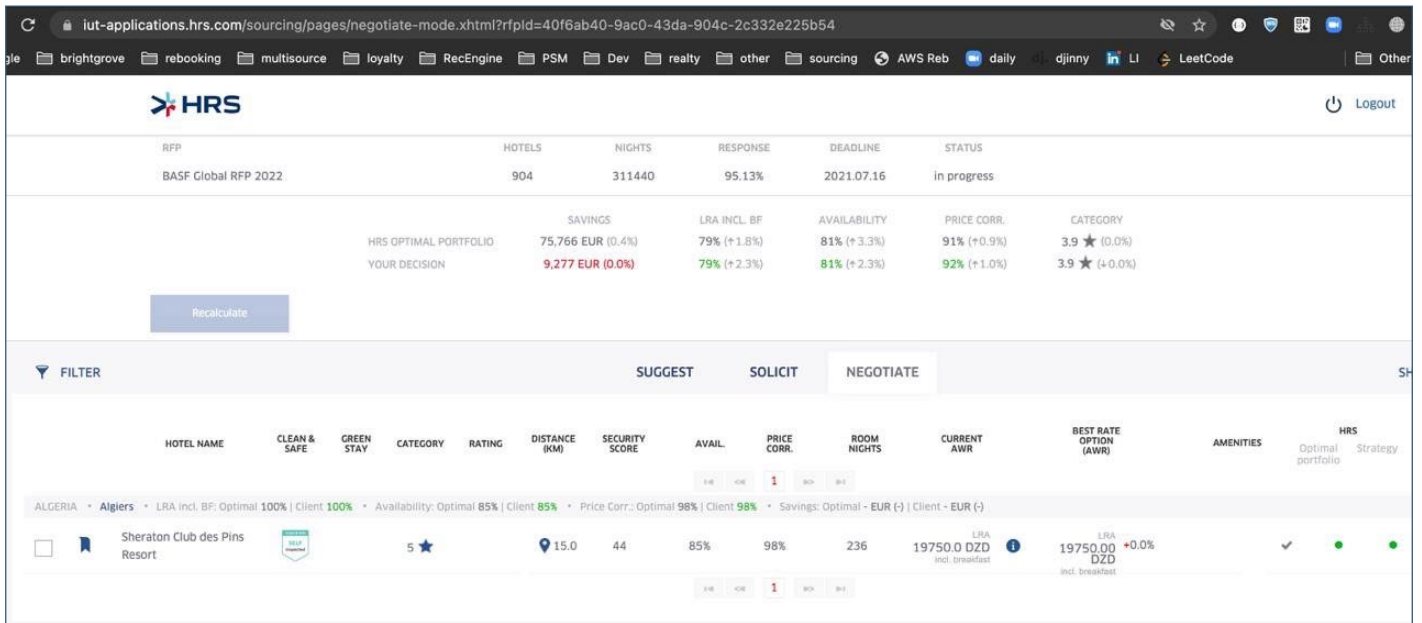
TECHNICAL LIMITATIONS

Monolithic Architecture Impact

- Each data update likely required full row or table refresh
- Synchronization between different data points was probably sluggish
- Changes in filters or sorting would trigger heavy server requests
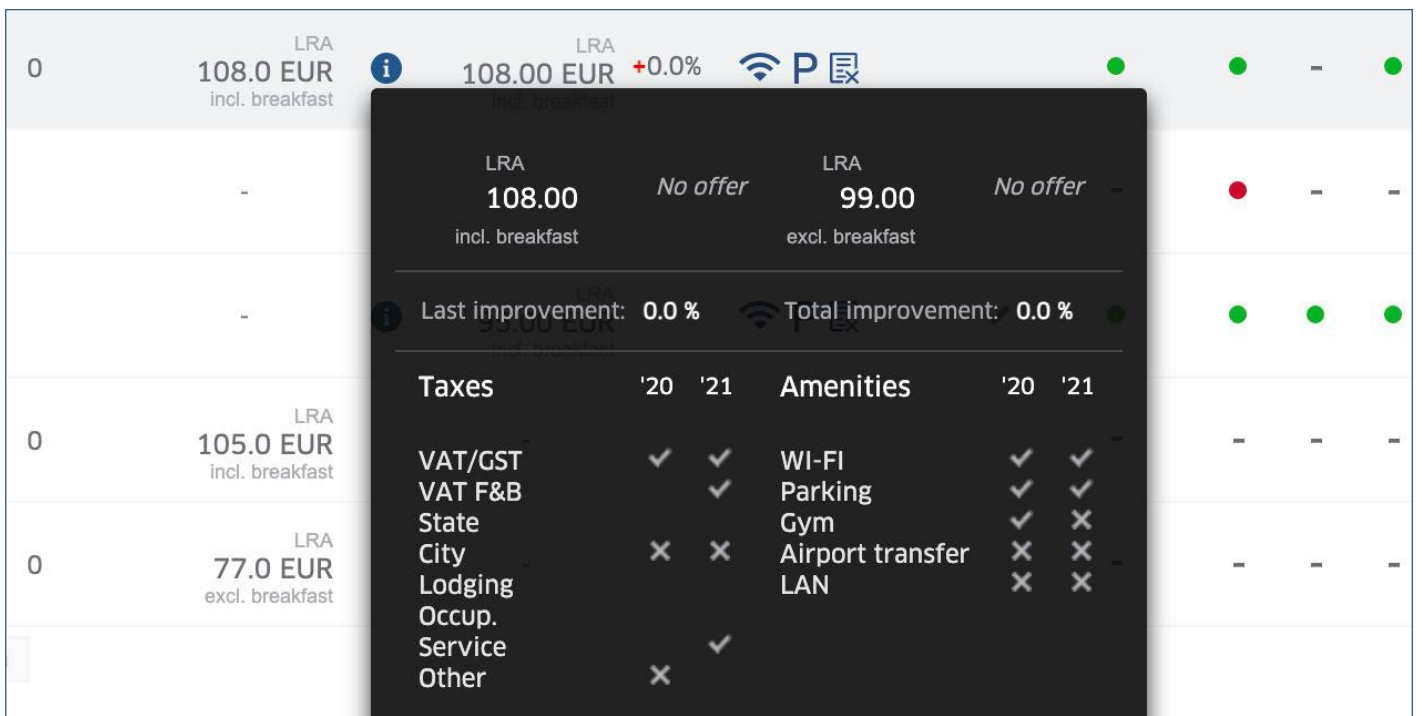
State Management Issues

- Complex state with multiple interdependent values
- Many calculated fields that needed constant updates
- Synchronization issues between different table sections

Browser Performance

- Heavy DOM manipulation with many elements
- Complex CSS calculations for layout and styling
- Resource-intensive rendering of large data tables
- Poor handling of real-time updates

FRONT-END MODERNIZATION SOLUTION

Core Technology Stack

- Migrated from JSF to React 17 (later upgraded to React 18)
- Implemented Redux for state management
- Integrated Redux-Saga for side-effect handling
- Utilized Next.js for server-side rendering capabilities

Architecture Improvements

- Component-Based Structure
- Developed reusable component library
- Implemented micro-frontend architecture for gradual migration
- Created feature flags system for controlled rollout
- Performance Optimizations
- Implemented code-splitting
- Added lazy loading for improved initial page loads
- Utilized React.memo for component memoization
- Applied useMemo for computation optimization
- Initial data rendering optimization: achieved ~80ms per row load time
- SEO Enhancements
- Server-side rendering for critical pages
- Dynamic meta tags implementation
- Structured data integration

TECHNICAL IMPLEMENTATION DETAILS
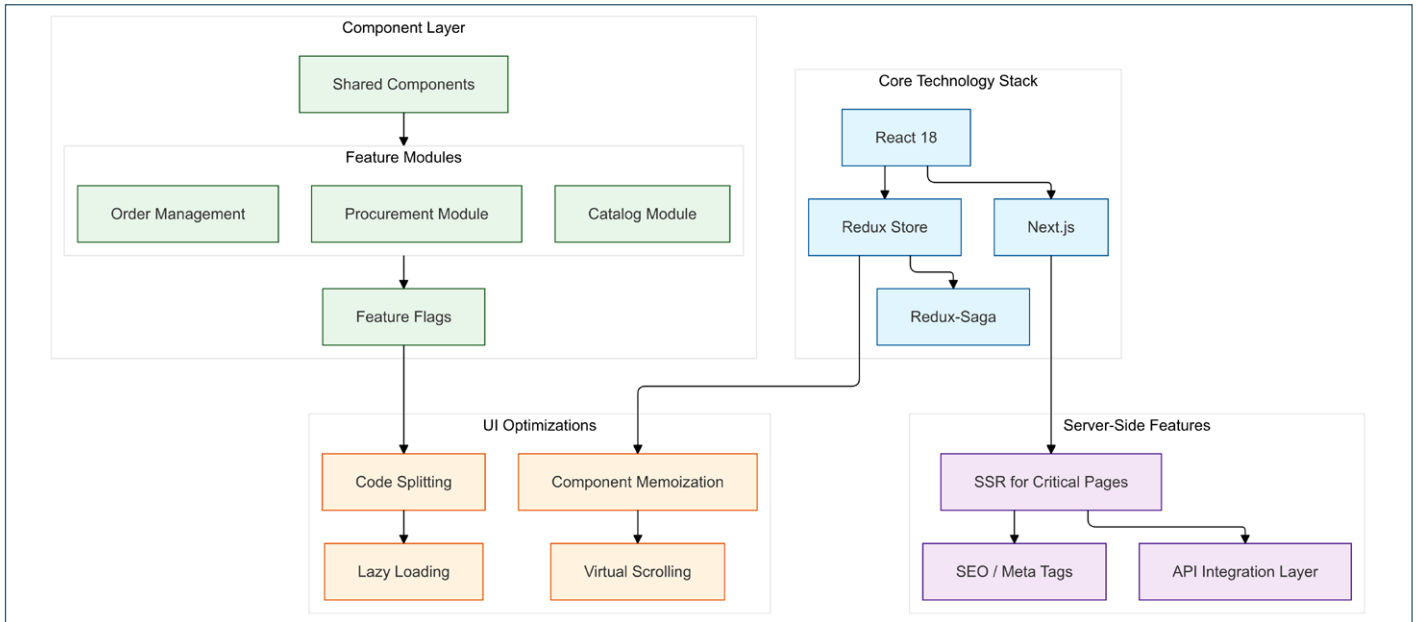
State Management

- Centralized Redux store
- Saga middleware for complex operations
- Optimized state updates for performance

Component Architecture

- Atomic design principles
- Shared component library
- Modular styling system

Loading Optimizations

- Dynamic imports
- Route-based code splitting
- Asset preloading strategies

IMPLEMENTATION APPROACH & MIGRATION STRATEGY

Gradual Transition

- Micro-frontend architecture allowing incremental updates
- Feature flags for controlling new React component deployment
- Parallel operation of old and new systems during transition

Development Timeline

- Core React Application Development: 6 months
- Incremental Feature Migration: 12 months
- Performance Optimization: 3 months

MEASURABLE OUTCOMES

The platform upgrade delivered substantial improvements in both performance and development efficiency. These measurable outcomes demonstrate the significant impact of our strategic technology investments.

Performance Improvements

- 50% reduction in initial page load time
- Successful handling of 300% increased data load
- Optimized rendering performance (80ms per row)

Development Efficiency

- 40% faster time-to-market for new features
- 50% reduction in feature integration time
- Improved code maintainability through component reuse

This modernization effort successfully transformed the user interface while maintaining system stability and improving overall performance metrics.

Ready to take the next step? Reach us today at info@brightgrove.com