Frontend Component Examples

Custom Hooks

usePatentSearch Hook

```typescript
// hooks/usePatentSearch.ts
import { useState } from 'react';
import { useMutation } from '@tanstack/react-query';
import { searchAPI } from '../services/api';

interface SearchFilters {
  dateRange?: {
    start: string;
    end: string;
  };
  classifications?: string[];
  assignees?: string[];
  patentStatus?: string;
  country?: string;
}

interface SearchParams {
  query: string;
  filters?: SearchFilters;
  searchType?: 'basic' | 'semantic' | 'advanced';
}

export const usePatentSearch = () => {
  const [searchHistory, setSearchHistory] = useState<SearchParams[]>([]);

  const searchMutation = useMutation({
    mutationFn: async (params: SearchParams) => {
      switch (params.searchType) {
        case 'semantic':
          return searchAPI.semanticSearch(params);
        case 'advanced':
          return searchAPI.advancedSearch(params);
        default:
          return searchAPI.searchPatents(params);
      }
    },
    onSuccess: (data, variables) => {
      // Add to search history
      setSearchHistory(prev => [variables, ...prev.slice(0, 9)]);
    },
  });

  const clearHistory = () => setSearchHistory([]);

  return {
    search: searchMutation.mutate,
    searchAsync: searchMutation.mutateAsync,
    isSearching: searchMutation.isPending,
    searchResults: searchMutation.data,
    searchError: searchMutation.error,
    searchHistory,
    clearHistory,
  };
};
```

useBatchProgress Hook

```
// hooks/useBatchProgress.ts
import { useEffect, useState } from 'react';
import { useQuery } from '@tanstack/react-query';
import { io, Socket } from 'socket.io-client';
import { batchAPI } from '../services/api';

interface BatchProgress {
  jobId: string;
  progress: number;
  status: string;
  completedItems: number;
  totalItems: number;
  errors?: string[];
}

export const useBatchProgress = (batchJobId: string) => {
  const [realTimeProgress, setRealTimeProgress] = useState<BatchProgress | null>(null);
  const [socket, setSocket] = useState<Socket | null>(null);

  // Fetch initial batch job data
  const { data: batchJob, refetch } = useQuery({
    queryKey: ['batchJob', batchJobId],
    queryFn: () => batchAPI.getBatchJob(batchJobId),
    enabled: !!batchJobId,
    refetchInterval: (data) => {
      // Stop polling when job is complete
      return data?.status === 'completed' || data?.status === 'failed' ? false : 2000;
    },
  });

  // Set up WebSocket connection for real-time updates
  useEffect(() => {
    if (!batchJobId) return;

    const newSocket = io(process.env.REACT_APP_API_URL || 'http://localhost:5000');
    setSocket(newSocket);

    newSocket.emit('join-batch-room', batchJobId);

    newSocket.on('batch-progress', (data: BatchProgress) => {
      if (data.jobId === batchJobId) {
        setRealTimeProgress(data);
        refetch(); // Refresh the main data
      }
    });

    return () => {
      newSocket.disconnect();
    };
  }, [batchJobId, refetch]);

  // Combine real-time data with fetched data
  const currentProgress = realTimeProgress || {
    jobId: batchJobId,
    progress: batchJob?.progress || 0,
    status: batchJob?.status || 'unknown',
    completedItems: batchJob?.completedItems || 0,
    totalItems: batchJob?.totalItems || 0,
    errors: batchJob?.errorLog || [],
  };

  return {
```

```
    batchJob,
    progress: currentProgress,
    isConnected: socket?.connected || false,
  };
};
```

Advanced Components

PatentCard Component

```tsx
// components/patents/PatentCard.tsx
import React from 'react';
import {
  Card,
  CardContent,
  CardActions,
  Typography,
  Chip,
  Button,
  Box,
  IconButton,
  Tooltip,
  Collapse,
} from '@mui/material';
import {
  ExpandMore as ExpandMoreIcon,
  Bookmark as BookmarkIcon,
  Share as ShareIcon,
  Analytics as AnalyticsIcon,
} from '@mui/icons-material';

interface Patent {
  patentNumber: string;
  title: string;
  abstract: string;
  assignee: string;
  inventors: string[];
  filingDate: string;
  grantDate?: string;
  classifications: string[];
  relevanceScore?: number;
}

interface PatentCardProps {
  patent: Patent;
  onAnalyze?: (patent: Patent) => void;
  onBookmark?: (patent: Patent) => void;
  onShare?: (patent: Patent) => void;
  showRelevanceScore?: boolean;
  compact?: boolean;
}

export const PatentCard: React.FC<PatentCardProps> = ({
  patent,
  onAnalyze,
  onBookmark,
  onShare,
  showRelevanceScore = false,
  compact = false,
}) => {
  const [expanded, setExpanded] = React.useState(false);

  const handleExpandClick = () => {
    setExpanded(!expanded);
  };

  const formatDate = (dateString: string) => {
    return new Date(dateString).toLocaleDateString();
  };

  return (
    <Card sx={{ mb: 2, '&:hover': { boxShadow: 4 } }}>
```

```
      <CardContent>
        {/* Header */}
        <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'flex-
start', mb: 2 }}>
          <Box sx={{ flex: 1 }}>
            <Typography variant="h6" component="h3" gutterBottom>
              {patent.title}
            </Typography>
            <Typography variant="subtitle2" color="primary" gutterBottom>
              {patent.patentNumber}
            </Typography>
          </Box>

          {showRelevanceScore && patent.relevanceScore && (
            <Chip
              label={`${Math.round(patent.relevanceScore * 100)}% match`}
              color="primary"
              size="small"
            />
          )}
        </Box>

        {/* Patent Details */}
        <Box sx={{ mb: 2 }}>
          <Typography variant="body2" color="text.secondary" gutterBottom>
            <strong>Assignee:</strong> {patent.assignee}
          </Typography>
          <Typography variant="body2" color="text.secondary" gutterBottom>
            <strong>Inventors:</strong> {patent.inventors.join(', ')}
          </Typography>
          <Typography variant="body2" color="text.secondary" gutterBottom>
            <strong>Filing Date:</strong> {formatDate(patent.filingDate)}
            {patent.grantDate && ` • Grant Date: ${formatDate(patent.grantDate)}`}
          </Typography>
        </Box>

        {/* Classifications */}
        <Box sx={{ mb: 2 }}>
          {patent.classifications.slice(0, 3).map((classification) => (
            <Chip
              key={classification}
              label={classification}
              size="small"
              variant="outlined"
              sx={{ mr: 1, mb: 1 }}
            />
          ))}
          {patent.classifications.length > 3 && (
            <Chip
              label={`+${patent.classifications.length - 3} more`}
              size="small"
              variant="outlined"
              sx={{ mr: 1, mb: 1 }}
            />
          )}
        </Box>

        {/* Abstract (expandable) */}
        {!compact && (
          <>
            <Typography variant="body2" color="text.secondary">
              {expanded ? patent.abstract : `${patent.abstract.substring(0, 200)}...`}
            </Typography>
```

```
          <Collapse in={expanded} timeout="auto" unmountOnExit>
            <Box sx={{ mt: 2 }}>
              {/* Additional details when expanded */}
              <Typography variant="subtitle2" gutterBottom>
                Full Classifications:
              </Typography>
              <Box sx={{ mb: 2 }}>
                {patent.classifications.map((classification) => (
                  <Chip
                    key={classification}
                    label={classification}
                    size="small"
                    variant="outlined"
                    sx={{ mr: 1, mb: 1 }}
                  />
                ))}
              </Box>
            </Box>
          </Collapse>
        </>
      )}
    </CardContent>

    <CardActions sx={{ justifyContent: 'space-between' }}>
      <Box>
        {onAnalyze && (
          <Button
            startIcon={<AnalyticsIcon />}
            onClick={() => onAnalyze(patent)}
            size="small"
          >
            Analyze
          </Button>
        )}

        {!compact && (
          <IconButton
            onClick={handleExpandClick}
            aria-expanded={expanded}
            aria-label="show more"
          >
            <ExpandMoreIcon
              sx={{
                transform: expanded ? 'rotate(180deg)' : 'rotate(0deg)',
                transition: 'transform 0.3s',
              }}
            />
          </IconButton>
        )}
      </Box>

      <Box>
        {onBookmark && (
          <Tooltip title="Bookmark">
            <IconButton onClick={() => onBookmark(patent)} size="small">
              <BookmarkIcon />
            </IconButton>
          </Tooltip>
        )}

        {onShare && (
          <Tooltip title="Share">
```

```
              <IconButton onClick={() => onShare(patent)} size="small">
                <ShareIcon />
              </IconButton>
            </Tooltip>
          )}
        </Box>
      </CardActions>
    </Card>
  );
};
```

AnalysisResultsVisualization Component

```tsx
// components/analysis/AnalysisResultsVisualization.tsx
import React from 'react';
import {
  Box,
  Card,
  CardContent,
  Typography,
  Grid,
  LinearProgress,
  Chip,
  Accordion,
  AccordionSummary,
  AccordionDetails,
  List,
  ListItem,
  ListItemText,
  ListItemIcon,
} from '@mui/material';
import {
  ExpandMore as ExpandMoreIcon,
  CheckCircle as CheckCircleIcon,
  Warning as WarningIcon,
  Error as ErrorIcon,
  TrendingUp as TrendingUpIcon,
} from '@mui/icons-material';
import {
  RadarChart,
  PolarGrid,
  PolarAngleAxis,
  PolarRadiusAxis,
  Radar,
  ResponsiveContainer,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  PieChart,
  Pie,
  Cell,
} from 'recharts';

interface AgentResult {
  agentName: string;
  confidence: number;
  result: any;
  timestamp: number;
}

interface AnalysisResults {
  agentResults: {
    prior_art?: AgentResult;
    claims?: AgentResult;
    market?: AgentResult;
    legal?: AgentResult;
  };
  summary: string;
  confidenceScore: number;
}

interface AnalysisResultsVisualizationProps {
```

```
  results: AnalysisResults;
}

export const AnalysisResultsVisualization: React.FC<AnalysisResultsVisualizationProps>
= ({
  results,
}) => {
  // Prepare data for radar chart
  const radarData = Object.entries(results.agentResults).map(([key, agent]) => ({
    agent: key.replace('_', ' ').toUpperCase(),
    confidence: Math.round(agent.confidence * 100),
  }));

  // Prepare data for confidence distribution
  const confidenceData = [
    { name: 'High (80-100%)', value: 0, color: '#4caf50' },
    { name: 'Medium (60-79%)', value: 0, color: '#ff9800' },
    { name: 'Low (0-59%)', value: 0, color: '#f44336' },
  ];

  Object.values(results.agentResults).forEach((agent) => {
    const confidence = agent.confidence * 100;
    if (confidence >= 80) confidenceData[0].value++;
    else if (confidence >= 60) confidenceData[1].value++;
    else confidenceData[2].value++;
  });

  const getScoreColor = (score: number) => {
    if (score >= 80) return 'success';
    if (score >= 60) return 'warning';
    return 'error';
  };

  const getScoreIcon = (score: number) => {
    if (score >= 80) return <CheckCircleIcon color="success" />;
    if (score >= 60) return <WarningIcon color="warning" />;
    return <ErrorIcon color="error" />;
  };

  return (
    <Box>
      {/* Overall Summary */}
      <Card sx={{ mb: 3 }}>
        <CardContent>
          <Typography variant="h5" gutterBottom>
            Analysis Summary
          </Typography>
          <Box sx={{ display: 'flex', alignItems: 'center', mb: 2 }}>
            <Typography variant="h3" color="primary" sx={{ mr: 2 }}>
              {Math.round(results.confidenceScore * 100)}%
            </Typography>
            <Box sx={{ flex: 1 }}>
              <Typography variant="body2" color="text.secondary" gutterBottom>
                Overall Confidence Score
              </Typography>
              <LinearProgress
                variant="determinate"
                value={results.confidenceScore * 100}
                color={getScoreColor(results.confidenceScore * 100) as any}
                sx={{ height: 8, borderRadius: 4 }}
              />
            </Box>
          </Box>
```

```
            <Typography variant="body1">
              {results.summary}
            </Typography>
          </CardContent>
        </Card>

        {/* Visualizations */}
        <Grid container spacing={3}>
          {/* Agent Confidence Radar Chart */}
          <Grid item xs={12} md={6}>
            <Card>
              <CardContent>
                <Typography variant="h6" gutterBottom>
                  Agent Confidence Levels
                </Typography>
                <ResponsiveContainer width="100%" height={300}>
                  <RadarChart data={radarData}>
                    <PolarGrid />
                    <PolarAngleAxis dataKey="agent" />
                    <PolarRadiusAxis angle={90} domain={[0, 100]} />
                    <Radar
                      name="Confidence"
                      dataKey="confidence"
                      stroke="#1976d2"
                      fill="#1976d2"
                      fillOpacity={0.3}
                    />
                  </RadarChart>
                </ResponsiveContainer>
              </CardContent>
            </Card>
          </Grid>

          {/* Confidence Distribution */}
          <Grid item xs={12} md={6}>
            <Card>
              <CardContent>
                <Typography variant="h6" gutterBottom>
                  Confidence Distribution
                </Typography>
                <ResponsiveContainer width="100%" height={300}>
                  <PieChart>
                    <Pie
                      data={confidenceData}
                      cx="50%"
                      cy="50%"
                      outerRadius={80}
                      dataKey="value"
                      label={({ name, value }) => `${name}: ${value}`}
                    >
                      {confidenceData.map((entry, index) => (
                        <Cell key={`cell-${index}`} fill={entry.color} />
                      ))}
                    </Pie>
                    <Tooltip />
                  </PieChart>
                </ResponsiveContainer>
              </CardContent>
            </Card>
          </Grid>
        </Grid>

        {/* Detailed Agent Results */}
```

```
      <Box sx={{ mt: 3 }}>
        <Typography variant="h5" gutterBottom>
          Detailed Agent Results
        </Typography>

        {Object.entries(results.agentResults).map(([agentName, agentResult]) => (
          <Accordion key={agentName} sx={{ mb: 1 }}>
            <AccordionSummary expandIcon={<ExpandMoreIcon />}>
              <Box sx={{ display: 'flex', alignItems: 'center', width: '100%' }}>
                <Typography variant="h6" sx={{ flex: 1 }}>
                  {agentName.replace('_', ' ').toUpperCase()} Agent
                </Typography>
                <Chip
                  icon={getScoreIcon(agentResult.confidence * 100)}
                  label={`${Math.round(agentResult.confidence * 100)}% confidence`}
                  color={getScoreColor(agentResult.confidence * 100) as any}
                  sx={{ mr: 2 }}
                />
              </Box>
            </AccordionSummary>
            <AccordionDetails>
              {/* Render agent-specific results */}
              {agentName === 'prior_art' && (
                <PriorArtResults result={agentResult.result} />
              )}
              {agentName === 'claims' && (
                <ClaimsResults result={agentResult.result} />
              )}
              {agentName === 'market' && (
                <MarketResults result={agentResult.result} />
              )}
              {agentName === 'legal' && (
                <LegalResults result={agentResult.result} />
              )}
            </AccordionDetails>
          </Accordion>
        ))}
      </Box>
    </Box>
  );
};

// Agent-specific result components
const PriorArtResults: React.FC<{ result: any }> = ({ result }) => (
  <Box>
    <Typography variant="subtitle1" gutterBottom>
      Novelty Score: {result.noveltyScore}%
    </Typography>
    <Typography variant="subtitle2" gutterBottom>
      Similar Patents Found:
    </Typography>
    <List>
      {result.similarPatents?.map((patent: any, index: number) => (
        <ListItem key={index}>
          <ListItemText
            primary={patent.patentNumber}
            secondary={`Similarity: ${Math.round(patent.similarity * 100)}%`}
          />
        </ListItem>
      ))}
    </List>
  </Box>
);
```

```
const ClaimsResults: React.FC<{ result: any }> = ({ result }) => (
  <Box>
    <Typography variant="subtitle2" gutterBottom>
      Claim Strength Analysis:
    </Typography>
    <Box sx={{ mb: 2 }}>
      <Typography variant="body2">
        Independent Claims: {Math.round(result.claimStrength?.independent * 100)}%
      </Typography>
      <LinearProgress
        variant="determinate"
        value={result.claimStrength?.independent * 100}
        sx={{ mb: 1 }}
      />
      <Typography variant="body2">
        Dependent Claims: {Math.round(result.claimStrength?.dependent * 100)}%
      </Typography>
      <LinearProgress
        variant="determinate"
        value={result.claimStrength?.dependent * 100}
      />
    </Box>
    <Typography variant="subtitle2" gutterBottom>
      Recommendations:
    </Typography>
    <List>
      {result.recommendations?.map((rec: string, index: number) => (
        <ListItem key={index}>
          <ListItemIcon>
            <TrendingUpIcon color="primary" />
          </ListItemIcon>
          <ListItemText primary={rec} />
        </ListItem>
      ))}
    </List>
  </Box>
);

const MarketResults: React.FC<{ result: any }> = ({ result }) => (
  <Box>
    <Typography variant="subtitle1" gutterBottom>
      Competitor Activity: {result.competitorActivity}
    </Typography>
    <Typography variant="subtitle2" gutterBottom>
      Market Trends:
    </Typography>
    <List>
      {result.marketTrends?.map((trend: string, index: number) => (
        <ListItem key={index}>
          <ListItemText primary={trend} />
        </ListItem>
      ))}
    </List>
    <Typography variant="subtitle2" gutterBottom>
      Opportunities:
    </Typography>
    <List>
      {result.opportunities?.map((opportunity: string, index: number) => (
        <ListItem key={index}>
          <ListItemIcon>
            <TrendingUpIcon color="success" />
          </ListItemIcon>
```

```
          <ListItemText primary={opportunity} />
        </ListItem>
      ))}
    </List>
  </Box>
);

const LegalResults: React.FC<{ result: any }> = ({ result }) => (
  <Box>
    <Typography variant="subtitle1" gutterBottom>
      Patentability Score: {result.patentabilityScore}%
    </Typography>
    <Typography variant="subtitle2" gutterBottom>
      Identified Risks:
    </Typography>
    <List>
      {result.risks?.map((risk: string, index: number) => (
        <ListItem key={index}>
          <ListItemIcon>
            <WarningIcon color="warning" />
          </ListItemIcon>
          <ListItemText primary={risk} />
        </ListItem>
      ))}
    </List>
    <Typography variant="subtitle2" gutterBottom>
      Legal Recommendations:
    </Typography>
    <List>
      {result.recommendations?.map((rec: string, index: number) => (
        <ListItem key={index}>
          <ListItemIcon>
            <CheckCircleIcon color="success" />
          </ListItemIcon>
          <ListItemText primary={rec} />
        </ListItem>
      ))}
    </List>
  </Box>
);
```

State Management Examples

React Query Configuration

```typescript
// utils/queryClient.ts
import { QueryClient } from '@tanstack/react-query';

export const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      retry: (failureCount, error: any) => {
        // Don't retry on 4xx errors
        if (error?.response?.status >= 400 && error?.response?.status < 500) {
          return false;
        }
        return failureCount < 3;
      },
      staleTime: 5 * 60 * 1000, // 5 minutes
      cacheTime: 10 * 60 * 1000, // 10 minutes
      refetchOnWindowFocus: false,
    },
    mutations: {
      retry: 1,
    },
  },
});

// Query key factory
export const queryKeys = {
  all: ['patent-analysis'] as const,
  projects: () => [...queryKeys.all, 'projects'] as const,
  project: (id: string) => [...queryKeys.projects(), id] as const,
  analyses: () => [...queryKeys.all, 'analyses'] as const,
  analysis: (id: string) => [...queryKeys.analyses(), id] as const,
  search: (params: any) => [...queryKeys.all, 'search', params] as const,
  batchJobs: () => [...queryKeys.all, 'batch-jobs'] as const,
  batchJob: (id: string) => [...queryKeys.batchJobs(), id] as const,
  dashboard: (persona: string) => [...queryKeys.all, 'dashboard', persona] as const,
};
```

Context for Global State

```tsx
// contexts/AppContext.tsx
import React, { createContext, useContext, useReducer, ReactNode } from 'react';

interface AppState {
  sidebarOpen: boolean;
  theme: 'light' | 'dark';
  notifications: Notification[];
  currentProject: string | null;
  searchFilters: any;
}

interface Notification {
  id: string;
  type: 'success' | 'error' | 'warning' | 'info';
  message: string;
  timestamp: number;
}

type AppAction =
  | { type: 'TOGGLE_SIDEBAR' }
  | { type: 'SET_THEME'; payload: 'light' | 'dark' }
  | { type: 'ADD_NOTIFICATION'; payload: Omit<Notification, 'id' | 'timestamp'> }
  | { type: 'REMOVE_NOTIFICATION'; payload: string }
  | { type: 'SET_CURRENT_PROJECT'; payload: string | null }
  | { type: 'UPDATE_SEARCH_FILTERS'; payload: any };

const initialState: AppState = {
  sidebarOpen: true,
  theme: 'light',
  notifications: [],
  currentProject: null,
  searchFilters: {},
};

const appReducer = (state: AppState, action: AppAction): AppState => {
  switch (action.type) {
    case 'TOGGLE_SIDEBAR':
      return { ...state, sidebarOpen: !state.sidebarOpen };

    case 'SET_THEME':
      return { ...state, theme: action.payload };

    case 'ADD_NOTIFICATION':
      return {
        ...state,
        notifications: [
          ...state.notifications,
          {
            ...action.payload,
            id: Date.now().toString(),
            timestamp: Date.now(),
          },
        ],
      };

    case 'REMOVE_NOTIFICATION':
      return {
        ...state,
        notifications: state.notifications.filter(n => n.id !== action.payload),
      };

    case 'SET_CURRENT_PROJECT':
```

```
      return { ...state, currentProject: action.payload };

    case 'UPDATE_SEARCH_FILTERS':
      return { ...state, searchFilters: { ...state.searchFilters, ...action.payload } }
;

    default:
      return state;
  }
};

const AppContext = createContext<{
  state: AppState;
  dispatch: React.Dispatch<AppAction>;
} | null>(null);

export const AppProvider: React.FC<{ children: ReactNode }> = ({ children }) => {
  const [state, dispatch] = useReducer(appReducer, initialState);

  return (
    <AppContext.Provider value={{ state, dispatch }}>
      {children}
    </AppContext.Provider>
  );
};

export const useApp = () => {
  const context = useContext(AppContext);
  if (!context) {
    throw new Error('useApp must be used within an AppProvider');
  }
  return context;
};

// Helper hooks
export const useNotifications = () => {
  const { state, dispatch } = useApp();

  const addNotification = (notification: Omit<Notification, 'id' | 'timestamp'>) => {
    dispatch({ type: 'ADD_NOTIFICATION', payload: notification });
  };

  const removeNotification = (id: string) => {
    dispatch({ type: 'REMOVE_NOTIFICATION', payload: id });
  };

  return {
    notifications: state.notifications,
    addNotification,
    removeNotification,
  };
};
```

These examples demonstrate advanced React patterns, custom hooks, complex components, and state management strategies that would be used in the patent analysis web app.