Development Guide

Patent Analysis Web App - Step-by-Step Implementation

Phase 1: Project Setup and Foundation (Week 1-2)

Step 1.1: Initialize Project Structure

```
# Create main project directory
mkdir patent-analysis-app
cd patent-analysis-app

# Initialize backend
mkdir backend
cd backend
npm init -y
npm install express typescript @types/node @types/express
npm install -D nodemon ts-node
cd ..

# Initialize frontend
npx create-react-app frontend --template typescript
cd frontend
npm install @mui/material @emotion/react @emotion/styled
npm install @tanstack/react-query axios react-router-dom
cd ..

# Initialize shared configurations
mkdir config
mkdir deployment
```

Step 1.2: Set Up Development Environment

```
# Install Docker and Docker Compose
# Set up PostgreSQL and Redis containers
docker-compose -f config/docker-compose.dev.yml up -d

# Set up environment variables
cp config/.env.template backend/.env
cp config/.env.template frontend/.env
```

Step 1.3: Configure TypeScript and Build Tools

```
// backend/tsconfig.json
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "lib": ["ES2020"],
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "resolveJsonModule": true,
    "declaration": true,
    "declarationMap": true,
    "sourceMap": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist"]
}
```

Phase 2: Backend Core Development (Week 3-6)

Step 2.1: Database Setup and Models

```
# Install database dependencies
cd backend
npm install pg @types/pg typeorm reflect-metadata
npm install bcryptjs @types/bcryptjs jsonwebtoken @types/jsonwebtoken
```

```
// backend/src/entities/User.ts
import { Entity, PrimaryGeneratedColumn, Column, CreateDateColumn, UpdateDateColumn } f
rom 'typeorm';

@Entity('users')
export class User {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column({ unique: true })
  email: string;

  @Column()
  passwordHash: string;

  @Column()
  firstName: string;

  @Column()
  lastName: string;

  @Column({
    type: 'enum',
    enum: ['inventor', 'corporate_rd', 'patent_attorney', 'business_strategist']
  })
  personaType: string;

  @Column({ nullable: true })
  organization: string;

  @Column({ default: 'free' })
  subscriptionTier: string;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;

  @Column({ nullable: true })
  lastLogin: Date;

  @Column({ default: true })
  isActive: boolean;
}
```

Step 2.2: Authentication System

```
// backend/src/middleware/auth.ts
import jwt from 'jsonwebtoken';
import { Request, Response, NextFunction } from 'express';

export interface AuthRequest extends Request {
  user?: any;
}

export const authenticateToken = (req: AuthRequest, res: Response, next: NextFunction)
=> {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Access token required' });
  }

  jwt.verify(token, process.env.JWT_SECRET!, (err, user) => {
    if (err) {
      return res.status(403).json({ error: 'Invalid or expired token' });
    }
    req.user = user;
    next();
  });
};
```

Step 2.3: Core API Routes

```
// backend/src/routes/auth.ts
import express from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { User } from '../entities/User';
import { AppDataSource } from '../data-source';

const router = express.Router();
const userRepository = AppDataSource.getRepository(User);

router.post('/register', async (req, res) => {
  try {
    const { email, password, firstName, lastName, personaType, organization } = req.bod
y;

    // Check if user exists
    const existingUser = await userRepository.findOne({ where: { email } });
    if (existingUser) {
      return res.status(400).json({ error: 'User already exists' });
    }

    // Hash password
    const passwordHash = await bcrypt.hash(password, 10);

    // Create user
    const user = userRepository.create({
      email,
      passwordHash,
      firstName,
      lastName,
      personaType,
      organization
    });

    await userRepository.save(user);

    // Generate JWT
    const token = jwt.sign(
      { userId: user.id, email: user.email, personaType: user.personaType },
      process.env.JWT_SECRET!,
      { expiresIn: '24h' }
    );

    res.status(201).json({
      message: 'User created successfully',
      token,
      user: {
        id: user.id,
        email: user.email,
        firstName: user.firstName,
        lastName: user.lastName,
        personaType: user.personaType
      }
    });
  } catch (error) {
    res.status(500).json({ error: 'Internal server error' });
  }
});

export default router;
```

Step 2.4: Project and Analysis Models

```
// backend/src/entities/Project.ts
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne, OneToMany, CreateDate-
Column, UpdateDateColumn } from 'typeorm';
import { User } from './User';
import { PatentAnalysis } from './PatentAnalysis';

@Entity('projects')
export class Project {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ManyToOne(() => User, user => user.projects)
  user: User;

  @Column()
  name: string;

  @Column({ type: 'text', nullable: true })
  description: string;

  @Column()
  projectType: string;

  @Column({ default: 'active' })
  status: string;

  @Column({ type: 'jsonb', default: {} })
  settings: object;

  @OneToMany(() => PatentAnalysis, analysis => analysis.project)
  analyses: PatentAnalysis[];

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;
}
```

Phase 3: Frontend Core Development (Week 7-10)

Step 3.1: Authentication Components

```
// frontend/src/components/auth/LoginForm.tsx
import React, { useState } from 'react';
import { Box, TextField, Button, Typography, Alert } from '@mui/material';
import { useMutation } from '@tanstack/react-query';
import { useNavigate } from 'react-router-dom';
import { authAPI } from '../../services/api';

interface LoginFormData {
  email: string;
  password: string;
}

export const LoginForm: React.FC = () => {
  const [formData, setFormData] = useState<LoginFormData>({ email: '', password: '' });
  const [error, setError] = useState<string>('');
  const navigate = useNavigate();

  const loginMutation = useMutation({
    mutationFn: authAPI.login,
    onSuccess: (data) => {
      localStorage.setItem('token', data.token);
      localStorage.setItem('user', JSON.stringify(data.user));
      navigate('/dashboard');
    },
    onError: (error: any) => {
      setError(error.response?.data?.error || 'Login failed');
    }
  });

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    loginMutation.mutate(formData);
  };

  return (
    <Box component="form" onSubmit={handleSubmit} sx={{ maxWidth: 400, mx: 'auto', mt:
4 }}>
      <Typography variant="h4" component="h1" gutterBottom>
        Sign In
      </Typography>

      {error && <Alert severity="error" sx={{ mb: 2 }}>{error}</Alert>}

      <TextField
        fullWidth
        label="Email"
        type="email"
        value={formData.email}
        onChange={(e) => setFormData({ ...formData, email: e.target.value })}
        margin="normal"
        required
      />

      <TextField
        fullWidth
        label="Password"
        type="password"
        value={formData.password}
        onChange={(e) => setFormData({ ...formData, password: e.target.value })}
        margin="normal"
        required
      />
```

```
      <Button
        type="submit"
        fullWidth
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
        disabled={loginMutation.isPending}
      >
        {loginMutation.isPending ? 'Signing In...' : 'Sign In'}
      </Button>
    </Box>
  );
};
```

Step 3.2: Dashboard Components

```tsx
// frontend/src/components/dashboard/InventorDashboard.tsx
import React from 'react';
import { Grid, Card, CardContent, Typography, Box } from '@mui/material';
import { useQuery } from '@tanstack/react-query';
import { dashboardAPI } from '../../services/api';
import { QuickActions } from './QuickActions';
import { RecentAnalyses } from './RecentAnalyses';
import { PatentabilityMeter } from './PatentabilityMeter';

export const InventorDashboard: React.FC = () => {
  const { data: dashboardData, isLoading } = useQuery({
    queryKey: ['dashboard', 'inventor'],
    queryFn: dashboardAPI.getInventorDashboard
  });

  if (isLoading) return <div>Loading...</div>;

  return (
    <Box sx={{ flexGrow: 1, p: 3 }}>
      <Typography variant="h4" component="h1" gutterBottom>
        Inventor Dashboard
      </Typography>

      <Grid container spacing={3}>
        {/* Quick Actions */}
        <Grid item xs={12} md={4}>
          <QuickActions
            actions={[
              { label: 'New Patent Search', action: 'search', icon: 'search' },
              { label: 'Check Patentability', action: 'patentability', icon: 'check' },
              { label: 'Prior Art Analysis', action: 'prior-art', icon: 'analysis' }
            ]}
          />
        </Grid>

        {/* Patentability Overview */}
        <Grid item xs={12} md={4}>
          <Card>
            <CardContent>
              <Typography variant="h6" gutterBottom>
                Recent Patentability Scores
              </Typography>
              <PatentabilityMeter scores={dashboardData?.patentabilityScores || []} />
            </CardContent>
          </Card>
        </Grid>

        {/* Recent Analyses */}
        <Grid item xs={12} md={4}>
          <RecentAnalyses analyses={dashboardData?.recentAnalyses || []} />
        </Grid>

        {/* Search History */}
        <Grid item xs={12}>
          <Card>
            <CardContent>
              <Typography variant="h6" gutterBottom>
                Recent Searches
              </Typography>
              {/* Search history component */}
            </CardContent>
          </Card>
```

```
          </Grid>
        </Grid>
      </Box>
    );
};
```

Step 3.3: Search Interface

```tsx
// frontend/src/components/search/PatentSearch.tsx
import React, { useState } from 'react';
import { Box, TextField, Button, Chip, Grid, Card, CardContent } from '@mui/material';
import { useMutation } from '@tanstack/react-query';
import { searchAPI } from '../../services/api';
import { SearchResults } from './SearchResults';
import { AdvancedFilters } from './AdvancedFilters';

export const PatentSearch: React.FC = () => {
  const [query, setQuery] = useState('');
  const [filters, setFilters] = useState({});
  const [showAdvanced, setShowAdvanced] = useState(false);

  const searchMutation = useMutation({
    mutationFn: searchAPI.searchPatents,
    onSuccess: (data) => {
      // Handle search results
    }
  });

  const handleSearch = () => {
    searchMutation.mutate({ query, filters });
  };

  return (
    <Box sx={{ p: 3 }}>
      <Grid container spacing={3}>
        <Grid item xs={12}>
          <Card>
            <CardContent>
              <Box sx={{ display: 'flex', gap: 2, mb: 2 }}>
                <TextField
                  fullWidth
                  label="Search patents..."
                  value={query}
                  onChange={(e) => setQuery(e.target.value)}
                  placeholder="Enter keywords, patent numbers, or natural language
queries"
                />
                <Button
                  variant="contained"
                  onClick={handleSearch}
                  disabled={!query || searchMutation.isPending}
                >
                  Search
                </Button>
              </Box>

              <Box sx={{ display: 'flex', gap: 1, mb: 2 }}>
                <Chip
                  label="Advanced Filters"
                  onClick={() => setShowAdvanced(!showAdvanced)}
                  variant={showAdvanced ? 'filled' : 'outlined'}
                />
                <Chip label="Semantic Search" variant="outlined" />
                <Chip label="Prior Art Focus" variant="outlined" />
              </Box>

              {showAdvanced && (
                <AdvancedFilters
                  filters={filters}
                  onChange={setFilters}
```

```
                />
              )}
            </CardContent>
          </Card>
        </Grid>

        <Grid item xs={12}>
          {searchMutation.data && (
            <SearchResults
              results={searchMutation.data}
              isLoading={searchMutation.isPending}
            />
          )}
        </Grid>
      </Grid>
    </Box>
  );
};
```

Phase 4: Multi-Agent System Integration (Week 11-14)

Step 4.1: Agent Service Architecture

```python
# backend/agents/base_agent.py
from abc import ABC, abstractmethod
from typing import Dict, Any, List
import asyncio
import logging

class BaseAgent(ABC):
    def __init__(self, name: str, config: Dict[str, Any]):
        self.name = name
        self.config = config
        self.logger = logging.getLogger(f"agent.{name}")

    @abstractmethod
    async def analyze(self, patent_data: Dict[str, Any]) -> Dict[str, Any]:
        """Perform analysis on patent data"""
        pass

    def get_confidence_score(self, analysis_result: Dict[str, Any]) -> float:
        """Calculate confidence score for the analysis"""
        return analysis_result.get('confidence', 0.0)

    def format_result(self, raw_result: Dict[str, Any]) -> Dict[str, Any]:
        """Format the analysis result for consumption"""
        return {
            'agent_name': self.name,
            'result': raw_result,
            'confidence': self.get_confidence_score(raw_result),
            'timestamp': asyncio.get_event_loop().time()
        }
```

Step 4.2: Specific Agent Implementations

```python
# backend/agents/prior_art_agent.py
from .base_agent import BaseAgent
import openai
from typing import Dict, Any

class PriorArtAgent(BaseAgent):
    def __init__(self, config: Dict[str, Any]):
        super().__init__("prior_art", config)
        self.openai_client = openai.OpenAI(api_key=config['openai_api_key'])

    async def analyze(self, patent_data: Dict[str, Any]) -> Dict[str, Any]:
        """Analyze patent for prior art"""
        try:
            # Extract key information from patent
            patent_title = patent_data.get('title', '')
            patent_abstract = patent_data.get('abstract', '')
            patent_claims = patent_data.get('claims', [])

            # Generate search queries for prior art
            search_queries = await self._generate_search_queries(patent_title, pat-
ent_abstract)

            # Search for similar patents
            similar_patents = await self._search_similar_patents(search_queries)

            # Analyze novelty
            novelty_analysis = await self._analyze_novelty(patent_data, simil-
ar_patents)

            return self.format_result({
                'similar_patents': similar_patents,
                'novelty_score': novelty_analysis['score'],
                'novelty_explanation': novelty_analysis['explanation'],
                'search_queries_used': search_queries,
                'confidence': novelty_analysis['confidence']
            })

        except Exception as e:
            self.logger.error(f"Prior art analysis failed: {str(e)}")
            return self.format_result({
                'error': str(e),
                'confidence': 0.0
            })

    async def _generate_search_queries(self, title: str, abstract: str) -> List[str]:
        """Generate effective search queries using GPT"""
        prompt = f"""
        Given this patent title and abstract, generate 5 effective search queries to
find prior art:

        Title: {title}
        Abstract: {abstract}

        Generate queries that focus on:
        1. Core technical concepts
        2. Alternative implementations
        3. Related technologies
        4. Broader application areas
        5. Specific technical features

        Return only the queries, one per line.
        """
```

```
        response = await self.openai_client.chat.completions.create(
            model="gpt-4",
            messages=[{"role": "user", "content": prompt}],
            max_tokens=500
        )

        queries = response.choices[0].message.content.strip().split('\n')
        return [q.strip() for q in queries if q.strip()]
```

Step 4.3: Agent Coordinator

```typescript
// backend/src/services/AgentCoordinator.ts
import { PythonShell } from 'python-shell';
import { PatentAnalysis } from '../entities/PatentAnalysis';
import { AppDataSource } from '../data-source';

export class AgentCoordinator {
  private analysisRepository = AppDataSource.getRepository(PatentAnalysis);

  async coordinateAnalysis(
    patentData: any,
    analysisType: string,
    analysisId: string
  ): Promise<any> {
    try {
      // Update analysis status
      await this.updateAnalysisStatus(analysisId, 'processing');

      // Determine which agents to run
      const agentsToRun = this.selectAgents(analysisType);

      // Run agents in parallel
      const agentPromises = agentsToRun.map(agentName =>
        this.runAgent(agentName, patentData)
      );

      const agentResults = await Promise.allSettled(agentPromises);

      // Process results
      const processedResults = this.processAgentResults(agentResults);

      // Generate summary
      const summary = await this.generateSummary(processedResults);

      // Save results
      await this.saveAnalysisResults(analysisId, {
        agentResults: processedResults,
        summary,
        status: 'completed'
      });

      return {
        agentResults: processedResults,
        summary,
        analysisId
      };

    } catch (error) {
      await this.updateAnalysisStatus(analysisId, 'failed');
      throw error;
    }
  }

  private selectAgents(analysisType: string): string[] {
    const agentMap = {
      'comprehensive': ['prior_art', 'claims', 'market', 'legal'],
      'prior_art_only': ['prior_art'],
      'patentability': ['prior_art', 'legal'],
      'competitive': ['market', 'prior_art'],
      'legal_review': ['legal', 'claims']
    };

    return agentMap[analysisType] || ['prior_art'];
```

```
  }

  private async runAgent(agentName: string, patentData: any): Promise<any> {
    return new Promise((resolve, reject) => {
      const options = {
        mode: 'json' as const,
        pythonPath: 'python3',
        scriptPath: './agents/',
        args: [JSON.stringify({ agent: agentName, data: patentData })]
      };

      PythonShell.run('run_agent.py', options, (err, results) => {
        if (err) {
          reject(err);
        } else {
          resolve(results?.[0] || {});
        }
      });
    });
  }
}
```

Phase 5: Batch Processing System (Week 15-16)

Step 5.1: Batch Job Queue

```typescript
// backend/src/services/BatchProcessor.ts
import Queue from 'bull';
import { BatchJob } from '../entities/BatchJob';
import { AppDataSource } from '../data-source';
import { AgentCoordinator } from './AgentCoordinator';

const batchQueue = new Queue('batch processing', {
  redis: { host: process.env.REDIS_HOST, port: parseInt(process.env.REDIS_PORT!) }
});

export class BatchProcessor {
  private batchJobRepository = AppDataSource.getRepository(BatchJob);
  private agentCoordinator = new AgentCoordinator();

  async submitBatchJob(
    userId: string,
    projectId: string,
    jobData: {
      name: string;
      type: string;
      inputData: any[];
      analysisType: string;
    }
  ): Promise<string> {
    // Create batch job record
    const batchJob = this.batchJobRepository.create({
      userId,
      projectId,
      jobName: jobData.name,
      jobType: jobData.type,
      inputData: jobData.inputData,
      totalItems: jobData.inputData.length,
      status: 'queued'
    });

    await this.batchJobRepository.save(batchJob);

    // Add to queue
    await batchQueue.add('process-batch', {
      batchJobId: batchJob.id,
      analysisType: jobData.analysisType,
      inputData: jobData.inputData
    }, {
      attempts: 3,
      backoff: {
        type: 'exponential',
        delay: 2000
      }
    });

    return batchJob.id;
  }

  setupBatchProcessor() {
    batchQueue.process('process-batch', async (job) => {
      const { batchJobId, analysisType, inputData } = job.data;

      // Update job status
      await this.updateBatchJobStatus(batchJobId, 'processing');

      const results = [];
      const errors = [];
```

```
      for (let i = 0; i < inputData.length; i++) {
        try {
          // Process individual item
          const result = await this.agentCoordinator.coordinateAnalysis(
            inputData[i],
            analysisType,
            `${batchJobId}-${i}`
          );

          results.push(result);

          // Update progress
          const progress = Math.round(((i + 1) / inputData.length) * 100);
          job.progress(progress);

          await this.updateBatchJobProgress(batchJobId, i + 1, 0);

        } catch (error) {
          errors.push({
            index: i,
            item: inputData[i],
            error: error.message
          });

          await this.updateBatchJobProgress(batchJobId, i, 1);
        }
      }

      // Save final results
      await this.completeBatchJob(batchJobId, results, errors);
    });
  }
}
```

Step 5.2: Batch Progress UI

```
// frontend/src/components/batch/BatchProgress.tsx
import React, { useEffect } from 'react';
import { Box, LinearProgress, Typography, Card, CardContent, List, ListItem } from '@mu
i/material';
import { useQuery } from '@tanstack/react-query';
import { io } from 'socket.io-client';
import { batchAPI } from '../../services/api';

interface BatchProgressProps {
  batchJobId: string;
}

export const BatchProgress: React.FC<BatchProgressProps> = ({ batchJobId }) => {
  const [progress, setProgress] = React.useState(0);
  const [status, setStatus] = React.useState('queued');

  const { data: batchJob, refetch } = useQuery({
    queryKey: ['batchJob', batchJobId],
    queryFn: () => batchAPI.getBatchJob(batchJobId),
    refetchInterval: status === 'processing' ? 2000 : false
  });

  useEffect(() => {
    const socket = io(process.env.REACT_APP_API_URL!);

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

    socket.on('batch-progress', (data) => {
      if (data.jobId === batchJobId) {
        setProgress(data.progress);
        setStatus(data.status);
        refetch();
      }
    });

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

  if (!batchJob) return <div>Loading...</div>;

  return (
    <Card>
      <CardContent>
        <Typography variant="h6" gutterBottom>
          Batch Job: {batchJob.jobName}
        </Typography>

        <Box sx={{ mb: 2 }}>
          <Typography variant="body2" color="text.secondary">
            Status: {status}
          </Typography>
          <LinearProgress
            variant="determinate"
            value={progress}
            sx={{ mt: 1 }}
          />
          <Typography variant="body2" sx={{ mt: 1 }}>
            {batchJob.completedItems} of {batchJob.totalItems} items processed
          </Typography>
        </Box>
```

```
        {batchJob.errorLog && batchJob.errorLog.length > 0 && (
          <Box>
            <Typography variant="subtitle2" color="error">
              Errors ({batchJob.errorLog.length}):
            </Typography>
            <List dense>
              {batchJob.errorLog.slice(0, 5).map((error, index) => (
                <ListItem key={index}>
                  <Typography variant="body2" color="error">
                    {error}
                  </Typography>
                </ListItem>
              ))}
            </List>
          </Box>
        )}
      </CardContent>
    </Card>
  );
};
```

Phase 6: Testing and Quality Assurance (Week 17-18)

Step 6.1: Backend Testing Setup

```
// backend/src/tests/auth.test.ts
import request from 'supertest';
import { app } from '../app';
import { AppDataSource } from '../data-source';

describe('Authentication', () => {
  beforeAll(async () => {
    await AppDataSource.initialize();
  });

  afterAll(async () => {
    await AppDataSource.destroy();
  });

  describe('POST /api/auth/register', () => {
    it('should register a new user', async () => {
      const userData = {
        email: 'test@example.com',
        password: 'password123',
        firstName: 'Test',
        lastName: 'User',
        personaType: 'inventor'
      };

      const response = await request(app)
        .post('/api/auth/register')
        .send(userData)
        .expect(201);

      expect(response.body).toHaveProperty('token');
      expect(response.body.user.email).toBe(userData.email);
    });

    it('should not register user with existing email', async () => {
      const userData = {
        email: 'existing@example.com',
        password: 'password123',
        firstName: 'Test',
        lastName: 'User',
        personaType: 'inventor'
      };

      // Register first user
      await request(app).post('/api/auth/register').send(userData);

      // Try to register again
      const response = await request(app)
        .post('/api/auth/register')
        .send(userData)
        .expect(400);

      expect(response.body.error).toBe('User already exists');
    });
  });
});
```

Step 6.2: Frontend Testing Setup

```
// frontend/src/components/__tests__/LoginForm.test.tsx
import React from 'react';
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import { BrowserRouter } from 'react-router-dom';
import { LoginForm } from '../auth/LoginForm';

const createWrapper = () => {
  const queryClient = new QueryClient({
    defaultOptions: {
      queries: { retry: false },
      mutations: { retry: false }
    }
  });

  return ({ children }: { children: React.ReactNode }) => (
    <QueryClientProvider client={queryClient}>
      <BrowserRouter>
        {children}
      </BrowserRouter>
    </QueryClientProvider>
  );
};

describe('LoginForm', () => {
  it('renders login form', () => {
    render(<LoginForm />, { wrapper: createWrapper() });

    expect(screen.getByLabelText(/email/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/password/i)).toBeInTheDocument();
    expect(screen.getByRole('button', { name: /sign in/i })).toBeInTheDocument();
  });

  it('submits form with valid data', async () => {
    render(<LoginForm />, { wrapper: createWrapper() });

    fireEvent.change(screen.getByLabelText(/email/i), {
      target: { value: 'test@example.com' }
    });
    fireEvent.change(screen.getByLabelText(/password/i), {
      target: { value: 'password123' }
    });

    fireEvent.click(screen.getByRole('button', { name: /sign in/i }));

    await waitFor(() => {
      expect(screen.getByText(/signing in/i)).toBeInTheDocument();
    });
  });
});
```

Phase 7: Deployment and Production Setup (Week 19-20)

Step 7.1: Production Docker Configuration

```
# backend/Dockerfile
FROM node:18-alpine

WORKDIR /app

# Copy package files
COPY package*.json ./
RUN npm ci --only=production

# Copy source code
COPY . .

# Build TypeScript
RUN npm run build

# Expose port
EXPOSE 5000

# Start application
CMD ["npm", "start"]
```

```
# frontend/Dockerfile
FROM node:18-alpine as build

WORKDIR /app
COPY package*.json ./
RUN npm ci

COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Step 7.2: Production Docker Compose

```yaml
# deployment/docker-compose.prod.yml
version: '3.8'

services:
  frontend:
    build:
      context: ../frontend
      dockerfile: Dockerfile
    ports:
      - "80:80"
    environment:
      - REACT_APP_API_URL=https://api.patentanalysis.com
    depends_on:
      - backend

  backend:
    build:
      context: ../backend
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=${DATABASE_URL}
      - REDIS_URL=${REDIS_URL}
      - JWT_SECRET=${JWT_SECRET}
    depends_on:
      - db
      - redis
    volumes:
      - ./logs:/app/logs

  db:
    image: postgres:14
    environment:
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./backups:/backups
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data
    ports:
      - "6379:6379"

  nginx:
    image: nginx:alpine
    ports:
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - frontend
      - backend
```

```
volumes:
  postgres_data:
  redis_data:
```

Step 7.3: CI/CD Pipeline

```yaml
# .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:14
        env:
          POSTGRES_PASSWORD: postgres
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'
          cache-dependency-path: |
            backend/package-lock.json
            frontend/package-lock.json

      - name: Install backend dependencies
        run: |
          cd backend
          npm ci

      - name: Install frontend dependencies
        run: |
          cd frontend
          npm ci

      - name: Run backend tests
        run: |
          cd backend
          npm run test
        env:
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test

      - name: Run frontend tests
        run: |
          cd frontend
          npm run test -- --coverage --watchAll=false

      - name: Build frontend
        run: |
          cd frontend
          npm run build

  deploy:
```

```
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - uses: actions/checkout@v3

      - name: Deploy to production
        run: |
          # Add deployment script here
          echo "Deploying to production..."
```

This comprehensive development guide provides step-by-step instructions for implementing the patent analysis web app. Each phase builds upon the previous one, ensuring a systematic approach to development. The guide includes code examples, testing strategies, and deployment configurations to support the complete development lifecycle.