**Complete AI Resume Parser and Job Recommendation Roadmap**

**Phase 1: Core Engine Components (Week 1)**

**Day 1: Enhanced Resume Parser**

**File**: advanced\_resume\_parser.py **Dependencies**: Extends existing resume\_parser.py **Cursor Prompt**: #1

# Key components to implement:

class AdvancedResumeParser(ResumeParser):

- Field classification algorithm

- Experience level detection

- Career trajectory analysis

- Skills categorization

- Leadership indicators extraction

**Test File**: test\_advanced\_parser.py

# Test scenarios:

- Sample resumes from target demographic

- Field detection accuracy validation

- Experience level calculation tests

**Day 2: Income-Aware Job Matcher**

**File**: income\_boost\_job\_matcher.py **Dependencies**: Requires income\_comparator.py, job\_searcher.py **Cursor Prompt**: #2

# Key components to implement:

class IncomeBoostJobMatcher:

- Salary-focused search parameters

- Multi-dimensional job scoring

- Field-specific search strategies

- Company quality assessment

**Test File**: test\_job\_matcher.py

# Test scenarios:

- Salary filtering accuracy

- Job scoring algorithm validation

- MSA targeting effectiveness

**Day 3: Three-Tier Selection Engine**

**File**: three\_tier\_selector.py **Dependencies**: Requires income\_boost\_job\_matcher.py **Cursor Prompt**: #3

# Key components to implement:

class ThreeTierJobSelector:

- Conservative tier selection (15-20% increase)

- Optimal tier selection (25-30% increase)

- Stretch tier selection (35%+ increase)

- Diversity enforcement algorithm

- Application strategy generation

**Test File**: test\_tier\_selector.py

# Test scenarios:

- Tier classification accuracy

- Diversity validation across tiers

- Application strategy appropriateness

**Day 4: Master Recommendation Engine**

**File**: mingus\_job\_recommendation\_engine.py **Dependencies**: All previous components **Cursor Prompt**: #4

# Key components to implement:

class MingusJobRecommendationEngine:

- Complete workflow orchestration

- Error handling and recovery

- Performance optimization

- Result caching

- Analytics integration

**Configuration File**: recommendation\_config.py

# Configuration settings:

- Processing time targets

- Salary increase thresholds

- MSA targeting parameters

- API rate limiting settings

**Day 5: Integration Testing**

**File**: test\_complete\_workflow.py **Dependencies**: All engine components

# End-to-end testing:

- Complete user journey validation

- Performance benchmarking

- Error scenario testing

- Data consistency validation

**Phase 2: Web Interface Integration (Week 2)**

**Day 6: Flask Application Enhancement**

**File**: app.py (Enhanced existing file) **Dependencies**: All engine components **Cursor Prompt**: #5

# New routes and enhancements:

- Enhanced /upload route

- /process-recommendations endpoint

- /api/recommendations endpoint

- Progress tracking endpoints

**File**: forms.py (New file)

# Enhanced form classes:

class EnhancedResumeUploadForm:

- Demographic collection fields

- Career goals and preferences

- Salary expectations

- Geographic flexibility

**Day 7: Enhanced User Interface**

**File**: templates/enhanced\_upload.html **Dependencies**: Enhanced Flask routes **Cursor Prompt**: #6

**File**: templates/targeted\_results.html

<!-- New template sections: -->

- Three-tier recommendation cards

- Income impact visualizations

- Skills gap analysis displays

- Application strategy guidance

**File**: static/css/recommendations.css

/\* Styling for: \*/

- Three-tier card layout

- Income improvement visualizations

- Progress indicators

- Mobile-responsive design

**File**: static/js/recommendation\_ui.js

// Interactive features:

- Progress tracking during processing

- Expandable recommendation details

- Application tracking functionality

- Mobile touch interactions

**Day 8: Testing and Quality Assurance**

**Files**: Complete testing suite **Cursor Prompt**: #7

**File**: tests/test\_recommendation\_quality.py

# Quality assurance tests:

- Recommendation appropriateness validation

- Demographic accuracy testing

- Income calculation verification

**File**: tests/test\_user\_scenarios.py

# User scenario testing:

- Target demographic profiles

- Various metro area scenarios

- Different career stages

- Edge case handling

**File**: test\_data/sample\_profiles.json

// Sample user profiles for testing:

- African American professionals 25-35

- Various education levels

- Different metro areas

- Income ranges $40k-$100k

**Day 9: Analytics and Performance**

**File**: analytics\_tracker.py **Dependencies**: Recommendation engine **Cursor Prompt**: #8

# Analytics components:

class RecommendationAnalytics:

- User behavior tracking

- Recommendation effectiveness metrics

- Performance monitoring

- A/B testing framework

**File**: performance\_monitor.py

# Performance monitoring:

- Response time tracking

- Resource usage monitoring

- API usage optimization

- Cost tracking

**File**: admin/analytics\_dashboard.py

# Admin dashboard:

- Real-time metrics display

- Recommendation quality reports

- User success tracking

- System health monitoring

**Day 10: Production Deployment**

**File**: deployment/production\_config.py **Dependencies**: All components

**File**: deployment/heroku\_setup.sh

# Deployment scripts:

- Environment variable setup

- Database migrations

- Static asset compilation

- Health check configuration

**File**: monitoring/health\_checks.py

# Production monitoring:

- System health endpoints

- Performance alerting

- Error tracking

- User success metrics

**Supporting Artifacts**

**Documentation Files**

**File**: docs/api\_documentation.md

# API Documentation:

- Endpoint specifications

- Request/response formats

- Authentication requirements

- Rate limiting information

**File**: docs/user\_guide.md

# User Guide:

- Feature explanation

- Best practices for resume uploads

- Understanding recommendations

- Application strategies

**File**: docs/admin\_guide.md

# Admin Guide:

- System monitoring procedures

- Performance optimization

- Troubleshooting common issues

- Analytics interpretation

**Database Schema (if needed)**

**File**: database/recommendation\_schema.sql

-- Tables for:

- User profiles (optional)

- Recommendation history

- Analytics tracking

- Performance metrics

**File**: database/migrations/001\_initial\_schema.sql

-- Initial database setup

-- Index creation for performance

-- Data retention policies

**Configuration Files**

**File**: .env.example

# Environment variables:

- API keys for job search services

- Database connection strings

- Analytics tracking IDs

- Performance monitoring tokens

**File**: requirements.txt (Updated)

# Additional dependencies:

- Advanced analytics libraries

- Performance monitoring tools

- Testing frameworks

- Deployment utilities

**Implementation Sequence**

**Critical Path Dependencies:**

1. **resume\_parser.py** → **advanced\_resume\_parser.py**
2. **income\_comparator.py** → **income\_boost\_job\_matcher.py**
3. **job\_searcher.py** → **income\_boost\_job\_matcher.py**
4. **income\_boost\_job\_matcher.py** → **three\_tier\_selector.py**
5. **All components** → **mingus\_job\_recommendation\_engine.py**
6. **Engine** → **Flask integration**
7. **Flask** → **Templates and UI**
8. **Complete system** → **Testing and analytics**

**Daily Validation Checkpoints:**

* **Day 1**: Advanced parser correctly identifies fields and experience levels
* **Day 2**: Job matcher finds salary-appropriate opportunities
* **Day 3**: Three-tier selector produces diverse, appropriate recommendations
* **Day 4**: Complete engine processes sample resumes end-to-end
* **Day 5**: All components pass integration tests
* **Day 6**: Flask integration handles file uploads and displays results
* **Day 7**: UI displays three-tier recommendations attractively
* **Day 8**: All edge cases and user scenarios pass testing
* **Day 9**: Analytics track user behavior and recommendation effectiveness
* **Day 10**: Production deployment successfully serves real users

**Success Metrics:**

* **Processing Time**: <8 seconds end-to-end
* **Recommendation Quality**: 3 jobs, all 15%+ salary increase
* **User Experience**: Clear, actionable next steps for each opportunity
* **Technical Performance**: Handles 50+ concurrent users
* **Business Impact**: Clear path from current income to 25-45% increases

This roadmap ensures systematic development with clear dependencies, validation points, and measurable success criteria at each stage.