AI-Powered Resident Matching Engine - Complete Implementation Guide

📋 Overview

The AI-Powered Resident Matching Engine is a sophisticated system that helps families find the perfect assisted living home based on their specific needs, preferences, and circumstances. It combines weighted algorithm scoring with OpenAI-generated explanations to provide a "Tinder for senior care" experience.

🎯 Features Implemented

1. **Multi-step Input Form** - Collects family preferences across 4 steps
2. **Intelligent Matching Algorithm** - Weighted scoring system (Budget 30%, Condition 25%, Care Level 20%, Location 15%, Amenities 10%)
3. **AI-Powered Explanations** - GPT-4 generated natural language explanations for each match
4. **Results Visualization** - Beautiful UI showing top 5 matches with scores and insights
5. **Feedback System** - Thumbs up/down and placement confirmation tracking
6. **Analytics Ready** - All data stored for future ML training and reporting

🗂️ Project Structure

Database Schema (`prisma/schema.prisma`)

```
MatchRequest
├── Budget preferences (min/max)
├── Medical conditions
├── Care level required
├── Lifestyle preferences (gender, religion, dietary, hobbies, pets)
├── Location (zip code, max distance)
└── Timeline

MatchResult
├── Home reference
├── Fit score (0-100)
├── Match factors breakdown
├── AI explanation
└── Rank (1-5)

MatchFeedback
├── Home reference
├── Feedback type (THUMBS_UP, THUMBS_DOWN, PLACEMENT_CONFIRMED)
└── Optional notes
```

API Endpoints

- `POST /api/family/match` - Create match request and run algorithm
- `GET /api/family/match` - List all match requests for authenticated family
- `GET /api/family/match/[id]` - Get specific match request with results

- `POST /api/family/match/[id]/feedback` - Submit feedback on a match
- `GET /api/family/match/[id]/feedback` - Get all feedback for a match

Core Services

- `src/lib/matching/matching-algorithm.ts` - Scoring algorithm
- `src/lib/matching/openai-explainer.ts` - AI explanation generation

User Interface

- `src/app/dashboard/find-care/page.tsx` - 4-step input form
- `src/app/dashboard/find-care/results/[id]/page.tsx` - Results display

🔢 Matching Algorithm Details

Scoring Weights

- **Budget Match (30%)**: Price range overlap with family budget
- **Condition Compatibility (25%)**: Medical condition to amenity mapping
- **Care Level Match (20%)**: Exact or compatible care level
- **Location (15%)**: Proximity to family (zip code based)
- **Amenities (10%)**: Religion, dietary, pets, hobbies match

Budget Scoring Logic

- Perfect match (70-100): Home price overlaps with budget
- Good match (60-69): Within 10% of budget range
- Okay match (40-59): Within 20% of budget range
- Poor match (20-39): Within 30% of budget range
- No match (0-19): More than 30% outside budget

Condition Scoring

Maps medical conditions to expected amenities:
- Dementia → Memory Care, Secure Unit
- Diabetes → Medication Management, Diabetic Meals
- Mobility Issues → Wheelchair Accessible, Physical Therapy
- And more…

Care Level Scoring

- 100: Exact match
- 80: Compatible care level (e.g., Memory Care can provide Assisted Living)
- 70: Skilled Nursing can provide any care level
- 0: No compatible care level

🤖 OpenAI Integration

Model Used

GPT-4 with temperature 0.7 for natural, personalized responses

Prompt Structure

System prompt: “You are a compassionate senior care advisor…”

User prompt includes:
- Home name and fit score
- Key strengths (factors scoring >70)
- Potential concerns (factors scoring <50)
- Home amenities
- Request for warm, specific, reassuring explanation

Fallback Strategy

If OpenAI fails, uses template-based explanations to ensure system reliability

Rate Limiting

Batch processing with 3 concurrent requests and 1-second delays between batches

🎨 User Interface Design

Input Form (4 Steps)

1. **Budget & Care Level**
   - Min/max budget sliders
   - Care level cards (Independent, Assisted, Memory, Skilled)

2. **Medical Conditions**
   - 12 common conditions as toggleable buttons
   - Optional “Other” option

3. **Lifestyle Preferences**
   - Caregiver gender preference
   - Religion/cultural preferences
   - Dietary needs (8 options)
   - Hobbies & interests (12 options)
   - Pet preferences

4. **Location & Timeline**
   - Zip code input
   - Maximum distance slider
   - Move-in timeline (4 options)

Results Page

- **Top 5 Matches** displayed as cards
- **Fit Score Badge** (color-coded: green >80, blue >60, yellow >40, red <40)
- **Rank Indicator** (#1 through #5)
- **AI Explanation** in highlighted box
- **Individual Factor Scores** (5 mini scores)
- **Home Details**: Photos, pricing, location, amenities, availability
- **Action Buttons**: View Profile, Schedule Tour, Feedback
- **Feedback Buttons**: 👍 Like, 👎 Not Interested, ✓ Chose This Home

🔐 Security & Authorization

- All endpoints require authentication (`requireAuth()`)

- Family can only access their own match requests
- RBAC permissions enforced
- Audit logging for all match requests and feedback

📊 Analytics & Future ML

Data Collected

- **Match Requests**: All family preferences and requirements
- **Match Results**: Fit scores and factor breakdowns for each home
- **Feedback**: User reactions to recommendations
- **Placement Outcomes**: Which homes families actually choose

Future Enhancements

1. **Machine Learning Model**: Train on feedback data to improve scoring weights
2. **Personalization**: Learn family-specific preferences over time
3. **A/B Testing**: Test different scoring algorithms
4. **Predictive Analytics**: Predict placement likelihood
5. **Collaborative Filtering**: “Families like you also liked…”

🚀 Deployment Instructions

Environment Variables Required

```
OPENAI_API_KEY=sk-...  # Required for AI explanations
DATABASE_URL=postgresql://...  # PostgreSQL database
```

Migration Steps

1. Generate Prisma client: `npx prisma generate`
2. Run migrations: `npx prisma migrate deploy`
3. (Optional) Seed demo data if needed

Production Considerations

- **OpenAI Rate Limits**: Monitor API usage and implement caching
- **Database Indexing**: All key fields are indexed for performance
- **Error Handling**: Fallback explanations if OpenAI fails
- **Monitoring**: Track match request success rates and user feedback

🧪 Testing Guide

Manual Testing Flow

1. **Access Form**: Navigate to `/dashboard/find-care`
2. **Fill Step 1**: Set budget ($3000-$5000) and care level (Assisted Living)
3. **Fill Step 2**: Select medical conditions (Dementia, Diabetes)
4. **Fill Step 3**: Set preferences (dietary needs, hobbies)
5. **Fill Step 4**: Enter zip code and distance
6. **Submit**: Click “Find My Perfect Match”

7. **View Results**: See top 5 matches with scores
8. **Test Feedback**: Click thumbs up/down, try placement confirmation
9. **Verify API**: Check network tab for API calls

API Testing with cURL

```
# Create match request
curl -X POST http://localhost:5000/api/family/match \
  -H "Content-Type: application/json" \
  -H "Cookie: next-auth.session-token=..." \
  -d '{
    "budgetMin": 3000,
    "budgetMax": 5000,
    "careLevel": "ASSISTED_LIVING",
    "medicalConditions": ["Dementia"],
    "zipCode": "94102",
    "maxDistance": 25,
    "moveInTimeline": "1_3_MONTHS"
  }'

# Get match results
curl http://localhost:5000/api/family/match/{matchRequestId} \
  -H "Cookie: next-auth.session-token=..."

# Submit feedback
curl -X POST http://localhost:5000/api/family/match/{matchRequestId}/feedback \
  -H "Content-Type: application/json" \
  -H "Cookie: next-auth.session-token=..." \
  -d '{
    "homeId": "{homeId}",
    "feedbackType": "THUMBS_UP"
  }'
```

📝 Known Limitations & Future Improvements

Current Limitations

1. **Location Scoring**: Uses simplified zip code comparison instead of geocoding
2. **Medical Conditions**: Static mapping, could be dynamic based on home capabilities
3. **OpenAI Dependency**: Requires API key and has rate limits
4. **No Realtime Updates**: Results are static after generation

Planned Improvements

1. **Google Maps Integration**: Accurate distance calculation
2. **Real-time Availability**: Check bed availability in real-time
3. **Tour Scheduling**: Integrate calendar booking system
4. **Messaging**: Direct messaging with operators
5. **Comparison Mode**: Side-by-side home comparison
6. **Saved Searches**: Save preferences for future searches
7. **Email Notifications**: Alert families when new matches become available
8. **Mobile App**: Native mobile experience
9. **Video Tours**: Virtual tour integration
10. **Reviews Integration**: Show family reviews and ratings

🎓 Technical Decisions & Rationale

Why Weighted Scoring?

- **Transparent**: Families can see exactly why a home was recommended
- **Tunable**: Weights can be adjusted based on feedback
- **Explainable**: Easy to understand and communicate to users

Why OpenAI GPT-4?

- **Natural Language**: Generates warm, personalized explanations
- **Context Aware**: Understands nuances of senior care
- **Scalable**: Can handle many requests per second
- **Fallback Ready**: Template system ensures reliability

Why Multi-Step Form?

- **Reduced Cognitive Load**: One topic at a time
- **Better Completion Rates**: Users less likely to abandon
- **Progressive Disclosure**: Only show relevant fields
- **Validation at Each Step**: Catch errors early

Why PostgreSQL + Prisma?

- **Type Safety**: Prisma provides full TypeScript types
- **Migrations**: Safe schema evolution
- **Performance**: Excellent for complex queries
- **Scalability**: Can handle millions of records

🎉 Implementation Timeline

- **Phase 1** (Database & API): 2 hours - ✅ Complete
- **Phase 2** (Matching Algorithm): 3 hours - ✅ Complete
- **Phase 3** (OpenAI Integration): 2 hours - ✅ Complete
- **Phase 4** (Input Form UI): 3 hours - ✅ Complete
- **Phase 5** (Results Page UI): 3 hours - ✅ Complete
- **Phase 6** (Feedback System): 1 hour - ✅ Complete
- **Total**: ~14 hours of development

📞 Support & Maintenance

Common Issues

1. **OpenAI Rate Limit**: Implement caching or queue system
2. **Slow Matching**: Optimize database queries, add indexes
3. **No Results**: Adjust scoring weights or expand search radius
4. **Invalid Zip Code**: Add zip code validation and geocoding

Monitoring Checklist

- [ ] OpenAI API usage and costs
- [ ] Match request success rate

- [ ] Average match scores
- [ ] Feedback distribution (thumbs up vs down)
- [ ] Placement confirmation rate
- [ ] Page load times
- [ ] API response times
- [ ] Error rates

🔗 Related Documentation

- Prisma Schema (/prisma/schema.prisma)
- Matching Algorithm (/src/lib/matching/matching-algorithm.ts)
- OpenAI Explainer (/src/lib/matching/openai-explainer.ts)
- API Routes (/src/app/api/family/match/)
- UI Components (/src/app/dashboard/find-care/)

Built with ❤️ for CareLinkAI | Last Updated: December 15, 2025