Workout Generator - Implementation Guide

A rule-based exercise recommendation system that generates personalized weekly workout plans based on user profiles, fitness goals, physical limitations, and equipment availability.

🎯 Features

- **Safety First**: Automatically filters exercises based on physical limitations and contraindications
- **Personalized**: Adapts to fitness level, goals, age, and equipment availability
- **Flexible**: Supports 2, 3, or 5-day workout splits
- **Goal-Aligned**: Exercise selection matches user fitness goals (weight loss, muscle gain, strength, etc.)
- **Data-Driven**: Uses exercise programming data for accurate time and calorie estimates
- **Complete Plans**: Generates full weekly schedules with exercise details, sets, reps, and rest periods

---

📋 Prerequisites

Required Files

1. **fitplan.db** - User database with profiles
2. **exercises.db** - Exercise database with contraindications and programming data
3. **workout_generator.py** - Main generator class (provided)
4. **app.py** - Your Flask application

Database Tables Required

fitplan.db:

- `users` table with columns: id, age, gender, weight, fitness_goals, activity_level, physical_limitations, available_equipment, tdee, bmr

exercises.db:

- `exercises` - Core exercise data
- `exercise_primary_muscles` - Primary muscle targeting
- `exercise_contraindications` - Safety contraindications
- `contraindications` - Contraindication details
- `modification_categories` - Limitation categories
- `exercise_programming` - Sets, reps, rest, calorie data

🚀 Quick Start

1. Test the Generator Standalone

First, verify the workout generator works independently:

```bash
# Run the test suite
python test_workout_generator.py
```

This will:

- Create test users with different profiles
- Generate workout plans for each
- Validate plan structure
- Save JSON output files

Expected Output:

```
📋 Setting up test users...
✅ Created/found 3 test users

👤 Testing User: 1
  Goal: weight_loss
  Activity: lightly_active
  Limitations: ['back_problems']
  Equipment: ['dumbbells', 'resistance_bands']

⚙️  Generating workout plan...
✅ Plan Generated Successfully!
  Fitness Level: beginner
  Workout Days: 3
  Total Weekly Calories: 1247.5 kcal

📅 Weekly Schedule:
  Monday     - Push             (4 exercises, 42 min, 415 kcal)
  Wednesday  - Pull             (4 exercises, 38 min, 390 kcal)
  Friday     - Legs & Core      (4 exercises, 35 min, 443 kcal)

💾 Saved to test_plan_user_1.json
```

2. Integrate with Flask App

Add the generator to your Flask app:

```python
```

```
# At the top of app.py, after app initialization
from workout_generator import WorkoutGenerator

workout_generator = WorkoutGenerator(
    fitplan_db='fitplan.db',
    exercise_db='exercises.db'
)
```

3. Replace Placeholder Endpoints

Replace the existing `/api/generate-workout-plan` endpoint with the one from `app_integration.py`.

Also update the `/create-plan` route to use real workout generation instead of sample data.

4. Test in the Web App

1. Start your Flask app: `python app.py`
2. Navigate to `http://localhost:5000`
3. Complete the onboarding questionnaire
4. Click "Create Your Plan"
5. View your personalized workout on the dashboard

📊 How It Works

Stage 1: User Profile Analysis

```python
user_profile = {
    'age': 32,
    'gender': 'male',
    'weight': 180,  # lbs
    'fitness_goal': 'weight_loss',
    'activity_level': 'lightly_active',
    'physical_limitations': ['back_problems'],
    'available_equipment': ['dumbbells', 'resistance_bands'],
    'tdee': 2200,
    'bmr': 1800
}
```

Fitness Level Determination:

- Activity level + Age → Beginner/Intermediate/Advanced
- Older users (55+) downgraded one level for safety

Workout Frequency:

- Beginner: 3 days/week
- Intermediate: 3 days/week
- Advanced: 5 days/week
- Adjusted for goals (muscle gain/weight loss may add 1 day)

Stage 2: Exercise Filtering

Step 1: Filter by Physical Limitations

```sql
-- Exclude exercises with high/moderate severity contraindications
-- matching user's physical limitations
```

Example: User with `back_problems` excludes exercises with:

- "back and spinal issues" category
- Severity: high or moderate

Step 2: Filter by Equipment

```sql
-- Include exercises using available equipment
-- Always include bodyweight exercises
```

Example: User with `['dumbbells', 'resistance_bands']` gets:

- Dumbbell exercises
- Band exercises
- Bodyweight exercises

Step 3: Filter by Fitness Level

- Beginner: Only beginner exercises
- Intermediate: Beginner + intermediate
- Advanced: All levels

Stage 3: Workout Split Selection

3-Day Split (Intermediate):

- Monday: Push (Chest, Shoulders, Triceps)
- Wednesday: Pull (Back, Biceps, Forearms)
- Friday: Legs & Core (Quads, Hamstrings, Glutes, Abs)

5-Day Split (Advanced):

- Monday: Chest
- Tuesday: Back
- Wednesday: Legs
- Thursday: Shoulders & Arms
- Saturday: Full Body & Conditioning

Stage 4: Exercise Selection Algorithm

For each workout day:

1. **Compound Movements First** (1-2 exercises)
   - Multi-joint exercises (e.g., chest press, rows, squats)
   - Scored by muscle group match and goal alignment
   - Highest priority for strength/muscle gain goals
2. **Isolation/Accessory Movements** (2-4 exercises)
   - Single-joint exercises (e.g., bicep curls, lateral raises)
   - Target specific muscles not fully covered
   - Variety across weeks

Scoring System:

```python
score = 100
+ 20 per matching muscle group
+ 30 if compound movement
+ 20 if compound + strength goal
+ 15 if cardio + weight loss goal
+ 5 if bodyweight (accessibility)
- 100 if already selected this week
± 5 random variance
```

Stage 5: Programming Calculation

For each exercise, determine:

Sets:

- Beginner: 3 sets
- Intermediate: 3 sets
- Advanced: 4 sets

Reps (based on goal):

- Strength: 6 reps
- Muscle Gain: 10 reps
- Weight Loss: 15 reps
- Maintenance: 12 reps

Rest Periods:

- Beginner: 90 seconds
- Intermediate: 60 seconds
- Advanced: 60 seconds

Stage 6: Calorie Calculation

Formula:

```
Time per exercise = (reps × 3 seconds + rest) × sets / 60
Calories = calorie_rate × time_minutes
```

Calorie Rates (per minute):

- Compound movements: 5-7 cal/min (level dependent)
- Isolation movements: 3.5-5.5 cal/min
- Uses exercise-specific rates from database when available

Weekly Total:

- Sum of all workout day calories
- Used to adjust nutrition plan

🔧 Configuration

Adjusting Workout Frequency

Edit `determine_workout_frequency()` in `workout_generator.py`:

```python
def determine_workout_frequency(self, fitness_level: str, fitness_goal: str) -> int:
    frequency_map = {
        'beginner': 3,      # Change to 2 for less frequent
        'intermediate': 4,  # Change to 3 or 5
        'advanced': 5
    }
    return frequency_map[fitness_level]
```

Customizing Workout Splits

Edit `get_workout_split()` to add new splits or modify existing ones:

```python
splits = {
    4: {  # Add 4-day split
        'intermediate': [
            {'day': 'Monday', 'focus': 'Upper', 'muscle_groups': [...]},
            {'day': 'Tuesday', 'focus': 'Lower', 'muscle_groups': [...]},
            {'day': 'Thursday', 'focus': 'Upper', 'muscle_groups': [...]},
            {'day': 'Friday', 'focus': 'Lower', 'muscle_groups': [...]}
        ]
    }
}
```

Adjusting Exercise Selection

Modify the scoring in `score_exercise()`:

```python
# Increase compound movement priority
if exercise['mechanic'] == 'compound':
    score += 50  # Was 30

# Add preference for certain equipment
if exercise['equipment'] == 'dumbbell':
    score += 10  # Prefer dumbbells
```

📤 Output Format

JSON Structure

```json
```

```
{
  "user_id": 1,
  "week_of": "2025-03-18",
  "fitness_level": "intermediate",
  "workout_days_per_week": 3,
  "total_weekly_calories": 1247.5,
  "days": [
    {
      "day": "Monday",
      "focus": "Push",
      "target_muscles": ["chest", "shoulders", "triceps"],
      "duration_minutes": 42.3,
      "estimated_calories": 415.2,
      "exercises": [
        {
          "order": 1,
          "id": "Dumbbell_Bench_Press",
          "name": "Dumbbell Bench Press",
          "sets": 3,
          "reps": 10,
          "rest_seconds": 60,
          "estimated_time_min": 8.5,
          "estimated_calories": 51.0,
          "instructions": ["Step 1...", "Step 2..."],
          "primary_muscles": ["chest", "triceps"],
          "equipment": "dumbbell",
          "images": ["Dumbbell_Bench_Press/0.jpg"]
        }
        // ... more exercises
      ]
    },
    // ... more days
    {
      "day": "Tuesday",
      "focus": "Rest Day",
      "duration_minutes": 0,
      "estimated_calories": 0,
      "description": "Active recovery recommended: light walking, stretching, or yoga."
    }
  ]
}
```

🧪 Testing Checklist

Unit Tests

- [ ] Exercise filtering works for all limitation types

- [ ] Equipment filtering includes bodyweight + selected equipment
- [ ] Fitness level correctly determined from activity level
- [ ] Workout splits return correct number of days

Integration Tests

- [ ] Generator works with real user data from database
- [ ] Generated plans save to database correctly
- [ ] Dashboard displays workout plans properly
- [ ] Exercise images and instructions load

User Acceptance Tests

- [ ] Beginner user gets appropriate difficulty exercises
- [ ] User with back problems gets safe exercises
- [ ] User with limited equipment gets viable workouts
- [ ] Weekly calorie burn is reasonable (not excessive)

🐛 Troubleshooting

Issue: "No eligible exercises found"

Cause: Too many constraints (limitations + equipment)

Solutions:

1. Check if user limitations are correctly mapped to database categories
2. Verify equipment names match database values
3. Check if exercises.db has sufficient exercise variety

```python
# Debug: Print eligible exercises count
eligible = generator.get_eligible_exercises(user_profile, fitness_level)
print(f"Found {len(eligible)} eligible exercises")
```

Issue: Workouts too short/long

Cause: Programming data missing or calorie rates off

Solutions:

1. Verify exercise_programming table has data
2. Adjust default values in `calculate_programming()`
3. Modify calorie rates in `calculate_exercise_calories()`

Issue: Duplicate exercises across days

Cause: Not enough exercise variety or scoring issue

Solutions:

1. Ensure `already_selected` list is passed correctly
2. Increase exercise database size
3. Adjust scoring to penalize recent selections more

Issue: Muscle group imbalance

Cause: Workout split muscle coverage insufficient

Solutions:

1. Review split definitions in `get_workout_split()`
2. Add more muscle groups to each day's `muscle_groups` list
3. Adjust exercise selection to prioritize underworked muscles

---

📈 Future Enhancements

Progressive Overload

Track user's workout history and gradually increase:

- Weight recommendations
- Rep ranges
- Exercise difficulty

```python
def adjust_for_progression(self, user_id, current_plan):
    # Get last 4 weeks of workouts
    # Increase by 5% if user completing consistently
    # Suggest next progression tier
```

Exercise Substitution

Allow users to swap exercises they don't like:

```python
```

```
def find_exercise_alternative(self, exercise_id, constraints):
    # Find exercises with similar:
    # - Primary muscles
    # - Difficulty level
    # - Equipment
    # Return top 3 alternatives
```

Periodization

Implement training phases (strength → hypertrophy → endurance):

```python
def generate_periodized_plan(self, user_id, weeks=12):
    # Week 1-4: Strength (low reps, high weight)
    # Week 5-8: Hypertrophy (moderate reps)
    # Week 9-12: Endurance (high reps, lower weight)
```

Workout Templates

Pre-designed programs for specific goals:

```python
TEMPLATES = {
    'beginner_weight_loss': {...},
    'intermediate_muscle_building': {...},
    'advanced_powerlifting': {...}
}
```

📚 API Reference

WorkoutGenerator Class

`__init__(fitplan_db, exercise_db)`

Initialize generator with database paths.

`generate_weekly_plan(user_id: int) -> Dict`

Main method. Generates complete weekly workout plan.

Returns:

```python
```

```
{
    'user_id': int,
    'week_of': str,
    'fitness_level': str,
    'workout_days_per_week': int,
    'total_weekly_calories': float,
    'days': List[Dict]
}
```

`get_user_profile(user_id: int) -> Dict`

Fetch and parse user data from fitplan.db.

`determine_fitness_level(activity_level: str, age: int) -> str`

Calculate fitness level from activity and age.

Returns: 'beginner', 'intermediate', or 'advanced'

`get_eligible_exercises(user_profile: Dict, fitness_level: str) -> List[Dict]`

Get all exercises passing safety and equipment filters.

`select_exercises_for_day(eligible_exercises, day_info, fitness_level, fitness_goal, already_selected) -> List[Dict]`

Select 4-6 exercises for a single workout day.

🤝 Contributing

To extend the workout generator:

1. **Add New Workout Splits:** Edit `get_workout_split()`
2. **Modify Scoring Logic:** Edit `score_exercise()`
3. **Add New Filters:** Create new filter methods
4. **Adjust Programming:** Edit `calculate_programming()`

📞 Support

For issues or questions:

1. Check the troubleshooting section
2. Review test output for errors
3. Verify database structure matches README.md
4. Test with simplified user profiles (fewer constraints)

✅ Implementation Checklist

- [ ] Download all provided files (workout_generator.py, app_integration.py, test_workout_generator.py)
- [ ] Verify exercises.db has required tables and data
- [ ] Run test suite: `python test_workout_generator.py`
- [ ] Review generated JSON files for quality
- [ ] Add generator import to app.py
- [ ] Replace placeholder endpoints with real implementation
- [ ] Test onboarding → plan generation → dashboard flow
- [ ] Verify workout plans display correctly in UI
- [ ] Test with various user profiles (beginner, advanced, limitations)
- [ ] Validate calorie calculations
- [ ] Deploy and monitor for errors

Version: 1.0

Last Updated: 2025-10-06

Compatible With: FitPlan Flask App v2.0