Progress Tracking & Monthly Reports System

Overview

Complete implementation of user progress tracking and monthly report generation system, integrating frontend UI components with backend data persistence and analytics.

Backend Implementation

1. Progress Logging API (`/api/progress`)

GET `/api/progress`

- Returns user's progress logs (last 30 by default)
- Query params: `?limit=N` to control number of logs returned
- Response: Array of ProgressLog documents

POST `/api/progress/workout`

- Logs a completed workout
- Request body:

```
{
  "day": "Day 1",
  "completedExercises": 8,
  "totalExercises": 10,
  "durationSec": 3600
}
```

- Creates ProgressLog entry with type 'workout'

POST `/api/progress/meal`

- Logs a completed meal
- Request body:

```
{
  "mealName": "Breakfast",
  "calories": 450,
  "macros": {
    "p": 30,
    "c": 50,
    "f": 15
  }
}
```

- Creates ProgressLog entry with type 'meal'

GET `/api/progress/stats`

- Returns aggregated progress statistics
- Query params: `?days=30` (default)
- Response:

```
{
  "workoutsCompleted": 12,
  "totalMealsLogged": 45,
  "activeDays": 15,
  "currentStreak": 3
}
```

- Aggregates from ProgressLog collection
- Calculates consecutive active days for streak

2. Monthly Reports API (`/api/reports`)

GET `/api/reports/workout/monthly/:year/:month`

- Auto-generates or retrieves cached workout report for specified month
- Calculates from ProgressLog (workout entries) and WorkoutPlan data
- Response:

```
{
  "userId": "...",
  "year": 2024,
  "month": 1,
  "completedWorkouts": 18,
  "totalWorkouts": 20,
  "adherenceScore": 90,
  "avgDuration": 3240,
  "generatedAt": "2024-01-31T..."
}
```

- Adherence score: (completedWorkouts / totalWorkouts) * 100
- Caches in MonthlyWorkoutReport collection

GET `/api/reports/diet/monthly/:year/:month`

- Auto-generates or retrieves cached diet report for specified month
- Calculates from ProgressLog (meal entries) and DietPlan data
- Response:

```
  {
    "userId": "...",
    "year": 2024,
    "month": 1,
    "daysLogged": 28,
    "totalDays": 31,
    "adherenceScore": 90.3,
    "avgDailyCalories": 2100,
    "avgMacros": {
      "protein": 150,
      "carbs": 200,
      "fats": 70
    },
    "generatedAt": "2024-01-31T..."
  }
```

- Adherence score: (daysLogged / totalDays) * 100
- Averages calculated from logged meals
- Caches in MonthlyDietReport collection

POST `/api/reports/generate`

- Force-regenerates current month's reports
- Deletes cached reports and recalculates from raw progress logs
- Use for manual refresh or when data corrections are made

Database Models

ProgressLog

```
{
  userId: ObjectId,
  date: Date,
  type: 'workout' | 'meal',
  // Workout fields
  workout: {
    day: string,
    completedExercises: number,
    totalExercises: number,
    durationSec: number
  },
  // Meal fields
  meal: {
    name: string,
    calories: number,
    macros: { p: number, c: number, f: number }
  }
}
```

MonthlyWorkoutReport

```
{
  userId: ObjectId,
  year: number,
  month: number,
  completedWorkouts: number,
  totalWorkouts: number,
  adherenceScore: number,
  avgDuration: number,
  generatedAt: Date
}
```

MonthlyDietReport

```
{
  userId: ObjectId,
  year: number,
  month: number,
  daysLogged: number,
  totalDays: number,
  adherenceScore: number,
  avgDailyCalories: number,
  avgMacros: {
    protein: number,
    carbs: number,
    fats: number
  },
  generatedAt: Date
}
```

Frontend Implementation

1. Progress Hook (`useUserProgress`)

```
const {
  stats,           // Current progress stats
  logs,            // Recent progress logs
  loading,
  error,
  logWorkout,      // Log workout completion
  logMeal,         // Log meal completion
  refresh          // Manual refresh
} = useUserProgress();

// Log workout: (day, completedExercises, totalExercises, durationSec?)
await logWorkout('Day 1', 8, 10, 3600);
```

```
// Log meal: (mealName, calories?, macros?)
await logMeal('Breakfast', 450, { p: 30, c: 50, f: 15 });
```

2. Progress Page (`/progress`)

Features:

- Current streak display with fire emoji
- Workouts completed counter
- Meals logged counter
- Active days tracker
- Recent activity summary
- Link to monthly reports

Data Source: `/api/progress/stats`

3. Monthly Reports

Reports Landing Page (`/reports`)

- Lists last 6 months
- Links to both workout and diet reports for each month
- Month/year navigation

Workout Report Page (`/reports/workout/[year]/[month]`)

Displays:

- Completed workouts count
- Adherence score with color-coded badge
- Average workout duration (formatted as HH:MM:SS)
- Performance insights based on adherence:
  - 90%+: "Excellent consistency! Keep up the great work!"
  - 70-89%: "Good progress! Try to maintain consistency."
  - 50-69%: "You're on track. Push for more consistency next month."
  - <50%: "Let's work on building a consistent routine."

Diet Report Page (`/reports/diet/[year]/[month]`)

Displays:

- Days logged count
- Adherence score with color-coded badge
- Average daily calories
- Average macros (protein, carbs, fats)
- Performance insights based on adherence:
  - 90%+: "Outstanding nutrition tracking!"
  - 70-89%: "Great job tracking your meals!"

    - 50-69%: "Keep improving your tracking consistency."
    - <50%: "Consider tracking more consistently for better results."

4. Component Updates

WorkoutTimer

- Uses `useUserProgress` hook
- Logs workout on completion with:
    - Day label (e.g., "Day 1")
    - Completed exercises count
    - Total exercises count
    - Duration in seconds
- Props: `exercises`, `onComplete`, `onStop`, `workoutId`, `day`

MealCard

- Uses `useUserProgress` hook
- Logs meal on "Log Meal" button click with:
    - Meal name
    - Calories (optional)
    - Macros object (p/c/f format)
- Props: `mealName`, `time`, `calories`, `foods`, `macros`, `onLog`, `mealId`

Data Flow

Workout Completion Flow

1. User completes workout in WorkoutTimer component
2. Component calls `logWorkout(day, completedExercises, totalExercises, durationSec)`
3. Hook sends POST to `/api/progress/workout`
4. Backend creates ProgressLog entry
5. Progress stats automatically include new workout
6. Monthly report includes workout in next generation

Meal Logging Flow

1. User clicks "Log Meal" in MealCard component
2. Component calls `logMeal(mealName, calories, macros)`
3. Hook sends POST to `/api/progress/meal`
4. Backend creates ProgressLog entry
5. Progress stats automatically include new meal
6. Monthly report includes meal in next generation

Monthly Report Generation Flow

1. User visits `/reports/workout/2024/1` or `/reports/diet/2024/1`
2. Frontend calls GET `/api/reports/workout/monthly/2024/1` (or diet)

3. Backend checks for cached report in MonthlyWorkoutReport/MonthlyDietReport
4. If not cached or outdated:
    - Aggregates ProgressLog entries for that month
    - Calculates adherence scores and averages
    - Saves to cache collection
5. Returns report data to frontend
6. Frontend displays with color-coded insights

Testing

Manual Testing Checklist

- [ ] Complete a workout and verify ProgressLog entry created
- [ ] Log a meal and verify ProgressLog entry created
- [ ] Check `/progress` page displays correct stats
- [ ] Verify streak calculation with consecutive active days
- [ ] View monthly workout report for current month
- [ ] View monthly diet report for current month
- [ ] Verify adherence scores calculate correctly
- [ ] Test report insights display for different adherence levels
- [ ] Force regenerate report with POST `/api/reports/generate`
- [ ] Verify cached reports are reused (check generatedAt timestamp)

Backend Build Status

 PASSED - TypeScript compilation successful with no errors

Frontend Build Status

 PASSED - Next.js build successful

- All pages compile without errors
- Report pages generated as dynamic routes
- WorkoutTimer and MealCard components updated

Architecture Benefits

1. **Data Persistence**: All progress now stored in database, not in-memory
2. **Performance**: Monthly reports cached to avoid repeated aggregations
3. **Scalability**: Can handle large numbers of progress logs efficiently
4. **Analytics**: Easy to add new metrics by aggregating ProgressLog data
5. **User Experience**: Real-time stats update as users log activities
6. **Insights**: Adherence scores provide actionable feedback
7. **Consistency**: Single source of truth for progress data

Future Enhancements

1. **Delete Progress Logs**: Add DELETE endpoint for incorrect entries
2. **Edit Progress Logs**: Add PATCH endpoint for corrections

3. **Export Reports**: PDF/CSV export functionality
4. **Trend Analysis**: Multi-month comparison graphs
5. **Goals & Targets**: Set and track progress towards specific goals
6. **Notifications**: Reminders for logging meals/workouts
7. **Leaderboards**: Compare progress with other users (optional)
8. **Exercise-Level Tracking**: Track individual exercise completion within workouts
9. **Meal Photos**: Upload photos with meal logs
10. **Integration**: Connect with fitness trackers (Apple Health, Google Fit)

API Authentication

All endpoints require JWT authentication via `Authorization: Bearer <token>` header. The middleware extracts `req.user.userId` from the token and scopes all queries to the authenticated user.

Error Handling

- Invalid date ranges return 400 Bad Request
- Missing required fields return 400 Bad Request
- Unauthorized requests return 401 Unauthorized
- Server errors return 500 Internal Server Error with descriptive message