Python Term Project – Personal Fitness Tracking Application

1. Project Overview

- **Goal:** Create a terminal application that tracks workouts, nutrition, and wellness metrics with personalized analytics.
- **Scenario:** A personal trainer wants a simple offline tool to log daily activity and generate progress summaries for clients.
- **Core Skills:** data collection, time-series analysis, modular architecture, file persistence, optional OOP.

2. Learning Outcomes

- Model workouts, nutrition logs, and biometric measurements.
- Implement scheduling, goal tracking, and trend analysis.
- Persist user data to JSON/CSV with backup options.
- Provide insights via calculated metrics (calorie balance, PRs).
- Design intuitive CLI menus with validation and helpful prompts.

3. Deliverables

1. `README.md` detailing setup, data entry workflows, and report generation.
2. Python modules:
    - `main.py` – application entry point and menu system.
    - `profiles.py` – user profile management and authentication.
    - `workouts.py` – workout logging and analytics.
    - `nutrition.py` – meal logging and calorie tracking.
    - `metrics.py` – weight, measurements, goals, and trend analysis.
    - `storage.py` – persistence, backups, and schema validation.
3. Data files (`data/users.json`, `data/workouts.json`, `data/nutrition.json`, `data/metrics.json`).
4. Automated tests covering workout logging, goal progress calculation, and input validation.

4. Functional Requirements

4.1 User Profiles & Goals

- Create profiles with demographic info (age, height, weight), activity level, and goals (weight loss, muscle gain, endurance).
- Support login/logout and client switching during a session.
- Allow goal adjustments and track start/end dates.

Required Functions (`profiles.py`):

```
```

```
def load_users(path: str) -> list: ...
def save_users(path: str, users: list) -> None: ...
def register_user(users: list, profile: dict) -> dict: ...
def authenticate_user(users: list, email: str, pin: str) -> dict | None:
...
def update_goal(users: list, user_id: str, goal_data: dict) -> dict: ...
```

4.2 Workout Logging

- Record workouts with date, type (strength/cardio/flexibility), duration, exercises, sets, reps, weights, and notes.
- Flag personal records (heaviest lift, fastest time).
- Provide weekly summaries of total workouts, time spent, and intensity.

Required Functions (`workouts.py`):

```
def log_workout(workouts: list, workout_data: dict) -> dict: ...
def update_workout(workouts: list, workout_id: str, updates: dict) ->
dict: ...
def delete_workout(workouts: list, workout_id: str) -> bool: ...
def weekly_workout_summary(workouts: list, user_id: str, week_start: str)
-> dict: ...
def personal_records(workouts: list, user_id: str) -> dict: ...
```

4.3 Nutrition Tracking

- Log meals with timestamp, food items, macronutrients, calories, and meal type.
- Calculate daily caloric intake and compare against goals.
- Support importing meal data from CSV (optional).

Required Functions (`nutrition.py`):

```
def log_meal(meals: list, meal_data: dict) -> dict: ...
def update_meal(meals: list, meal_id: str, updates: dict) -> dict: ...
def delete_meal(meals: list, meal_id: str) -> bool: ...
def daily_calorie_summary(meals: list, user_id: str, date: str) -> dict:
...
def macro_breakdown(meals: list, user_id: str, date_range: tuple[str,
str]) -> dict: ...
```

4.4 Metrics & Analytics

- Track weight, body measurements, sleep hours, water intake, and mood.
- Visualize progress via textual charts (e.g., ASCII sparkline) or summary tables.
- Calculate trends such as 7-day moving average for weight or calorie balance.
- Provide goal progress updates and projected completion dates.

Required Functions (`metrics.py`):

```
def log_metric(metrics: list, metric_data: dict) -> dict: ...
def metrics_summary(metrics: list, user_id: str, metric_type: str, period:
tuple[str, str]) -> dict: ...
def goal_progress(users: list, metrics: list, user_id: str) -> dict: ...
def generate_ascii_chart(values: list[float]) -> str: ...
```

4.5 Persistence & Backups

- Save separate files for users, workouts, nutrition, and metrics.
- Perform backups before overwriting files; store in `backups/`.
- Offer restore option in case of corruption.

Required Functions (`storage.py`):

```
def load_state(base_dir: str) -> tuple[list, list, list, list]: ...
def save_state(base_dir: str, users: list, workouts: list, meals: list,
metrics: list) -> None: ...
def backup_state(base_dir: str, backup_dir: str) -> list[str]: ...
def validate_workout_entry(entry: dict) -> bool: ...
```

5. User Experience Requirements

- Dashboard summarizing today's workouts, calories, and goal progress.
- Reminders for missing logs (e.g., “No workout recorded for 3 days”).
- Confirm deletions and highlight unsaved changes.
- Provide help commands at each menu.

6. Validation & Business Rules

- Enforce valid date/time formats and numeric ranges (e.g., weight > 0).
- Prevent duplicate entries with same timestamp and type.
- Ensure meals and workouts cannot be logged for future dates unless explicitly allowed via setting.
- Handle invalid menu choices gracefully.

7. Testing Expectations

- Tests for caloric balance calculations, personal record detection, and goal progress.
- Provide sample datasets for deterministic testing.

8. Suggested Timeline

1. Week 1 – Profile management and data storage setup.
2. Week 2 – Workout logging and summaries.
3. Week 3 – Nutrition tracking and analytics.
4. Week 4 – Metrics, dashboards, and backups.

5. Week 5 – Testing, docs, final tuning.

9. Grading Rubric (100 pts)

- Workout & Nutrition Logging: 30 pts
- Metrics & Goal Tracking: 20 pts
- Data Persistence & Recovery: 15 pts
- Analytics & Dashboards: 15 pts
- Validation & Error Handling: 10 pts
- Documentation & UX: 5 pts
- Testing & Code Quality: 5 pts

10. Submission Checklist

- [ ] Sample profile with workout and nutrition data included.
- [ ] README demonstrates logging workflow and reports.
- [ ] Automated tests included and passing.
- [ ] Weekly commits show incremental updates.