ChatGPT

Weekly Routine Plan DSL & Consistency Tracking Specification

Overview

We propose a system for users to plan and track a weekly routine, focusing on health and habit consistency. The core is a **Domain-Specific Language (DSL)** (structured JSON format) that defines a **one-week plan** of daily activities (meals, workouts, habits, etc.), along with a framework for **consistency scoring** and progress tracking over time. Users will create a weekly plan (the plan repeats every week to form a monthly routine) and then log their daily adherence. Key features include:

- A **Weekly Plan DSL** that captures a fixed schedule of “slots” (times of day with activity categories) as a skeleton, and specific **daily variant details** for each slot on each day of the week.
- **Immutable Plans** once finalized: users can clone and modify into new plans but the original plan remains the baseline for consistency comparisons.
- **Daily Tracking** of completed activities with ability to **swap** equivalent activities between days and add occasional extra items (without altering the plan).
- A **Category-based weighting** system for each slot to quantify its importance (e.g. exercise slots have higher weight than snack slots) for consistency scoring.
- A **Consistency Score** computed daily (aiming for 100% each day) based on completed activities vs. plan, with **penalties** for negative habits (e.g. cheat meals) and minor allowances for extra positive activities (capped at a maximum of 100%).
- A **Momentum-based long-term consistency metric** (using a rolling average or exponential smoothing) to reflect habit strength over time – rewarding sustained adherence and forgiving occasional lapses [1].
- **Streak tracking** for perfect days and individual categories (e.g. exercise streak, diet streak), and analytics to show patterns and trends (calendar heatmaps, slot-wise adherence trends).
- Support for **offline use** (local caching of plan and logs) with sync to a dedicated backend for persistence and analytics [2].
- **Security and Privacy** considerations (including HIPAA compliance for any personal health data, e.g. encryption in transit and at rest [3]).

The following sections detail the DSL structure, plan lifecycle, tracking and scoring algorithms, and data structures for analytics and visualization.

Weekly Routine Plan DSL

The Routine Plan DSL defines the user's weekly routine in a predictable JSON structure. It consists of two parts: a **skeleton** (common daily schedule structure) and **variants** (day-specific content for each slot in the skeleton). This separation ensures the routine's **time schedule (slots)** is consistent every day of the week, while allowing the actual activities/foods in those slots to vary each day.

- **Skeleton Structure (Weekly Template):** Defines the sequence of slots in a day, with their time and metadata. This is the "skeleton" of the routine repeated every day. For example, a user's skeleton might include slots like: *On Waking Up (06:00)*, *Other* (06:30), *Workout* (07:00), *During Workout* (07:30), *Post Workout* (08:30), *Before Lunch* (12:00), *Lunch* (13:30), *Evening Snack* (17:00), *Cardio* (19:00), *Dinner* (20:00), *Post Dinner* (20:30) [4] [5] . The skeleton is consistent across all days (same slots in same order and times each day). Each slot in the skeleton includes:
- `slotId` (or name): e.g. `"On Waking Up"`, `"Lunch"`.
- `time`: e.g. `"06:00"`, `"13:30"`.
- `category`: one of predefined categories like `food`, `exercise`, `habit`, `hydration`, `sleep`, or `other`.
- `baseWeight`: numeric weight/priority for this slot's importance (based on category defaults or user-defined for "other" slots).

- (Optional) `enabledByDefault`: a boolean indicating if this slot is normally active. If a slot is not part of a particular day's routine (e.g. rest day for workout), it can be marked disabled in that day's variant (overriding the default).

- **Variants Structure (Daily Activities):** Defines the specific activities or items for each slot for each day of the week (7 variants, e.g. Monday through Sunday). Each day's entry is essentially a list of slots mirroring the skeleton (same order and slots), but providing day-specific details:

- `day`: e.g. `"Monday"`, `"Tuesday"`, etc.
- For each slot in the skeleton:
  - `slotId` or `name` (to link back to skeleton slot).
  - `enabled`: boolean if this slot is active on this day (if false, this slot is effectively "off" on that day, e.g. no workout on rest days).
  - `items`: an **array of activities/items** to be done in that slot. Each item can include:
  - `name`: description of the activity or food item (e.g. `"Soaked Fenugreek Seed Water"`, `"Weight Lifting Session"`).
  - Additional key-value details as needed (e.g. `quantity` and `unit` for foods, `duration` for exercise, any `notes` or nutritional info, etc.). The DSL should be flexible to include varied details per item.
  - (Optional) `notes`: any special notes for that slot on that day (if not captured at item level).
  - (Optional) `expectedValue`: for meal slots, maybe calorie target, or for workout maybe target minutes – not mandatory but can be included for reference.

Example DSL (JSON):

```
{
  "skeleton": [
    { "id": 1, "name": "On Waking Up", "time": "06:00", "category": "habit",
"baseWeight": 0.5 },
    { "id": 2, "name": "Other", "time": "06:30", "category": "other",
"baseWeight": 1.0 },
    { "id": 3, "name": "Workout", "time": "07:00", "category": "exercise",
"baseWeight": 5 },
    { "id": 4, "name": "During Workout", "time": "07:30", "category":
"hydration", "baseWeight": 0.5 },
    { "id": 5, "name": "Post Workout", "time": "08:30", "category": "food",
"baseWeight": 1 },
    { "id": 6, "name": "Lunch", "time": "13:30", "category": "food",
"baseWeight": 3 },
    { "id": 7, "name": "Evening Snack", "time": "17:00", "category": "food",
"baseWeight": 1 },
    { "id": 8, "name": "Cardio", "time": "19:00", "category": "exercise",
"baseWeight": 5 },
    { "id": 9, "name": "Dinner", "time": "20:00", "category": "food",
"baseWeight": 3 },
    { "id": 10, "name": "Post Dinner", "time": "20:30", "category": "habit",
"baseWeight": 0.5 }
  ],
  "variants": {
    "Monday": [
      { "slotId": 1, "enabled": true, "items": [
          { "name": "Soaked Fenugreek Seed Water", "quantity": 200, "unit":
"ml" }
      ]},
      { "slotId": 2, "enabled": true, "items": [
          { "name": "Aloe Vera Juice", "quantity": 15, "unit": "ml" },
          { "name": "Water", "quantity": 200, "unit": "ml" },
          { "name": "Macadamia Nuts", "quantity": 2, "unit": "pcs" }
      ]},
      { "slotId": 3, "enabled": true, "items": [
          { "name": "Strength Training Workout", "duration": 45, "unit": "min",
"notes": "Focus: Upper body" }
      ]},
      { "slotId": 4, "enabled": true, "items": [
          { "name": "Water during workout", "quantity": 500, "unit": "ml" }
      ]},
      { "slotId": 5, "enabled": true, "items": [
          { "name": "Post-workout Smoothie", "quantity": 250, "unit": "ml",
"notes": "Protein + Carbs" }
      ]},
      { "slotId": 6, "enabled": true, "items": [
          { "name": "Grilled Chicken Salad", "quantity": 1, "unit": "bowl",
"calories": 400 }
      ]},
      { "slotId": 7, "enabled": true, "items": [
          { "name": "Greek Yogurt with Berries", "quantity": 1, "unit": "cup" }
      ]},
      { "slotId": 8, "enabled": true, "items": [
          { "name": "Evening Jog", "duration": 30, "unit": "min" }
      ]},
      { "slotId": 9, "enabled": true, "items": [
          { "name": "Steamed Fish and Vegetables", "quantity": 1, "unit":
"plate" }
      ]},
      { "slotId": 10, "enabled": true, "items": [
          { "name": "Herbal Tea", "quantity": 1, "unit": "cup" }
      ]}
    ],
    "Tuesday": [
      { "slotId": 1, "enabled": true, "items": [
          { "name": "Cucumber Detox Water", "quantity": 200, "unit": "ml" }
      ]},
      { "slotId": 2, "enabled": true, "items": [
          { "name": "Apple Cider Vinegar Tablet", "quantity": 1, "unit":
"tablet" },
          { "name": "Water", "quantity": 200, "unit": "ml" },
          { "name": "Almonds", "quantity": 5, "unit": "pcs" }
      ]},
      { "slotId": 3, "enabled": true, "items": [
          { "name": "Yoga Session", "duration": 60, "unit": "min" }
      ]},
      { "slotId": 4, "enabled": true, "items": [
          { "name": "Water during workout", "quantity": 500, "unit": "ml" }
      ]},
      { "slotId": 5, "enabled": true, "items": [
          { "name": "Post-workout Fruit Bowl", "quantity": 1, "unit": "bowl" }
      ]},
      { "slotId": 6, "enabled": true, "items": [
          { "name": "Paneer Stir Fry with Vegetables", "quantity": 1, "unit":
"plate" }
      ]},
      { "slotId": 7, "enabled": true, "items": [
          { "name": "Mixed Nuts and Seeds", "quantity": 1, "unit": "handful" }
      ]},
      { "slotId": 8, "enabled": false, "items": [] },  <!-- Cardio off on
Tuesday -->
      { "slotId": 9, "enabled": true, "items": [
          { "name": "Chicken Soup with Quinoa", "quantity": 1, "unit": "bowl" }
      ]},
      { "slotId": 10, "enabled": true, "items": [
          { "name": "Meditation", "duration": 15, "unit": "min" }
      ]}
    ],
    "...": [ ... ],
    "Sunday": [ ... ]
  }
}
```

(The JSON above illustrates the structure: Monday and Tuesday filled in, Sunday and others truncated for brevity.)

Notes on DSL usage:

- The DSL explicitly lists 7 days; each day must have exactly the same slots (in same order) as the skeleton. This ensures consistency in schedule.
- If a slot is not used on a certain day, set `"enabled": false` (and items can be an empty list) for that day's entry (e.g., `"Cardio"` disabled on Tuesday in the example).
- The DSL is the **source of truth** for the plan. It can be easily generated by an LLM or UI form and stored as JSON. Once the plan is finalized, this JSON is saved on the server and remains unchanged (read-only) as the reference plan.
- The DSL captures all needed detail to present a user-friendly schedule (which the app can render in a nice UI for user approval before finalizing).

Plan Creation and Lifecycle

Creating a Weekly Plan: Users will build a plan covering all 7 days. This involves selecting a skeleton schedule (either starting from a template or defining custom slots and times) and then specifying the activities for each slot on each day (the variants). The system should allow users to add or remove slots *before* finalizing the plan: - Users can introduce additional routine slots (e.g., add a new habit like "Evening Meditation" at 9:30 PM) or remove slots they don't want, as long as the final structure is consistent across the week. - Category and weight for any new slot must be set at creation (with defaults based on category or user input for custom categories).

Finalizing the Plan: Once the week's plan is complete and reviewed, the user finalizes it. A finalized plan becomes locked for editing to preserve a stable reference for consistency tracking. If the user wants to make changes later (e.g., change meal plans or slot times), the app will require creating a **new plan (clone and edit)**: - The user can clone the current plan, apply adjustments, and save as a new plan version. This new plan, once finalized, can replace the old plan for future weeks or months, while the original remains in history. - This versioning ensures past consistency data can be tied to the exact plan followed, and avoids altering history.

Week vs. Month Plans: The base planning unit is one week. A **"routine"** can be considered a month-long schedule, which in most cases will be the same weekly plan repeated 4 (or 4½) times. Users can choose to use the same weekly plan every week of the month (common routine) or define multiple week plans and sequence them (e.g., two distinct week plans alternating, or a four-week cycle if more variety is desired). For simplicity: - If a user defines one week plan for a month, the monthly routine is just that plan repeated each week. - If multiple distinct week plans are defined (say Plan A and Plan B), the user can specify an order (e.g., A, then B, then A, then B for 4 weeks) or any combination. The DSL and system currently focus on single-week definition; mixing weeks could be handled by linking multiple week-DSLs in sequence when generating the monthly view. - In all cases, consistency scoring will treat each week's plan the same way – there's always a defined set of slots each day to aim for.

Using the Plan Daily: Each day, the user will attempt to follow the plan. The app will present that day's slots (with the planned items/activities from the variant for that weekday) and allow the user to mark completion. Some runtime flexibility is provided without altering the saved plan: - **Swapping Variants Across Days:** Users might occasionally swap a planned activity with another day's. For example, if Tuesday's lunch was

“Paneer Stir Fry” and Wednesday’s lunch was “Grilled Fish”, and the user decides to eat the fish on Tuesday instead, they can swap the **Lunch slot** between Tuesday and Wednesday. The app should provide a swap function that allows swapping the *items* (or entire slot content) of the same slot between two days of the current week. This is restricted to the same slot category (you can only swap lunch with lunch, workout with workout, etc., not lunch with dinner). Swapping **does not change the underlying plan JSON**; it only affects the *actual log* of what was done on each day. - Swapping is intended to still fulfill the weekly plan in aggregate. By the end of the week, the user would have completed all the planned items, just on different days. This ensures variety/nutrition goals can still be met over the week even if not on the originally assigned day. - The system should track swaps in the weekly log for analytics (so we know if a user frequently switches certain meals to different days, etc.), but for consistency scoring, a completed slot (even with swapped content) counts as completed. In other words, as long as the user did *something* in that slot (either the planned item or a swapped equivalent), the slot is marked done for that day. - **One-off Additions:** Users can add extra activities on the fly in a given day. For instance, if the user did an additional workout session in the evening that wasn’t in the plan, or they had an extra snack (or an unscheduled cheat item), they can log these as one-off items. These do **not alter the plan** (the plan remains as originally defined), but they do affect the daily consistency calculation (extra positive activities can slightly improve the score, while negative indulgences will penalize it). One-off entries are essentially ad-hoc logged items tied to a date (and possibly to a category or slot if relevant) that exist only in the tracking log. - **Marking Completion:** The user should check off each item or slot as they complete it. The app may allow ticking off individual items (e.g., in a meal slot with multiple components, they can mark each item consumed), but for scoring we primarily consider the **slot as a whole**: - If all items in a slot are completed, the slot is “completed” fully. - If only some items are done, we consider the slot partially completed (which will yield partial points, see scoring below). For example, if a Dinner slot had 3 items and the user ate 2 out of 3, we might count 2/3 of that slot’s weight as earned. - If a slot is enabled but nothing was done (user skipped it entirely), it’s a miss (0 points for that slot). - If a slot was disabled for that day (e.g., no workout scheduled on Sunday), it doesn’t count against the user at all for that day’s consistency (its weight is excluded from that day’s total possible points). - **Editing/Adding after Finalize:** As noted, once the plan is finalized, the structure and planned items don’t change. The user can’t directly edit tomorrow’s planned meal in the plan – they would either swap with another day or just do something different and log it as a deviation. This keeps the plan as a stable template. If the user’s routine needs a fundamental change (e.g., they want to permanently remove the “Cardio” slot going forward), they should create a new plan version reflecting that change rather than editing the current plan.

Monthly Routines and Transitions: Typically, a plan will be used for multiple weeks (e.g., a whole month or more) to build consistency. At the end of a cycle (say end of month), the user might stick with the same plan for the next month if it’s working, or create a new one to incorporate progressive changes (like increased workout load or different diet). When a new week starts, any day-specific swaps from the previous week are reset – the plan goes back to its default assignment for each day. Each new week is a fresh iteration of the plan (unless a new plan is adopted).

Categories of Slots and Weightage

Each slot in the routine is assigned a **category** and a **weight** which together determine how much that slot contributes to the consistency score. The category reflects the type of activity, and the weight reflects its relative importance (or impact on the user’s goals).

Slot Categories: We define several categories of routine slots: - **Exercise** – Active exercise sessions (e.g., gym workout, cardio, yoga). *Default weight: 5* (High impact). - **Food (Main Meal)** – Major meals such as breakfast, lunch, or dinner. *Default weight: 3* (Moderate-high impact). - **Food (Snack)** – Lighter meals or snacks. *Default weight: 1* (Low impact). - **Hydration** – Water or other routine drinks (could be standalone or during workouts). *Default weight: 0.5* (Very low impact). - **Habit** – Miscellaneous positive habits (meditation, reading, supplements, etc.) that don't fall under exercise/food. *Default weight: 1* (or vary based on the habit's significance; many habits might be moderate impact). - **Sleep** – Sleep-related slots (e.g., "Bedtime by 10:30 PM"). *Default weight: 5* (sleep is crucial, similar to exercise). - **Other** – Any custom slot the user adds that doesn't fit the above. The user can assign a weight for these slots manually (within a reasonable range, say 0.5 to 5). By default, if unspecified, treat as weight 1 or based on user priority.

The default weights are chosen by the system to roughly match the impact: for example, Exercise and Sleep are often the most critical for health (hence high weight), main meals are important (but slightly less), snacks and minor habits are less critical individually.

Slot Weightage in Plan: Each slot in the skeleton will carry its weight. These weights will be used to calculate what fraction of the day's routine is completed. Some considerations: - The **total weight per day** is the sum of weights of all *enabled* slots for that day. This can vary day to day (e.g., a rest day might have fewer slots if exercise slots are disabled, thus a lower total possible weight that day). The scoring formula will normalize by the day's total, so each day is scaled to 100% if all its scheduled slots are done. - If multiple slots share a category, each carries the category's base weight (we do not automatically split weight between two exercise slots; each exercise slot can be full weight 5). This means a day with two exercise sessions could have a higher total weight (e.g., two exercise slots = 10 weight points) than a day with one exercise. That's acceptable – it implies that on two-workout days, there's more "to gain" but also more to potentially miss. The consistency score being percentage handles it (completing both yields 100% on that day, completing one of two would be 50% of that day). - **User-adjusted Priority:** For any "Other" slots or if the user feels a certain slot should count more/less for their personal goals, we may allow manual adjustment of that slot's weight. For example, if the user has a habit slot for "Study for 1 hour" and they want it to be a high priority, they might set weight 3 instead of the default 1 for a habit. - The weights and categories also help the app group and analyze performance by category (e.g., "consistency in exercise vs consistency in diet").

Example: Using the defaults above, suppose a day has: 1 exercise slot (5), 3 main meals (3 each = 9), 1 snack (1), 2 hydration slots (0.5 each =1), and 1 habit slot (0.5). The total weight = 5 + 9 + 1 + 1 + 0.5 = 16.5. Each component's percentage of the day is known (exercise ~30% of the day's points, each meal ~18%, etc.). If the user skips exercise, they lose 5 points out of 16.5 (dropping maximum achievable to ~11.5/16.5 = 70%). If they skip a snack (1 point), it's much smaller impact (would achieve ~15.5/16.5 = 94%). This illustrates how higher weight slots have bigger influence on the daily score.

Strict vs Relaxed Slider: The system will allow users to adjust an overall **consistency strictness** setting, which scales how easy or hard it is to reach 100%. This acts as a multiplier or modifier on the scoring calculation. The idea is to accommodate different personal approaches – some may want to be very strict (where missing even a small thing significantly lowers the score), others might want a more forgiving calculation. - For example, a "Strict" setting might multiply penalties by 1.2 (making negative points hit harder) and maybe require closer to full completion for a good score, whereas a "Relaxed" setting might multiply penalties by 0.8 or raise the credit of completed tasks slightly. - The exact implementation could be: **ConsistencyMultiplier (CM)** where 0.5 = very lenient, 1.0 = normal, 1.5 = very strict. We can apply this in the scoring formula by adjusting the effective weights or penalties. *E.g.*, under strict (1.5), missing a slot of weight 3 might effectively count as 4.5 points “lost” (so your percentage drops more), whereas under relaxed (0.8) missing the same slot counts as only 2.4 points lost. - Alternatively, the slider could adjust the threshold for a “perfect day” streak (e.g., 90% vs 80%) rather than the raw score. The exact method can be tuned, but the DSL or user profile should store this preference (it’s user-wide, not per slot). - By default, we assume a normal setting (CM = 1).

Consistency Scoring Mechanism

Daily Consistency Score: Each day, as the user completes or misses activities, a consistency score for that day is calculated (0% to 100%). The goal is to reach 100% by end of day, meaning fully adherent to the plan. The calculation is as follows: 1. **Calculate Completed Weight:** Sum the weights of all slots that were completed. If a slot is fully done (all items done), add the full slot weight. If partially done, add a fraction (for example, if a slot weight is 2 and the user did half the items, you might add ~1, though for simplicity we could also decide that a slot is only counted if fully done – but per user’s request, partial credit is desired). 2. **Calculate Missed Weight:** Determine the weight of slots that were enabled but not completed (or only partially). This could be used in calculating percentage or just derived as total minus completed. 3. **Apply Negative Penalties:** Sum up any penalty points for negative habits logged that day (e.g., junk food “cheat meal”, smoking, alcohol). Each negative event has a fixed negative weight: - For example: **Cheat Meal = -3 points**, **Alcohol consumption = -2**, **Smoking = -5** (exact values can be configured). If the user logged one cheat meal, we will subtract 3 from the completed weight (or effectively add to "missed" in terms of consistency). - Negative points directly reduce the day's score. They can even outweigh a small task’s completion. (We will ensure the daily score doesn’t go below 0; negative total will just be floored at 0%.) 4. **Apply Extra Credit:** If the user did extra positive activities that are not in the plan, we can give a small credit. For instance, an extra workout might be +1 or +2 points (capped). **However, the daily score cannot exceed 100%.** Extra credit is mainly to compensate for other misses or just to acknowledge extra effort, but we limit the effect: - If the user missed nothing, extra activities won’t raise them above 100 (so 100 is max). - If the user missed something, an extra activity can help make up the deficit *to a degree*, but not necessarily fully. (E.g., if they missed a 5-point workout but did an extra 5-point workout later, we could consider that fills the gap – effectively they did a workout, just at a different time – so score can go back towards 100. But if they missed a planned workout and their extra was just a short walk worth 1 point, it might only partially offset the miss.) - Essentially, we treat extra activities as additional completed weight, but **cap the completed weight at the total planned weight** (or slightly above with a small margin if we allow >100% up to a threshold like 110%, but the user indicated no infinite above 100, so likely cap at 100). 5. **Compute Percentage:** $$ \text{Daily Score %} = \frac{\text{(Completed Weight) - (Penalty Weight)}}{\text{Total Planned Weight (for that day)}} \times 100\%, $$ adjusted to 0–100% range. We then can apply the user’s strictness multiplier: if strictness slider is in use, the penalty weights or the denominator might be scaled accordingly. For simplicity in formula, assume we incorporate it into penalty: $$ \text{Effective Penalty} = \text{Penalty Weight} \times \text{StrictnessFactor}, $$ $$ \text{Daily Score \%} = \max\Big(0, \frac{\text{Completed Weight} - \text{Effective Penalty}}{\text{Total Weight}}\Big) \times 100\%, $$ capped at 100%. (If relaxed, StrictnessFactor < 1, meaning penalties are milder.)

This daily score dynamically updates as the user checks off items. At the end of the day, that final percentage represents how closely they stuck to the plan.

Interpreting Daily Score: A 100% means perfect adherence (all planned tasks done, no cheats). A score below 100 indicates some deviation: - Example: 80% might mean the user missed one significant activity or a few smaller ones (80% is also the chosen threshold for a "successful day" in streak terms). - A very low score (e.g. 50%) could mean a major slot missed (like skipping exercise and having a cheat meal). - It’s possible to get 0% if the user did almost nothing of the plan and/or did something with heavy negative penalty. - We do not expect negative percentages; 0 is the floor.

Negative Events (Cheat Meals, etc.): These have a multi-day impact notion in that one day's indulgence can affect overall momentum. In the daily score, its impact is immediate (a lower percentage that day). But there's no direct carry-over to the next day's *daily* score (each day resets fresh at 0 baseline). However, the impact carries into the **long-term consistency score** (next section) because that overall metric will reflect the drop and requires several good days to recover. This aligns with the idea that you can't undo a cheat with one extra workout on the same day – you need continuous consistency in following days to get back on track [6]. Thus, we achieve the multi-day effect via the momentum calculation rather than an arbitrary handicap on future daily scores.

Sample Pseudocode for Daily Score Calculation:

```
function calculateDailyScore(dayLog, planDay, strictFactor=1.0) {
    totalWeight = 0
    completedWeight = 0
    // Sum weights for enabled slots and calculate completed weight
    for each slot in planDay.slots:
        if slot.enabled:
            totalWeight += slot.weight
            slotCompleted = checkSlotCompletion(dayLog, slot.id)
            if slotCompleted == FULL:
                completedWeight += slot.weight
            else if slotCompleted == PARTIAL:
                // Partial credit proportional to items done
                completedWeight += slot.weight * slotCompleted.percent
            // if NONE, add 0
    // Add extra activities (positive) as bonus but do not exceed totalWeight
    extraWeight = sum(weight of extra activities logged in dayLog)
    // Cap bonus: allow compensation for missed but not beyond full plan
    effectiveCompleted = min(completedWeight + extraWeight, totalWeight)
    // Calculate penalties
    penaltyWeight = sum(penalty of each negative event in dayLog)  // penalties
defined as positive values
    effectivePenalty = penaltyWeight * strictFactor
    rawScore = (effectiveCompleted - effectivePenalty) / totalWeight
    rawScore = max(rawScore, 0)  // no negative score
    percentScore = round(rawScore * 100)
    return percentScore
}
```

In a relational database, if tasks and penalties were logged in tables, one might query the daily score components. For example, a SQL query to get a user's daily percentage could look like (assuming a table for planned slots, completed flags, and a table for logged penalties):

```sql
SELECT d.date,
       GREATEST(0,
         (SUM(CASE WHEN l.completed=1 THEN s.weight *
             (CASE WHEN l.partial_fraction IS NOT NULL THEN l.partial_fraction
ELSE 1 END)
           ELSE 0 END)
          + SUM(COALESCE(x.weight, 0))  -- x = extra activities
          - SUM(COALESCE(p.penalty, 0)) * :strictFactor
         ) / SUM(s.weight)
       ) * 100 AS consistency_percent
FROM plan_slots s
JOIN days d ON d.plan_id = s.plan_id
LEFT JOIN logs l ON l.slot_id = s.id AND l.date = d.date AND l.user_id = :user
LEFT JOIN extra_activities x ON x.date = d.date AND x.user_id = :user
LEFT JOIN penalties p ON p.date = d.date AND p.user_id = :user
WHERE d.date = :today AND d.user_id = :user
GROUP BY d.date;
```

(This is a conceptual query; actual schema may differ. It computes percentage = (completed + extras - penalties) / total.)

Threshold for “Good Day”: We define a threshold (say **80%**) above which a day is considered a success (a “green” day). This threshold (could be configurable per user or fixed at 80%) is used for streak calculations and user feedback. So: - If daily score >= 80%, we consider the day **achieved** (the user substantially met their routine). - This allows some slack (they might miss a minor thing or two and still count the day as a win). - A stricter user could set this threshold higher if desired via the strictness setting, but default is 80%.

Long-term Consistency and Momentum

While daily scores reset each day, we also maintain an **overall consistency score** or **habit strength** that reflects how the user has been doing over time. This score accounts for momentum: - If a user is consistently getting high daily scores, their overall consistency will increase. - If they have a lapse, a single bad day won’t destroy their progress entirely if they bounce back quickly (the momentum softens the impact of one-off misses), but repeated low days will drag the overall score down.

To implement this, we use a **rolling average or exponential smoothing** approach: - One simple measure could be a 30-day rolling average of daily percentages. For example, show the average consistency over the past month. However, a pure average might not emphasize recent performance enough or might be slow to change. - A better approach (used by some habit trackers like Loop) is an **exponential moving average (EMA)** that weights recent days more heavily [7]. Each day’s performance updates the score: - If the user’s overall score is low and they start performing well, the overall score will rise quickly (initial big gains). If the overall is high, continuing to be perfect yields smaller incremental gains (diminishing returns near 100) [1].

- One lapse after a long success streak will cause a dip, but not a crash, and returning to good habits will restore the score near its previous high relatively quickly [6]. However, frequent lapses will cause a steady decline. - This can be achieved by a formula like: $$ S_{new} = S_{prev} + \alpha (\text{dailyScore} - S_{prev}) $$ where α is a smoothing factor (e.g. 0.1 or 0.2). Or equivalently $S_{new} = (1-\alpha) S_{prev} + \alpha (\text{dailyScore})$. This gives more weight to recent dailyScore if α is, say, 0.2. - We may calibrate α such that roughly one month of perfect days can nearly maximize the score. For instance, Loop's data suggests ~3 months of perfect habit to approach ~99% [8]. We could choose $\alpha \approx 0.05$ to have a longer memory, or higher for shorter memory. - We will call this overall metric **Consistency Score** (0–100) as well, but it's understood as a separate long-term gauge. Perhaps "Habit Consistency Score" to differentiate from daily percent. It could be shown as, for example, *85/100*, and maybe a textual status like "Consistency: Stable" or so.

Streaks: In addition to the smooth score, we track streaks which are motivating: - **Perfect Day Streak:** Count of consecutive days meeting or exceeding the threshold (>=80% daily). Each day that qualifies extends the streak; a day below 80% resets it. We also track the **longest streak** achieved historically. - This streak count is a classic motivator (don't break the chain). - For display, we might show "Current streak: 5 days (goal ≥80% consistency per day)".

- **Category Streaks:** For each category or key slot, we can track streaks of adherence. For example, *Exercise Streak* = consecutive scheduled exercise days where the user did not skip the exercise. If the plan says user exercises Monday, Wed, Fri (and those are the only days with exercise slots enabled):
- The exercise streak would count how many exercise slots in a row were completed on the days they were supposed to happen. If Monday and Wednesday were done, that's a 2 streak; if Friday is missed, it resets.
- Essentially, each category streak advances when the user completes that category's slot on all days it was expected, and breaks if they miss on a day it was expected.
- For daily categories (like taking a medication every day, or doing a morning routine daily), this becomes a daily streak similar to habit streaks. For less frequent ones, the streak is only counted across occurrences.

- Category streaks help users identify consistency in specific areas (e.g., "You've done your workout 5 times in a row" or "You haven't missed a lunch meal in 20 days!").

- We will also track **"Perfect Weeks" or "Perfect Months"** – e.g., number of weeks where user met their daily goal >= 5 out of 7 days, etc., or months with at least X% average. These can be further achievements to motivate.

Analytics & Trends: Beyond raw scores: - We store daily scores and possibly category-wise completion stats. This enables charts and insights: - A **calendar heatmap** of daily consistency (each date colored by the percent or by simply green/red if above threshold) [2]. - **Weekly average trend** – e.g., a line graph of weekly average consistency over the last 12 weeks to show improvement or decline. - **Category performance** – e.g., a bar graph for each category (exercise, food, etc.) showing the percentage of those tasks completed over the last week or month. This can highlight problem areas (perhaps the user is 90% consistent with exercise but only 60% with their diet plan, indicating diet is the area to focus). - **Slot-specific trends** – drill down on a particular slot. For example, if "Post Dinner (20:30)" habit (say taking supplements) is frequently missed, the app could show that trend (maybe 50% completion over last month) and suggest improvements. - We also log whenever users swap or add extra items, which can later be analyzed to see *patterns* (e.g., user tends to swap out Wednesday lunch often – maybe they don't enjoy what was planned; or user consistently adds an extra evening walk – maybe the plan can include it formally). - Over time, analytics could detect improvements or regressions: e.g., *"You improved your consistency from 70% last month to 85% this month, great job!"*, or *"Consistency has dropped in the last two weeks, review your routine or try adjusting the difficulty."*

We will provide some sample outputs and scenarios below to illustrate these calculations and data formats.

Example Consistency Scenarios

To clarify how the scoring works in practice, consider the following scenarios with a sample daily plan:

Assume a day's plan has total weight 15, consisting of: 1 exercise slot (5), 3 main meals (3 each = 9), 1 hydration task (0.5), and 1 snack (0.5). Total = 15 (for simplicity, including two 0.5 tasks as 1 combined). Threshold for success = 80%.

- **Scenario A: Missing a High-Weight vs Low-Weight Activity**
  Case 1: User skips the **exercise** (5 points) but does everything else. Completed weight = 10, missed = 5. No extra, no penalties. Daily score = (10/15)*100 = 66.7%. This is well below 80%, so this breaks the streak and is clearly flagged as a low-consistency day (missing the highest-weight activity had a big impact).*
  Case 2: *User does exercise but skips an evening snack (0.5 points). Completed = 14.5, missed = 0.5. Score = (14.5/15)*100 = **96.7%**. This would still be considered an excellent day (snack is minor, almost 97%). It easily meets the 80% threshold, so the streak continues.
  Case 3: User skips one main meal (3 points) but does others. Completed = 12, score = 80%. That's right at the threshold (still a "pass" by our definition). Missing a meal (moderate weight) gave a moderate drop, but the day can still be salvaged as successful if everything else was done.

- **Scenario B: Extra Activity Bonus**
  Case 1: User misses the 0.5-weight hydration task (drinking water) but does an **extra 30-min walk** (assume we give that an extra credit of 1 point). Originally, missing hydration: Score would be (14.5/15)=96.7%. With an extra 1 point, completed weight effectively becomes 15 (capped at 15 max). Score becomes (15/15)=**100%**. The extra walk covered the only missed minor task, bringing the day to a perfect score.
  Case 2: User follows the plan 100% and also does an extra workout (+5). Completed would be 15, extra 5 -> 20, but we cap at 100% (15). So they still get **100%** (not 133%). The extra effort doesn't raise the percentage above 100, but it could be noted in the app as bonus effort (for personal satisfaction or perhaps to buffer a future cheat, but mathematically we handle future separately).
  Case 3: User misses a main meal (3 points) but does an extra workout (+5). Completed from plan = 12, extra = 5, sum = 17, cap to 15. Essentially the extra workout can compensate for the missed meal fully (we'll count at most 3 of that extra 5 to fill the gap). Score ~100%. In reality, we might just mark it as 100% (the user made up for the missed meal by burning calories or so). We may not always allow a one-to-one replacement (since eating vs exercise aren't directly interchangeable), but from a pure consistency standpoint, doing something extra strenuous could offset a missed meal in score. (This is a design decision: we might also decide not to fully compensate cross-category like this. But given user's note of "additional workout gives only little extra benefit," we might instead only allow a

partial offset. E.g., maybe only 50% of extra workout's weight counts toward missed meal category. These nuances can be adjusted in the scoring algorithm if needed.)

- **Scenario C: Cheat Meal Penalty**
  User follows the plan fully (15 points completed) but indulges in a **cheat meal** logged at -5 points. Effective completed weight = 15, penalty = 5. Score = ((15 - 5) / 15)*100 = 66.7%**. So despite doing all planned tasks, the day is pulled down to ~67% because of the unhealthy choice. This fails the 80% threshold (a "red" day). The user's perfect streak resets. Importantly, the effect of this cheat will reflect in the long-term consistency:

- Suppose before this day, the user's overall consistency score was 90. One bad day at 67 will cause a dip. If we use exponential smoothing (say α=0.1), the new overall score might be ~87 (a drop, but not all the way to 67 because of momentum).
- If the user gets back to 100% for the next days, the overall score will gradually climb back towards 90. It might take several days (maybe 4-5 days of perfect scores to get back to ~90 from 87). This matches the notion that after a cheat, you need a number of good days to fully recover your consistency momentum [9] .
- If the user had another cheat or bad day soon after, the overall score would drop further and be slower to recover. Frequent cheats would push their long-term consistency significantly lower.

These scenarios show how the weighting and penalties work in practice, and how certain behaviors impact the daily score differently. It aligns with the idea that missing high-priority activities or having unhealthy cheats greatly affects your consistency, whereas missing minor tasks or doing extra can balance out.

Data for Visualization and Analytics

To support the front-end in displaying progress, the system will output data in structures convenient for visualizations like heatmaps and trend charts. Below are sample data formats:

- **Calendar Heatmap Data:** An array of dates with their consistency scores. This can be used to color-code calendar days (e.g., green for high, red for low).

```json
[
  { "date": "2025-08-01", "consistency": 100 },
  { "date": "2025-08-02", "consistency": 85 },
  { "date": "2025-08-03", "consistency": 60 },
  ...,
  { "date": "2025-08-31", "consistency": 92 }
]
```

  Each entry provides the daily score%. The front-end can map 0-100 to color intensity. Alternatively, we could supply a simplified status (e.g., "success": true/false if >=80%) for a binary coloring, but percentage gives more nuance.

- **Slot/Category Trend Data:** We can structure data to show how each slot or category is being followed over time. For example, a weekly summary of each category completion:

```
{
  "weekStart": "2025-08-01",
  "category_completion": {
    "Exercise": 4 / 5,   // completed 4 out of 5 scheduled exercise slots this week
    "Food": 20 / 21,     // completed 20 of 21 meal slots (3 meals * 7 days = 21)
    "Hydration": 10 / 14,
    "Habit": 6 / 7
  }
}
```

This could be transformed into a bar chart (e.g., Exercise 80%, Food ~95%, etc. for that week). We would produce such data for each week to see trends (maybe a small multiple or sparkline for each category over weeks).

Alternatively, a timeline of each slot's completion:

```
"slot_history": {
  "Workout (07:00)": [
     {"date": "2025-08-01", "done": true},
     {"date": "2025-08-02", "done": true},
     {"date": "2025-08-03", "done": false},
     ...
  ],
  "Lunch (13:30)": [
     {"date": "2025-08-01", "done": true},
     {"date": "2025-08-02", "done": true},
     {"date": "2025-08-03", "done": true},
     ...
  ]
}
```

This allows plotting a raster of each slot across the calendar (similar to habit tracker grids for each habit). Slots or categories with lots of misses will visibly have more gaps.

- **Streaks and Averages:** We can output metrics like:

```
{
  "current_day_streak": 5,
  "longest_day_streak": 12,
  "current_exercise_streak": 8,
  "current_food_streak": 15,
  "overall_30d_average": 88.5,
  "overall_consistency_score": 91
}
```

The `overall_consistency_score` is the exponential smoothing value (habit strength). `30d_average` is a simple average of last 30 days for reference. These can be shown on a dashboard. A chart of overall_consistency_score over time could also be drawn (it will be a smoother line than daily percent).

- **Consecutive Misses or Patterns:** We might also surface if the data shows a pattern, e.g., "User tends to skip the Evening Snack on most Fridays." Such insights can be generated by querying the logs (e.g., filter completion by weekday).

For backend analytics, we can utilize SQL to retrieve some of these: - Daily scores for heatmap:

```
SELECT date, consistency_score
FROM daily_consistency
WHERE user_id = :user AND date BETWEEN :start AND :end;
```

- Weekly category completion:

```
SELECT week_start,
       SUM(CASE WHEN category='Exercise' AND completed=1 THEN 1 ELSE 0 END)
         || '/' || SUM(CASE WHEN category='Exercise' THEN 1 ELSE 0 END) as
exercise_ratio,
       ... (repeat for other categories)
FROM slot_completion_log
WHERE user_id = :user
GROUP BY week_start;
```

(Above uses string concatenation for ratio illustration; in application logic, better to output two numbers or a percentage.)

- Current streak calculation (pseudo-SQL or logic):
  One way is to query the last time a failure happened:

```
SELECT COUNT(*) as current_streak
FROM (
  SELECT date
  FROM daily_consistency
  WHERE user_id = :user AND date <= CURRENT_DATE
  ORDER BY date DESC
  /* take rows until a day with consistency_score < 80 is encountered */
) sub;
```

This is not standard SQL; typically we'd fetch the last N days until break via application logic. Pseudocode for streak:

```
streak = 0
for each day from today going backwards:
    if day.consistency >= 80: streak += 1
    else break
```

Similar approach per category: iterate backwards on scheduled days.

These data structures and queries allow us to create rich visual feedback for the user, highlighting both daily performance and long-term habits.

System Implementation Considerations

Real-time vs Batch Calculations: The app will calculate the day's progress in real-time on the device as the user checks off items (for instant feedback like "You're at 60% of your daily goal"). This is straightforward since it's just summing weights of completed tasks. At the end of the day (or when the user marks the day done), the final daily score is recorded on the server. Long-term metrics (streaks, averages, overall consistency) can be updated in a batch manner once per day (e.g., at midnight or the next time the user opens the app the following day). This separation ensures heavy calculations (like recomputing a 30-day average or updating the smoothed score) are not done constantly on the device, but we also don't need immediate updates of overall score until day is finished. Overnight processing or a lightweight cron job can handle updating streak counts and the exponential smoothing score for the new day.

Offline Support: The user might not always be online. The application will: - Maintain a **local copy** of the finalized plan JSON and the daily logs (completed/missed status, etc.). The user can mark tasks offline; these will be saved locally (e.g., in a SQLite database or local storage). - When connectivity is available, the app syncs the log updates to the backend. The backend can then recompute any scores if needed. - Because we assume primarily one device in use at a time (the user's phone, for example), conflicts are minimal. If multiple devices were used, we'd need to merge logs, but we expect one primary device which simplifies syncing (no concurrent edits to reconcile beyond simple additive logs). - The app itself can compute and display the daily percentage offline. For long-term data, if offline, it can show cached values and update once connected.

Backend and Real-time Framework: We will use a dedicated backend (with a database) to store: - User plans (DSL JSON, versions), - Daily activity logs (what was done, swaps, extras, cheats), - Derived metrics (daily score, streak, overall score, etc.).

Given the need for real-time updates (when online) and offline sync, an architecture using a cloud database with offline capabilities is suitable. Options include: - **Firestore/Firebase**: Provides local caching, real-time sync, and offline persistence out-of-the-box (but it's not open-source). It could handle storing the JSON plan and daily checkmarks easily, and syncing when online. - **Supabase (Postgres + realtime)**: Open-source alternative, with realtime subscription support. We could use Postgres for storing data (plans, logs) and Supabase's listener to get live updates. The app could poll or subscribe to changes for multi-device support. - **Custom Backend**: A Node/Express or Django API with a WebSocket or Socket.io for realtime. The app would call REST endpoints to post updates, and could receive push notifications for any updates. Given cost and complexity, leveraging an existing solution like Supabase might be easier and cost-effective (as the user indicated preference for open source and low cost). - **Local database**: For offline first, something like CouchDB/PouchDB combination could allow sync, but that might be overkill given we have a central server anyway.

The data model likely has tables like: `Plans`, `PlanSlots`, `PlanItems`, `DailyLog`, `LogItems`, etc., but since the DSL is one big JSON, we could store it as a JSON column in `Plans` for simplicity and have a separate `DailyLog` table for completion status and notes.

Notifications and Reminders: The app will schedule reminders for each slot time. Since times are fixed by the plan skeleton, we can, upon plan finalization, set local notifications on the device for those times (repeating weekly). For example, “07:00 – Workout time!” or “20:00 – Dinner”. On iOS/Android, local notifications can be set to recur on certain weekdays. If any changes to the plan (new plan) occur, we update the schedule. Basic alarm/notification functionality suffices (no special server push needed unless we plan remote notifications, but not necessary for user’s requirement).

Data Integrity: Since the plan and logs relate but plan is immutable, the daily log should reference the plan (plan id and slot id). If a plan is updated (new version), it gets a new id; ongoing logging will reference the current active plan id for that date range. This way, historical logs remain tied to the plan version used at that time.

Swaps Implementation: When a user swaps slots between two days, the system can handle it by swapping the *planned items in the UI for those days*, but not altering the plan storage: - One approach: in the DailyLog, have a way to note if a slot’s content was replaced by another day’s variant. E.g., log an entry that Monday Lunch used Tuesday’s item X. We could even log it as if Monday’s lunch completed = false (for original item) but an extra entry was done which matches Tuesday’s lunch item. But simpler: We might treat it as both Monday and Tuesday lunch completed, just each with the other's item. This could complicate item-level tracking but since we only care slot completion, we can simply mark both slots done on their respective days. - For nutrition tracking, if needed, one might track exactly what was consumed each day, but that’s beyond consistency – it’s more for a nutrition log. For our consistency, it's binary per slot (done or not). - We will, however, maintain a record for user’s reference: e.g., in Monday’s log, note “Swapped: had Tuesday’s planned meal instead.” This can be a note field.

Partial Completion: To implement partial credit, each slot’s items could be individually checkable. In the log, we can store something like `completed_count` / `total_count` of items for that slot, or simply a percentage per slot. If we need a finer granularity (maybe for nutrition, if they ate 80% of the calories, but let's not go that far), count of items is enough. - If partial completion is common (e.g., user ate 2 out of 3 components of a meal), the weight could be prorated by items: 2/3 of weight. - This is a straightforward calculation once we have item-level completion data.

Security & Privacy: Considering HIPAA (since diets, habits could be considered health-related data especially if tied to a nutritionist or trainer scenario): - **Data Encryption:** All communication should be over HTTPS. Sensitive personal data on the device and server should be encrypted. The database should encrypt ePHI at rest (especially if any medical info is included). For a habit app, the data (meals, exercises) might not be PHI unless tied to medical treatment, but to be safe, we treat it as sensitive. Implement encryption as required (AES-256 for any at-rest encryption, etc.) [3] . - **Access Control:** Only the user (and authorized personnel, if any, like a coach) can access their plan and logs. Ensure strong authentication (passwords, or OAuth, possibly 2FA) and consider biometric unlock on the app. If this app is solely personal (no doctors involved), HIPAA might not strictly apply [10], but good practice is to protect the data regardless. - **De-identification & Sharing:** If aggregate analytics or social features are added (like sharing streaks), ensure no sensitive detail is shared without consent. Any reporting or export should strip personal identifiers unless the user requests their data specifically. - **Audit Trail:** Maintain logs of changes in case needed (especially if in future a healthcare provider reviews the data, one might need an audit trail of edits). - **HIPAA Compliance Consideration:** If the app ever integrates with healthcare providers or falls under a covered entity, we would need full compliance (business associate agreements, etc.). At this stage, design with the security rule in mind so that scaling to compliance is easier (unique user IDs, automatic logoff after inactivity [11], and so on).

In summary, the system will combine a flexible plan representation (DSL) with robust tracking and scoring logic to motivate users. It leverages proven habit-tracking concepts (streaks, a consistency score that rewards sustained effort [2], visual progress charts) and is built with user customization and data integrity at its core. The next steps would be to refine the DSL format with any additional fields needed, implement the backend calculations and API, and create a friendly UI that presents this information clearly for user adoption.

---

[1] [6] [7] [8] [9] Frequently Asked Questions · iSoron uhabits · Discussion #689 · GitHub
https://github.com/iSoron/uhabits/discussions/689

[2] Top 12 Habit Tracker Apps You Need in 2025
https://niftypm.com/blog/best-habit-tracker-apps/

[3] [10] [11] HIPAA Compliance for Mobile Health Apps | Utility
https://utility.agency/resources/hipaa-compliance-for-mobile-apps-a-brief-guide

[4] [5] Customized Nutrition Plan 6 - Vishal Thota_AUG 2025.pdf
file://file-FNeER3e9YRor9VrWM61s3C