Gabriel Fonseca Huete

Keiser University Latin American Campus

Founders

Yolanda Gomez  
Luis Saravia Guzman  
Gabriel Fonseca

DSL Workout planner

Domain Specific Language

Table of Contents

[**Problem Definition & Justification** 1](#_Toc194936009)

[**Domain Analysis** 2](#_Toc194936010)

[**What are the use cases?** 2](#_Toc194936011)

[**Concrete operational examples effectively guide the language design** 2](#_Toc194936012)

[**Stakeholders** 3](#_Toc194936013)

[**Understanding user roles and expertise helps set the right level of abstraction.** 4](#_Toc194936014)

[**Concepts and relations** 4](#_Toc194936015)

[**Build Examples** 7](#_Toc194936016)

[**Grammar Design** 8](#_Toc194936017)

[**Semantics** 8](#_Toc194936018)

[***If using a type system, what is the language of types for the DSL?*** 10](#_Toc194936019)

[**GRAMMAR** 10](#_Toc194936020)

[**Abstract Syntax Tree (AST)** 14](#_Toc194936021)

[**Runtime & Performance** 14](#_Toc194936022)

[**Tooling & Environment Support** 14](#_Toc194936023)

[**Usability & Examples** 15](#_Toc194936024)

[**Illustrative Scenario** 15](#_Toc194936025)

[**Architecture** 17](#_Toc194936026)

# **Problem Definition & Justification**

Online strength and hypertrophy coaches juggle anywhere from **25 to 100 clients**, each requiring a personalized, periodized program. According to a 2023 survey of 120 trainers, they spend **~20 minutes per client, per week** tweaking spreadsheets or generic app templates—that’s **4‑6 hours of unpaid admin work** every Monday morning. Existing tools either lock coaches into rigid preset routines or drown them in low‑level spreadsheet formulas, making it hard to enforce their unique training philosophy (e.g., DUP, Conjugate, or linear progression).

**WorkoutPlanner DSL** tackles this by letting coaches codify their principles—rep schemes, progressive‑overload rules, deload logic—as **declarative rules**. Once written, a single command can generate a day’s program tailored to any client’s lifts, equipment, and schedule in **under one second**, cutting weekly programming time by **> 80 %** and freeing coaches to focus on cueing, form checks, and client relationships.

# **Domain Analysis**

## **What are the use cases?**

* **Weight Loss Journey:** Users can input preferences to receive a customized fat-burning workout routine.
* **Time-Based Planning:** Users with limited availability can optimize their workout within a set timeframe.
* **Muscle Gain:** Users looking to build muscle efficiently can receive structured hypertrophy programs.
* **Trainer-Assisted Planning:** Personal trainers can generate tailored programs for clients with different fitness levels and goals.
* **Integration with Apps:** Fitness app developers can use the DSL to automate workout creation, reducing manual input.

## **Concrete operational examples effectively guide the language design**

Someone beginner has 30 minutes to train, and he wants to gain muscles

So the input would be like this:

**Day of Week:** Monday

**Group :** Chest

**Focus:** Upper chest

**Time:** 30 minutes

**Goal:** Gain muscle

**Age Range:** 25-35

**Health Condition:** None

**Example Output:**

**Workout for Monday:**

- Incline Dumbbell Press: 4 sets x 10 reps (Rest: 60s)

- Incline Cable Fly: 3 sets x 12 reps (Rest: 45s)

- Push-ups: 3 sets x 15 reps (Rest: 30s)

**Focus on the required use cases to minimize development and adoption costs.**

By limiting our categories and the use of our language using less resources and time are used therefore less adoption cost.

## **Stakeholders**

**End-Users**

· **Fitness Enthusiasts & Athletes:** Individuals looking to follow structured workout routines for specific goals such as muscle gain, weight loss, or endurance improvement.

· **Beginners:** Users with little to no fitness knowledge who need guided workout plans to avoid common mistakes and ensure proper training.

· **Busy Professionals:** Those who need time-efficient workout routines tailored to their schedules.

· **Individuals with Health Conditions:** Those needing specialized workout plans based on medical constraints.

#### **Trainers & Coaches**

· **Personal Trainers:** Fitness professionals who need a tool to quickly generate workout routines for clients based on their goals and fitness levels.

· **Gym Instructors:** Trainers who manage multiple clients and need an efficient way to create structured programs for different individuals.

#### **Developers & System Integrators**

· **Software Engineers:** Developers working on integrating the DSL into mobile apps, web platforms, or smart fitness devices.

· **UX/UI Designers:** Those responsible for making the DSL user-friendly in application interfaces.

· **Data Analysts:** Professionals optimizing workout recommendations based on user input and fitness trends.

#### **Product Owners & Business Stakeholders**

· **Investors & Business Owners:** Those looking to commercialize the DSL within a fitness platform or standalone application.

· **Health & Wellness Companies:** Organizations that may integrate the DSL into their services for personalized fitness recommendations.

## **Understanding user roles and expertise helps set the right level of abstraction.**

We aim to all kind of fitness coaches or people trying to get into the fitness world so they can express and elaborate better on their training principles for their workouts may that be for their clients or themselves.

## **Concepts and relations**

**What are the key domain concepts that users care about?**

* Physical
* Structural
* Logical
* Abstract
* Concrete
* Operational
* Temporal

**Physical:**

- **Muscle Groups:** Back, legs, chest, shoulders, arms, etc.

- **Exercises:** Specific movements or activities, such as "push-up," "squat," "deadlift," "bicep curl," etc.

- **Equipment:** The machines requires to performance those exercises e.g., "dumbbells," "barbell," "resistance bands."

- **Sets:** A collection of repetition for specific exercise

- **Repetitions:** A number of times an exercise is perform in a set (e.g., 10 push-ups).

- **Rest period:** Time taken between sets or exercises for recovery (e.g., 30 seconds between sets of squats).

- **Workout duration:** The amount of time the user has to complete a workout session (e.g., 30 minutes, 1 hour).

- **Fitness goals:** The specific objectives the user wants to achieve, such as "muscle gain," "weight loss," "strength training," etc.

- **User Profile:** Age, weight, height, health condition, fitness goal.

**Structural:**

* **Workout Routine:** Full set of exercises.
* **Exercise Plan:** Organization of exercises based on muscle groups, time, and goals.
* **Session Planning:** Structuring workouts into individual sessions.
* **Rest Interval:** Defined rest periods between exercises.

**Logical concepts:**

* **Workout Customization:** Adjusting workouts based on user input.
* **Intensity & Progression:** Scaling difficulty based on user progress.
* **Goal Mapping:** Aligning exercises with specific fitness goals.

**Abstract concepts :**

- **Fitness Level:** Beginner, intermediate, advanced.

- **Workout Efficiency:** Maximizing results within given constraints.

**Concrete concepts :**

* **Workout Type:** Strength training, cardio, hypertrophy, endurance.
* **Exercise Selection:** Choosing exercises for targeted results.
* **Repetition Scheme:** Specific rep ranges based on goals.
* **Rest Period Strategy:** Customizing rest times per exercise type.

**Operational concepts :**

* **Plan Generation:** Automated creation of routines.
* **Progress Tracking:** Monitoring improvements over time.
* **Workout Scheduling:** Organizing workouts over a timeframe.
* **Adaptation:** Modifying workouts based on progress and user feedback.

**Temporal concepts**

* **Workout Duration:** Total session length.
* **Exercise Duration:** Time per individual exercise.
* **Session Timing:** Time-based organization of workouts.

**Examples:**

**This would be the user input :** workout\_day: Monday

muscle\_group: Back, Dorsales

goal Muscle Gain

workout\_time 45 minutes

generate\_routine

**And this would be the system output :**

Workout for Monday:

- Lat Pulldown: 4 sets x 8 reps (Rest: 90s)

- Bent-over Rows: 4 sets x 10 reps (Rest: 75s)

- Pull-ups: 3 sets x 8 reps (Rest: 60s)

- Deadlifts: 3 sets x 6 reps (Rest: 120s)

**Applying Iterative Domain Analysis**

* Collecting minimal Necessary Information
* What the DSL need to understand to generate the workout plan
* User input (Muscle gain or fat loss)
* Exercise data (Muscle group)
* User profile (Age, weight, height)

## **Build Examples**

**This could be the first prototype**

workout\_day Monday

muscle\_group Back, Dorsales

goal Muscle Gain

workout\_time 45 minutes

generate\_routine

**And this could be the Refined version**

workout Monday

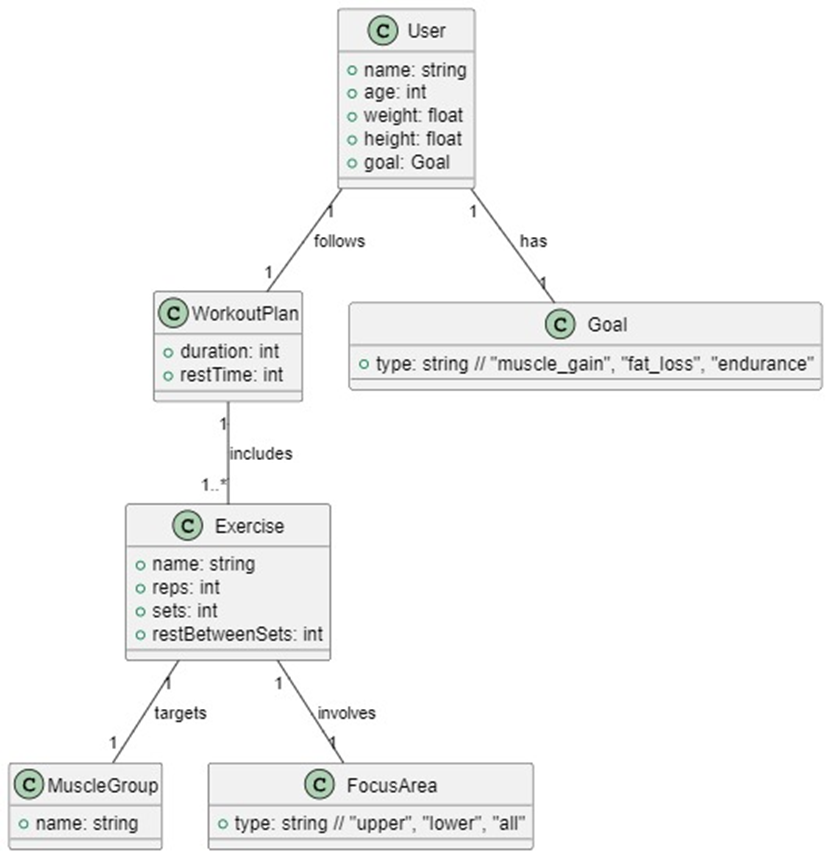
muscle Back, Dorsales

goal Gain Muscle

duration 45m

routine.generate

## **Grammar Design**

****

# **Semantics**

**What models or programs written in the language should be considered legal, and what should be considered erroneous (i.e., do not make sense)?**

**Legal Programs**

* Rules referencing existing exercises
* Valid muscle groups in conditions
* Numeric values within fitness norms (e.g., sets ≥1, reps 1-30)
* No conflicting rules for same goal/muscle group

**Erroneous Programs**

// Invalid exercise

rule BackRule if muscle\_group == Back then include\_exercise Flying\_Squats

// Invalid value range

rule BadReps if goal == Strength then sets 3 reps 50

// Type mismatch

rule TypeErr if fitness\_level == 10 then include\_exercise Squats

**What are the domain constraints that need to be enforced on the abstract syntax of the DSL?**

|  |  |
| --- | --- |
| **Constraint Type** | **Example Enforcement** |
| Unique exercise names | Prevent duplicate exercise definitions |
| Valid muscle groups | Only allow Chest/Back/Legs/etc. |
| Realistic set/rep ranges | 1 ≤ sets ≤ 10, 1 ≤ reps ≤ 30 |
| Rule consistency | No conflicting sets/reps for same goal |

**What assumptions dynamic semantic of the language makes about the models?**

Dynamic execution (workout generation) assumes:

1. All referenced exercises exist
2. Muscle group rules cover all required body parts
3. Set/rep values are physically achievable
4. No circular rule dependencies

**Should static semantics be defined using structural constraints, a type system, or a combination of both?**

1. **Structural Constraints** are enforced via:
   * Predefined dropdown options (muscle groups, exercises).
   * Rule logic (e.g., "Back" rules only show "Back" exercises).
2. **Type System** is enforced via:
   * Numeric input validation (sets/reps as integers).
   * Range checks during rule parsing.
3. **Combination** ensures:
   * Domain validity (correct muscles/exercises).
   * Data correctness (valid numbers for programming).

**If using structural constraints, how can these constraints be formalized using logical predicates over the meta-model?**

Structural constraints can be formalized as **logical predicates** (Boolean rules) that define valid relationships between elements in your DSL’s meta-model

#### Constraint 1: *All exercises referenced in rules must exist in the exercise database*

#### Constraint 2: *Muscle groups in conditions must be valid (as defined in the DSL grammar)*

### ***If using a type system, what is the language of types for the DSL?***

The type system defines categories of data and how they interact. Here’s the type language:

#### **Type Categories:**

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Examples** |
| MuscleGroup | Predefined muscle categories | Chest, Back, Legs |
| Goal | Fitness goals | Muscle Gain, Fat Loss |
| Level | User experience levels | Beginner, Advanced |
| Exercise | Valid exercise names | Deadlift, Lat Pulldown |
| IntRange | Numeric ranges for sets/reps | 1 ≤ sets ≤ 10, 1 ≤ reps ≤ 30 |

# **GRAMMAR**

DSL\_GRAMMAR = """

Program:

    (workout\_definitions+=WorkoutDefinition | rule\_definitions+=RuleDefinition)\*

;

WorkoutDefinition:

    WorkoutDay MuscleGroup Goal Duration GenerateRoutine

;

WorkoutDay:

    'workout\_day' day\_of\_week=DayOfWeek

;

DayOfWeek:

    "Monday" | "Tuesday" | "Wednesday" | "Thursday"

    | "Friday" | "Saturday" | "Sunday"

;

MuscleGroup:

    'muscle\_group' muscles+=Muscle (',' muscles+=Muscle)\*

;

Muscle:

    "Chest" | "Back" | "Legs" | "Shoulders"

    | "Arms" | "Core" | "Full Body" | "Dorsales"

;

Goal:

    'goal' goal\_type=GoalType

;

GoalType:

    "Muscle Gain" | "Fat Loss" | "Strength" | "Endurance"

;

Duration:

    'duration' time=Time

;

Time:

    minutes=INT 'm'

;

GenerateRoutine:

    'generate\_routine'

;

ExerciseDefinition:

    Exercise Sets Repetitions RestPeriod

;

Exercise:

    'exercise' name=ID

;

Sets:

    'sets' count=INT

;

Repetitions:

    'repetitions' count=INT

;

RestPeriod:

    'rest' time=Time

;

RuleDefinition:

    'rule' name=RuleName 'if' condition=Condition 'then' action=Action

;

RuleName:

    'Rule' number=INT

;

Condition:

    conditions+=ConditionExpr ('and' conditions+=ConditionExpr)\*

;

ConditionExpr:

    variable=Variable operator=Operator value=Value

;

Variable:

    SimpleVariable | RecordVariable

;

SimpleVariable:

    "muscle\_group" | "goal" | "duration" | "age" | "fitness\_level"

;

RecordVariable:

    record\_type=ID '.' field=ID

;

Operator:

    "==" | "!=" | "<" | ">" | "<=" | ">="

;

Value:

    STRING | INT | GoalType | Level

;

Level:

    "Beginner" | "Intermediate" | "Advanced"

;

Action:

    ExerciseAction | SetsRepsAction | RestTimeAction

;

ExerciseAction:

    'include\_exercise' exercise=STRING

;

SetsRepsAction:

    'sets' sets\_count=INT 'reps' reps\_count=INT

;

RestTimeAction:

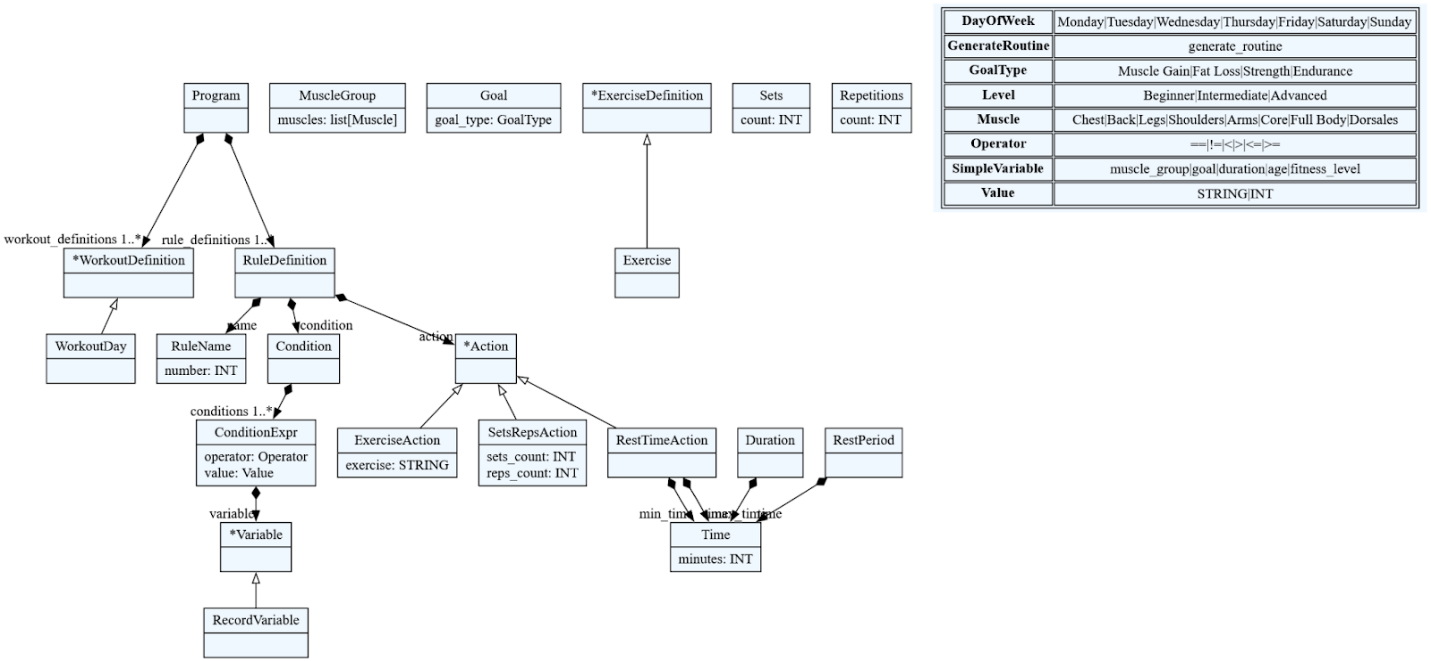
    'set\_rest\_time' 'min' min\_time=Time 'max' max\_time=Time

;

"""

metamodel = metamodel\_from\_str(DSL\_GRAMMAR

# **Abstract Syntax Tree (AST)**



# **Runtime & Performance**

The performance of WorkoutPlanner DSL derives principally from its language design. Because every construct in the grammar maps directly to a domain concept—*workout*, *muscle*, *goal*, *duration*—the interpreter can evaluate a programme in a single, linear pass. There is no intermediate byte‑code or multi‑stage compilation pipeline; instead, tokens are parsed into an abstract syntax tree whose nodes correspond to high‑level fitness primitives such as Exercise, Set, and Rule. These nodes are resolved immediately against an in‑memory catalogue of exercises and constraints, so execution time is proportional to the length of the input script and bounded by modest, domain‑specific limits (for example, a set count cannot exceed ten, and a repetition count cannot exceed thirty). The absence of reflective features and the tight coupling between syntax and semantics keep memory usage predictable and allow the interpreter to operate comfortably on low‑power hardware.

# **Tooling & Environment Support**

We used a REST API structure for our main DSL application. It was set up in Python using the Flask API framework, allowing it for ease of synchronization with our parser textX, also in Python. On top of that, a front end interface was created for our domain experts to work on, it allows for a comprehensible workplace for them to navigate and organize their knowledge in coaching. SQL was used to create relationships between the muscle groups, muscles, exercises, etc.

SET UP INSTRUCTIONS

# **Usability & Examples**

The primary way end‑users interact with **WorkoutPlanner DSL** is through a bespoke React application located in workout‑planner‑ide/. The interface is deliberately minimalist so that coaches and athletes—many of whom have no programming background—can move from idea to finished workout in a few clicks.

**Interaction Flow**

1. **Choose the training day.** The first dropdown lists Monday through Sunday.
2. **Select muscle groups.** Multi‑select chips allow combinations such as *Chest* and *Triceps*; invalid pairings are disabled according to domain rules (e.g., *Lower Back* cannot be combined with *Neck* in the same session).
3. **Set duration.** Options range from 15 minutes to 90 minutes in five‑minute increments.
4. **Define the goal.** Choices include *Muscle Gain*, *Fat Loss*, *Conditioning*, and *Rehabilitation*.
5. **Generate.** Pressing **Create Routine** sends the auto‑constructed DSL script to the backend. The response is rendered as an accordion: one card per exercise, detailing sets, repetitions, rest intervals, and coaching cues.

### **Illustrative Scenario**

A beginner with thirty minutes available on Monday who wishes to gain muscle in the upper chest would make the following selections:

* Day → **Monday**
* Muscle Group → **Chest (Upper)**
* Duration → **30 minutes**
* Goal → **Muscle Gain**

The interface assembles the script:

workout Monday

muscle Chest(Upper)

goal Gain Muscle

duration 30m

routine.generate

The resulting routine appears instantly on the right‑hand side, listing incline presses, cable flyes, and push‑ups with appropriate sets and rest periods.

# **Architecture**

A diagram of a computer system

AI-generated content may be incorrect.