Fretted String Instrument Practice System (FSIPS)

Version 0.1 — Architecture & Design Summary

---

1. Overview

The **Fretted String Instrument Practice System (FSIPS)** is a deterministic, fatigue-aware, instrument-agnostic framework for generating structured daily practice sessions for fretted string instruments (bass guitar, guitar, extended-range variants, alternate tunings).

The system is explicitly modeled after **periodized strength training** rather than ad-hoc music practice. Exercises are treated as reusable training primitives, sessions are composed from constrained blocks, and progression is driven by explicit overload dimensions rather than intuition or randomness.

The design goals are:

- Long-term coverage of all technical, harmonic, rhythmic, and musical skills
- Progressive overload that is explainable and controllable
- Explicit management of mechanical and cognitive fatigue
- Instrument-agnostic core architecture (bass is a configuration, not a special case)
- Deterministic, debuggable session generation
- Preservation of instructor intuition while eliminating memory loss and inconsistency

FSIPS is not an app yet. It is a **formal system specification** that can later be implemented as a CLI tool, notebook workflow, or application.

---

2. Core Design Principles

1. **Exercises are abstract; instruments are configurations**
2. **Overload is explicit and orthogonal** (one knob at a time)
3. **Fatigue is stateful and variant-dependent**
4. **Session structure constrains exercise choice**
5. **Determinism beats randomness**
6. **Instructor heuristics are encoded, not replaced**

---

3. Instrument Abstraction

InstrumentProfile

Physical characteristics are separated from exercises.

Key properties: - `string_count` - `tuning[]` (ordered low → high) - `technique_capabilities[]` (fingerstyle, pick, slap, tapping, etc.) - optional `scale_length`

Exercises reference: - string indices (0 … n-1) - fret distances - interval relationships

No exercise hardcodes: - bass vs guitar - string names - fixed tunings

---

4. Canonical Exercise Model

Exercise

A **canonical exercise** is an atomic conceptual unit (≈35 total identified so far). Variations are produced via overload settings, not duplication.

Key fields: - id, name - domains (Technique, Harmony, Rhythm, Musicianship) - technique tags - supported overload dimensions - instrument compatibility constraints

Exercises do **not** encode tempo, rhythm, or difficulty directly.

---

5. Overload Dimensions

Progression is controlled through explicit, orthogonal **overload dimensions**.

Categories include:

- **Temporal density**: tempo, subdivision, note density
- **Spatial load**: range, string topology, position constraints
- **Pattern complexity**: permutations, layering, ornamentation
- **Rhythmic stress**: accent displacement, polymeter, metric modulation
- **Constraint stacking**: technique restrictions, starting conditions
- **Volume / endurance**: duration, repetition count

Rule:

> Increase **only one overload dimension at a time**.

---

6. Fatigue Model

Fatigue is modeled simply and intentionally.

Fatigue Profiles

- **F0 (Low)** — precision, recovery, safe daily
- **F1 (Medium)** — standard development, requires rotation
- **F2 (High)** — CNS or tendon intensive, limited frequency

Fatigue is assigned to **exercise variants**, not exercises.

---

7. SessionBlocks (Structural Constraint Layer)

A **SessionBlock** is a constraint envelope that defines what kind of work is allowed in that portion of a session.

Canonical SessionBlocks

1. **Warmup / Calibration**
2. **Technique Development**
3. **Pitch / Harmony Development**
4. **Rhythm / Time Manipulation**
5. **Application / Musicianship**

Each block defines: - allowed domains - allowed fatigue profiles - overload bounds - explicitly disallowed combinations

Global Invariants

- Warmup is mandatory if any F2 work appears
- No adjacent F2 blocks
- Cognitive and mechanical peaks are separated
- Application always occurs last

---

8. Session Generator Logic

The generator is deterministic and state-driven.

Inputs

- InstrumentProfile
- Available time
- Session type (normal / light / heavy / deload)
- Goal weights

Persistent State (future)

- Exercise recency

- Rolling volume (7d / 28d)
- Mastery estimate
- Recent fatigue exposure

Generation Algorithm

1. Build session skeleton based on time
2. For each SessionBlock:
3. Filter compatible exercises
4. Score candidates using a need model
5. Select highest-need exercise
6. Assign overload via bounded progression rules
7. Enforce fatigue budgets
8. Emit an explainable, executable session plan

No machine learning is required.

---

9. Executable Output Requirement

A session is considered **invalid** unless each SessionBlock emits:

- Selected exercise
- Concrete playable variant
- Human-readable instructions
- Focus cues
- Stop / regression conditions

This guarantees immediate usability.

---

10. Current Capabilities (v0.1)

At this milestone, FSIPS can:

- Generate structured 20–75 minute practice sessions
- Balance technique, harmony, rhythm, and application
- Progress exercises safely and explicitly
- Adapt across fretted string instruments
- Explain why each exercise appears

---

11. Explicit Non-Goals (for now)

- UI / application implementation
- Audio analysis or transcription

- Machine learning or recommendation systems
- Perfect mastery estimation

These are deferred layers.

Appendix A — Identified Next Steps

The system is now at a clean branching point. Recommended next steps include:

1. **Reality Testing**
2. Walk a real 7-day practice week end-to-end
3. Validate fatigue budgeting and overload progression
4. **Logging UX Design**
5. Define the minimal daily logging surface
6. Optimize signal vs friction
7. **Mesocycle & Deload Logic**
8. Add bodybuilding-style accumulation, intensification, deload phases
9. **Exercise Instruction Templates**
10. Formalize how exercises auto-emit concrete playable variants
11. **Code-Level Specification**
12. Translate architecture into classes, schemas, and functions
13. **Instructor Review & Refinement**
14. Incorporate feedback and resolve disagreements explicitly

---

Status: Architecture frozen at v0.1