SYSTEM IN 73

Detailed Participant Handout & Evaluation Walkthrough

This sprint is designed to **teach you how real systems are built under pressure**.

You are not only being evaluated — you are **learning industry-grade habits**.
Every requirement below exists for a reason. Read carefully.

What This Sprint Is About

In real engineering:

- Requirements are incomplete
- Time is limited
- Systems must still work
- Changes arrive at the last minute

This sprint simulates that reality in **73 minutes**.

Qualification Rules (Very Important)

Direct Qualification

- The **first 10 teams** to make a **final valid Git commit** within **73 minutes** will qualify.

Qualification is **binary** — either your system works and is complete, or it doesn't.

Step 1: Choose a Simple System

You may choose **any small problem domain**, such as:

- Task manager
- Registration system
- Library borrowing
- Attendance tracker

- Inventory system

Guidance

- Choose something you can **finish**, not something impressive
- If it takes more than 5 minutes to explain, it's too big

You will NOT be judged on the domain.

Step 2: Design the Data Model (SQL)

What You Must Build

- **Minimum 3 tables**
- Each table must have:
  - Primary key
  - At least one constraint (`NOT NULL`, `UNIQUE`, `FOREIGN KEY`)
- At least **one relationship** between tables

Example

Event system:

- events(id, name, capacity)
- users(id, email)
- registrations(id, user_id, event_id)

Constraint example:

- `email` must be unique

Why We Check This

- This tests if you can think in **data structures**
- Weak schemas break systems early

Step 3: Enforce One Real Rule

Requirement

You must enforce **at least one real-world rule** that prevents invalid actions.

Examples:

- Prevent duplicate registrations

- Prevent exceeding capacity
- Prevent invalid status changes

Rules may be implemented using:

- SQL constraints
- Stored procedures
- Backend logic (must be explained)

What Evaluators Look For

- Rule actually works
- Rule blocks bad input

Step 4: Represent State (JSON)

Requirement

Create **one JSON file** that represents your system's state.

It must include:

- Current state
- Allowed next states

Example

```
{
  "status": "pending",
  "allowed_next": ["approved", "rejected"]
}
```

Why This Matters

- Most systems are **state machines**
- State controls what is allowed next

Step 5: Version Control with Git (Critical)

Mandatory Git Rules

- Use Git from the beginning
- Minimum **3 meaningful commits**
- Commits must show progression

Recommended commit flow:

1. `init: add database schema`
2. `feature: enforce business rule`
3. `feature: add minimal UI`

Why Git Is Heavily Weighted

- Git shows **how you think**, not just what you type
- In real teams, Git quality = trust

Poor Git discipline = automatic disqualification.

Step 6: Minimal UI

Requirement

Create a **basic interface** that:

- Accepts input
- Triggers logic
- Displays state

Examples:

- HTML form
- Simple page
- CLI-style interface

UI design does NOT matter.

Step 7: README (Mandatory)

Your README must clearly explain:

- What the system does
- What rule is enforced
- What the JSON state represents
- How to run or test the system

If the evaluator cannot understand your system in **2 minutes**, it will not qualify.

What Is a “Final Valid Commit”?

A final commit is valid only if:

- Pushed before **73 minutes cutoff**
- All core requirements are present
- Git history is clean and logical
- README exists and is readable

Missing any part = **not counted**.

Evaluation Walkthrough (How You Are Judged)

Evaluators will check:

1. **Completeness** – all required artifacts exist
2. **Correctness** – rules actually work
3. **Clarity** – README and commits make sense
4. **Discipline** – Git history is clean

There are **no style points**.

Final Advice

- Keep it small
- Don’t chase perfection
- Make rules explicit
- Commit with intent
- Stay calm

This sprint is about proving you can **build correctly under pressure**.

Good luck.