Project Structure & Architecture Document

Date: December 21, 2025
Version: 1.0
Status: Active Development

Executive Summary

This document outlines the complete architecture and modular structure of the Airport Ground Operations Simulator - a research-grade Python simulation engine for airport ground traffic. The system models aircraft taxiing, queueing, and interactions on the airport surface using a discrete-time cellular automaton approach.

Project Timeline (4 Weeks)

Week 1: Foundation & Infrastructure

Day	Tasks	Deliverables
1-2	Project setup, layout.py, params.py	Graph representation, hyperparameter system
3-4	rules.py, capacity.py	Rule engine, resource management
5	Integration testing	Working foundation layer

Week 2: Domain Logic & Routing

Day	Tasks	Deliverables
1-2	routing.py (pathfinding)	Dijkstra's algorithm, route precomputation
3-4	spawning.py	Aircraft generation at gates/runway ends
5	Testing & refinement	Complete domain logic layer

Week 3: Core Simulation & UI Integration

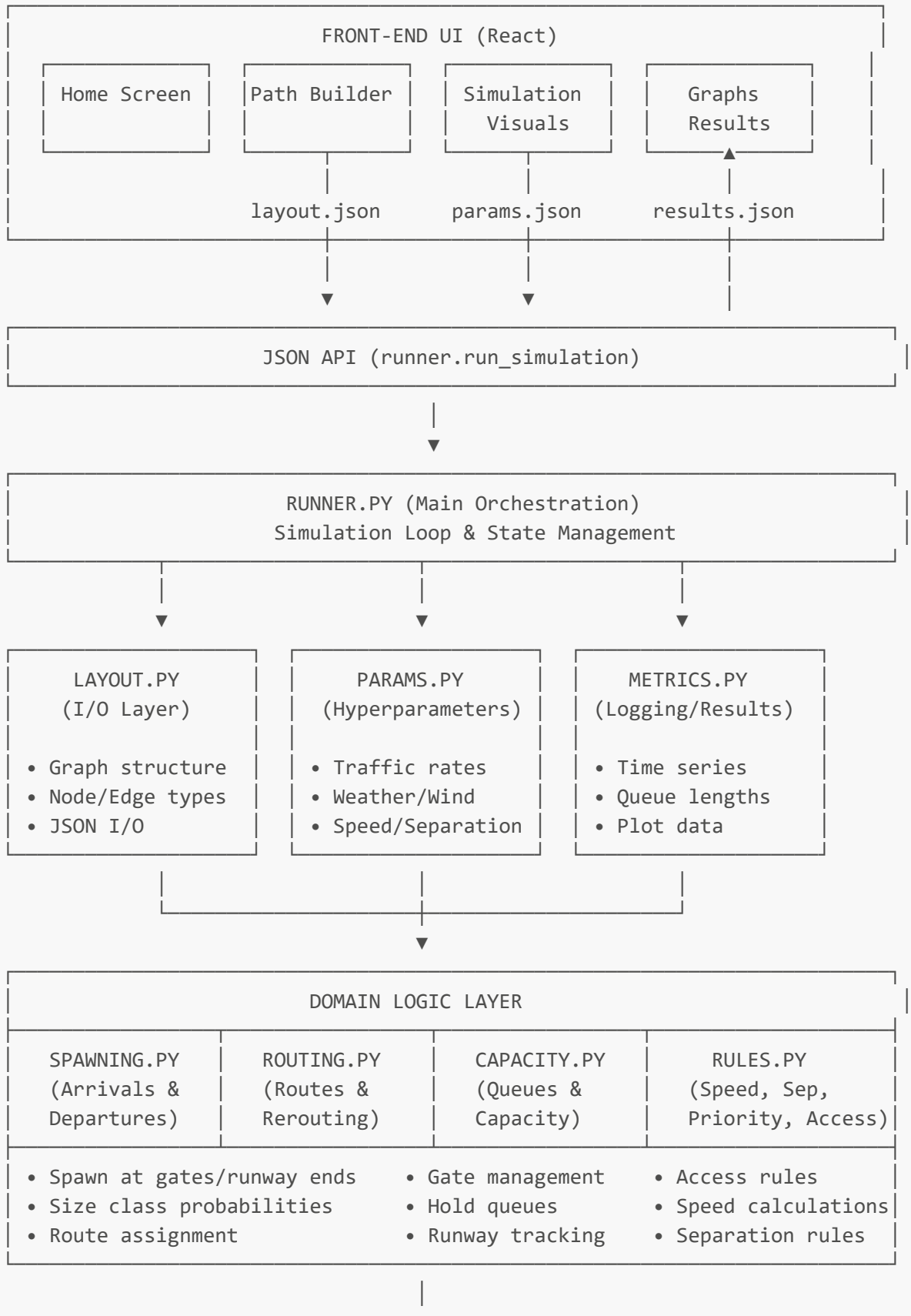
Day	Tasks	Deliverables
1-2	model_core.py (stub implementation)	CA state, step function interface
3-4	runner.py, metrics.py	Main loop, data collection
5	UI integration	JSON API working with React frontend

Week 4: Path Builder UI & Polish

Day	Tasks	Deliverables
1-3	Path Builder enhancements	Interactive layout creation tools

Day	Tasks	Deliverables
4-5	Testing, documentation, handover	

High-Level Architecture



```
                                    ▼
┌──────────────────────────────────────────────────────────────────────┐
│                    MODEL_CORE.PY (Cellular Automaton)                │
│                        [Research Partner Module]                     │
│                                                                      │
│  • SimulationState: aircraft positions, edge occupancies, time       │
│  • init_state(layout, params) → initial state                        │
│  • step(state, layout, params, rules, capacity) → (new_state, metrics)│
│                                                                      │
│  CA Movement Rules: acceleration, braking, collision avoidance,      │
│  separation enforcement, stochastic noise injection                  │
└──────────────────────────────────────────────────────────────────────┘
```

Module Specifications

1. LAYOUT.PY - Airport Graph Representation

Purpose: Represent the airport as a graph and handle JSON I/O.

Node Types:

- `INTERSECTION` - Taxiway/runway crossings
- `RUNWAY_END` - Start/end of runway
- `RUNWAY_ENTRY` - Entry point onto runway
- `RUNWAY_EXIT` - Exit from runway (including rapid exits)
- `GATE` - Aircraft parking position
- `HOLD_POINT` - Waiting position before runway/restricted area
- `APRON_CENTER` - Central point in apron area

Edge Types:

- `RUNWAY` - Main runway segment
- `TAXIWAY` - Standard taxiway
- `APRON_LINK` - Apron/gate connector
- `RAPID_EXIT` - High-speed runway exit (arrivals only)

Key Functions:

```
Layout.from_json(json_str) -> Layout
Layout.to_json() -> str
Layout.get_node(node_id) -> Node
Layout.get_edges_from_node(node_id) -> List[Edge]
```

---

2. PARAMS.PY - Hyperparameter System

Purpose: Central configuration for all simulation parameters.

Parameter Categories:

Category	Parameters	Description
Traffic	spawn rates, class mix, mode	Aircraft generation settings
Environment	weather, wind speed/direction	Environmental conditions
Movement	base speeds, multipliers	Speed calculations
Separation	distances by area, weather factors	Safety distances
Priority	runway, intersection, hold modes	Conflict resolution
Capacity	gates per apron, runway capacity	Resource limits
Simulation	time step, duration, seed	Run configuration

Parameter Modes:

- `OFF` - Feature disabled
- `FIXED` - Constant value
- `RANDOM` - Random within range
- `REALISTIC` - Based on real distributions

---

3. RULES.PY - Logic Engine

Purpose: Encode all rules for access, speed, separation, and priority.

Rule Types:

1. **Access Rules**
    - Arrivals: arrival-only edges, both, rapid exits
    - Departures: departure-only edges, both (no rapid exits)
    - Size restrictions on gates/taxiways
2. **Speed Rules**
    - `Speed = base_speed[class] × section_mult × weather_mult`
    - Runway > Taxiway > Apron (fastest to slowest)
3. **Separation Rules**
    - Runway: N/A (only 1 aircraft allowed)
    - Taxiway: configurable (default 50m)
    - Apron: configurable (default 30m)
    - Weather multipliers (bad weather = larger spacing)
4. **Priority Rules**
    - FIFO (first-come-first-served)
    - Depart-first / Arrive-first

- Weighted (custom scoring)
- Size priority (LARGE > MEDIUM > SMALL)

---

4. SPAWNING.PY - Traffic Generation

Purpose: Create arrivals and departures based on parameters.

Aircraft Properties:

- ID, size class (SMALL/MEDIUM/LARGE)
- Type (arrival/departure)
- Route, position, speed, status

Spawn Logic:

- Departures: At free gates matching size class
- Arrivals: At active runway ends (based on wind)
- Rate controlled by parameters

---

5. ROUTING.PY - Pathfinding

Purpose: Route assignment and rerouting.

Features:

- Dijkstra's algorithm for shortest paths
- Precomputed routes for common O-D pairs
- Forward-only rerouting (no reverse/backtrack)
- Wind-dependent runway direction selection

---

6. CAPACITY.PY - Resource Management

Purpose: Track and manage limited resources.

Resources Managed:

1. **Gates**: Occupancy tracking, assignment logic
2. **Hold Points**: Queue management, priority-based release
3. **Runways**: Strictly 1 aircraft at a time

---

7. MODEL_CORE.PY - Cellular Automaton

Purpose: Core simulation physics (Research Partner).

Interface:

```
init_state(layout, params) -> SimulationState
step(state, layout, params, rules, capacity) -> (new_state, observables)
```

State Contents:

- Aircraft list (edge, position, speed, route, status)
- Edge occupancies
- Simulation time
- Aggregated counters

Observables (returned each step):

- Edge densities and flows
- Queue lengths at holds
- Runway throughput

---

8. METRICS.PY - Data Collection

Purpose: Record data and prepare results for visualization.

Collected Metrics:

- Total aircraft on ground vs time
- Queue lengths at hold points vs time
- Runway throughput vs time
- Per-aircraft: taxi time, waiting time

Output Format:

```
{
  "summary": {
    "total_departures": 45,
    "total_arrivals": 38,
    "avg_taxi_time": 245.3,
    "avg_wait_time": 42.1
  },
  "plots": [
    {"type": "line", "label": "Aircraft on Ground", "x": [...], "y": [...]},
    {"type": "line", "label": "Hold Queue Length", "x": [...], "y": [...]}
  ]
}
```

---

9. RUNNER.PY - Main Orchestrator

Purpose: Entry point for simulations.

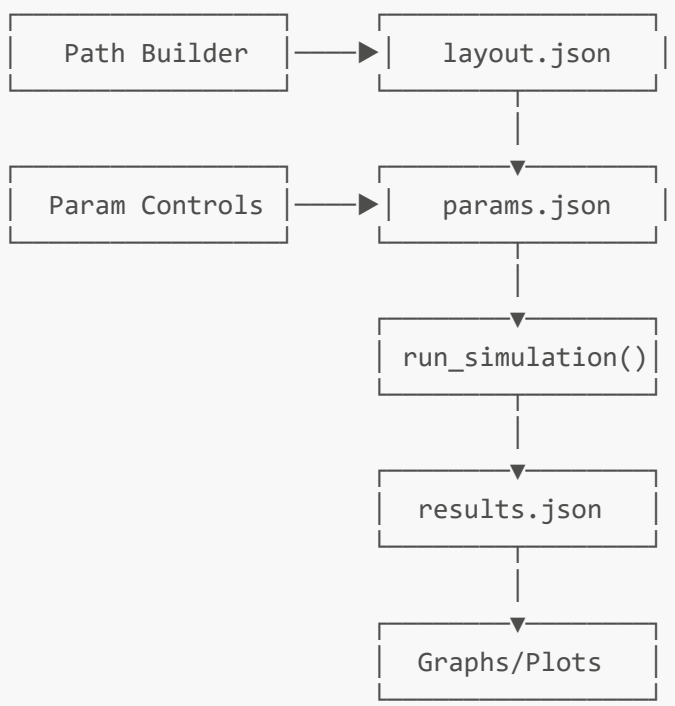
API:

```
run_simulation(layout_json, params_json) -> results_json
```

Main Loop (each time step):

1. Evaluate environment (weather, wind, runway config)
2. Spawn new arrivals/departures
3. Update capacity state (gates, holds, runway)
4. Call model_core.step() for movement
5. Log metrics
6. Accept mid-run parameter updates (optional)

Data Flow



File Structure

```
airport-simulator/
├── simulator/                    # Python backend package
│   ├── __init__.py
│   ├── layout.py                 # Graph representation
│   ├── params.py                 # Hyperparameter system
│   ├── rules.py                  # Rule engine
│   ├── spawning.py               # Traffic generation
│   ├── routing.py                # Pathfinding
│   ├── capacity.py               # Resource management
│   ├── model_core.py             # CA core (research partner)
│   ├── metrics.py                # Data collection
│   ├── runner.py                 # Main orchestrator
│   ├── schemas/
```

```
│   │   ├── layout_schema.json
│   │   └── params_schema.json
│   └── tests/
│       └── test_*.py
├── data/
│   ├── layouts/                  # Saved airport layouts
│   │   └── sample_airport.json
│   └── params/
│       └── default.json
├── ui/                           # React frontend
│   └── src/
│       ├── pages/
│       │   ├── Home.jsx
│       │   ├── PathBuilder.jsx
│       │   ├── Simulation.jsx
│       │   └── Graphs.jsx
│       └── ...
└── docs/
    └── PROJECT_STRUCTURE.md      # This document
```

Key Design Principles

1. **Data-Driven**: Every rule, rate, and behavior is configurable via JSON
2. **Parameter-Driven**: UI sends JSON; backend interprets without code changes
3. **Modular**: Clear separation of concerns between modules
4. **Extensible**: Easy to add new rules, metrics, or aircraft behaviors
5. **Testable**: Each module has well-defined interfaces for unit testing

---

Division of Work

Our Team Implements:

- ☑ layout.py - All graph I/O and representation
- ☑ params.py - Complete hyperparameter system
- ☑ rules.py - Access, speed, separation, priority logic
- ☑ spawning.py - Aircraft generation
- ☑ routing.py - Pathfinding and rerouting
- ☑ capacity.py - Resource management
- ☑ runner.py - Main orchestration loop
- ☑ metrics.py - Data collection and plotting
- ☑ UI components - All React frontend pages

Research Partner Implements:

- model_core.py - CA state and step function:
  - Aircraft movement physics
  - Acceleration/deceleration rates
  - Collision avoidance

- Stochastic noise injection
- Statistical physics metrics (optional)

Contact & Next Steps

Current Status: Milestone 1 - Architecture & Basic Simulation Loop

Next Milestone: Path Builder UI enhancements and full simulation visualization

For questions or clarifications, please reach out to the development team.