BOL - Beginning of Life Simulator

Integration Document

Generated: 2026-07-11 02:38
Version 1.0.0

1. Integration Overview

This document specifies every integration point in the BOL system: REST API endpoints, external service dependencies, data exchange formats, inter-module communication, and deployment pipeline.

Integration architecture: monolithic-with-clean-boundaries. Browser communicates via REST API (JSON over HTTPS). Server communicates with simulation via in-process Python calls with threading locks. File I/O for profiles, projects, scenarios, training, and bugs. Client-side CDN dependencies for Chart.js, Three.js, and Font Awesome. Browser-native Web Speech API for TTS.

2. REST API - Simulation Control

POST /api/start - Start a new simulation (accepts scenario, steps, seed, project_id). POST /api/stop - Stop running simulation. POST /api/pause - Toggle pause. GET /api/state - Current simulation state with molecule counts, complexity, fitness. GET /api/history - Time series for charting. GET /api/events - Recent event log. GET /api/scenarios - List available scenarios. POST /api/sweep - Run parameter sweep.

3. REST API - Data Endpoints

GET /api/particles - 3D particle positions for viewers. GET /api/molecules - Molecule type catalogue. GET /api/reactions - Live reaction fire counts and feed. GET /api/reaction_rules - Reaction rule metadata (temp, pH, rate, delta-G). GET /api/lineage - Protocell lineage tree. GET /api/energy - Energy depth profile. GET /api/network - Reaction network graph.

4. REST API - AI Laboratory

POST /api/ai/<module>/<action> - Run one of 27 analysis functions against live simulation state. Modules: prebiotic (pathway, yield, thermo, mineral), autocatalysis (raf, cycles, growth), membrane (cmc, stability, division), replication (template, error, ribozyme, heredity), metabolism (coupling, tca, gradient), selection (landscape, group, transitions), environment (sweep, compare, cycles), synthesis (full, bottleneck, hypothesis, report).

5. REST API - Projects

GET /api/projects - List projects (auth required). POST /api/projects - Create project. GET /api/projects/<id> - Get project details (owner check). PUT /api/projects/<id> - Update project config. DELETE /api/projects/<id> - Delete project. POST /api/projects/<id>/run - Run simulation with project config. GET /api/projects/<id>/download - Download project JSON. POST /api/projects/upload - Import project from JSON. GET /api/projects/<id>/report - Generate PDF report.

6. REST API - User Accounts

GET /api/profile - Current auth status. POST /api/profile/register - Register account. POST /api/profile/login - Authenticate (rate-limited 10/5min per IP). POST /api/profile/logout - End session. POST /api/profile/update - Update display name/bio. POST /api/profile/password - Change password. GET /api/users - List users (admin only). DELETE /api/users/<username> - Delete user (admin only).

7. REST API - Training & Bug Tracker

GET /api/training/modules - Full curriculum (8 categories, 32 modules). GET/POST /api/training/progress - Read/update user progress. GET/POST /api/training/notes/<id> - Read/save lesson notes. POST /api/training/quiz/<id> - Submit and grade quiz. GET /api/bugs - List bugs. POST /api/bugs - Create bug. PATCH /api/bugs/<id> - Update bug. DELETE /api/bugs/<id> - Delete bug. POST /api/bugs/<id>/test - Run individual test. POST /api/bugs/test-all-open - Test all open bugs.

8. External Dependencies

Python packages: Flask >=3.0 (BSD-3), NumPy >=1.24 (BSD-3), SciPy >=1.10 (BSD-3), Matplotlib >=3.7 (PSF), Vispy >=0.14 (BSD-3), Gunicorn (MIT, production only).

Client-side CDNs: Chart.js (cdn.jsdelivr.net) for dashboard charts, Three.js (cdn.jsdelivr.net) for 3D viewer, Font Awesome 6 (cdnjs.cloudflare.com) for icons.

Browser APIs: Web Speech API (TTS for training), Service Worker (PWA offline caching), Canvas 2D (microscope renderer), WebGL (Three.js 3D viewer).

9. Data Exchange Formats

Project JSON: {id, name, owner, description, created, updated, config: {initial_molecules, environment, energy_sources, reactions, replication, world_size}, runs: [{timestamp, steps, seed, results: {complexity, total_molecules, vesicles, diversity, fitness}}]}.

Scenario JSON: {name, description, initial_molecules, environment, energy_sources, reactions, replication}.

User Profile JSON: {username, password_hash, salt, display_name, bio, role, created}. Password hash uses PBKDF2-HMAC-SHA256 with 260,000 iterations.

Export formats: CSV (one row per step), JSON (final summary), PDF (multi-page report).

10. Deployment Pipeline

Step 1: Provision server - setup_server.sh creates bol user, installs Python 3.12, Nginx, certbot. Step 2: Deploy code - deploy_to_server.ps1 rsyncs project to /opt/bol, creates virtualenv. Step 3: Configure services - finish_setup.sh creates systemd unit, Nginx virtual hosts, SSL certs. Step 4: Harden headers - fix_nginx_headers.sh applies CSP, HSTS, X-Frame-Options to Nginx.

Environment variables: BOL_SECRET_KEY (session signing), BOL_SECURE_COOKIES=1 (HTTPS cookies), BOL_ADMIN_USERS (comma-separated admin usernames).

Infrastructure: GCP Compute Engine e2-small, Ubuntu 24.04 LTS, Nginx reverse proxy, Gunicorn 3 workers, systemd process manager, Let's Encrypt SSL, ufw firewall.

11. Inter-Module Communication

Threading model: simulation runs in background thread, Flask serves HTTP on main thread. Shared objects protected by _sim_lock, _project_lock, and _bug_lock threading locks.

Module dependency: config.py -> world.py -> {energy, chemistry, assembly, replication} -> simulation.py -> metrics.py -> web/server.py -> ai.py.

Data flow: Browser -> HTTPS -> Nginx -> HTTP -> Gunicorn -> Flask Route -> Page Route (Jinja2 -> HTML) or API Route (-> JSON). API routes either read/write JSON files or query simulation state.