SimDock: A Phased Development Roadmap

Phase 1: Foundational Backend Refactoring, Data Hygiene & QA

Goal: Address all critical bugs, build a stable backend, ensure no "bad data" enters the system, and enforce strict code quality via testing.

- **Establish the Project Structure:**
    - **Action:** Create the `simdock_pro/` directory with sub-folders: `core`, `gui`, `utils`, and **`tests`**.
    - **Action:** Initialize a strict Python environment to separate dependencies.
- **Abstract Configuration & Engine Abstraction:**
    - **Action:** Move hardcoded paths (Vina, OpenBabel) to `core/config_manager.py` (resolving cross-platform issues).
    - **Action:** Create the `DockingEngine` base class in `core/docking_engine.py` to support future engines (Vina, Smina, AutoDock-GPU).
    - **Action:** Implement `VinaEngine` inheriting from the base class to handle specific execution logic.
- **Smart Ligand & Protein Preparation (`core/file_manager.py`):**
    - **Action:** Implement "Smart Ligand Prep" logic that takes raw 2D structures (SDF/CSV), assigns Gasteiger charges, fixes bond orders, and generates conformers automatically.
    - **Action:** Develop "PDB Clean-Up" logic to parse raw PDB files, strip solvents, remove non-standard residues, and fix missing atoms.
- **Project Management & Version Control (`core/project_manager.py`):**
    - **Action:** Implement `ProjectManager` for relative-path session saving (resolving broken session loading).
    - **New Feature: "Git for Proteins" (Target Version Control).** Implement versioning for receptor files (e.g., `HER2_v1_raw` vs `HER2_v2_fixed`) to prevent users from running jobs on deprecated targets.
- **Unit Testing Suite (`tests/`):**
    - **Action:** Write **initial unit tests** to verify that `VinaEngine` can execute a simple dry-run docking.
    - **Action:** Write integrity tests for `ProjectManager` to ensure saving/loading a

session results in zero data loss.

Phase 2: Modern Desktop GUI & Workbench Experience

Goal: Create a responsive "Workbench" interface using `customtkinter` that offers real-time feedback, intuitive workflows, and bridges to external tools.

- **Build the Main Window with "Glass Panel" Dashboard:**
  - **Action:** Design the layout with a "Glass Panel" dashboard. Replace static logs with a real-time status pane that slots in "Molecule Cards" as jobs finish.
  - **Action:** Implement Drag-and-Drop Ingest for folders of files, triggering a "Processing Ring" visual for bulk preparation.
- **Wizard-Style Workflows & Robust Logging:**
  - **Action:** Create a GUI wrapper for Phase 1 preparation logic. Display a checklist (e.g., "Strip Solvent", "Fix Missing Atoms") that gets checked off automatically.
  - **Action: Implement Robust Logging.** Replace silent failures with detailed logging to a file. Catch exceptions and display user-friendly error messages (e.g., "Ligand preparation failed: Check bond orders" instead of "Error 1").
- **Results Triage & External Bridge:**
  - **Action:** Implement a "Traffic Light" Triage System (🔴 Fail, 🟡 Review, 🟢 Synthesize) and filtering.
  - **Action: External Viewer Bridge.** Add functionality to right-click a result and "Open in ChimeraX" or "Open in VMD" for expert-level analysis.
  - **Action:** Add Raw Data Export (one-click download of `.pdbqt`, `.log`, and `.csv`).
- **Async Operations:**
  - **Action:** Ensure all heavy lifting (docking, prep) runs in background threads to keep the UI responsive (fixing UI freezes).

Phase 3: High-Throughput "Galaxy" & Database Intelligence

Goal: Expand to batch processing with intelligent data management, "fail-fast" logic, and advanced visualization.

- **Integrate Multiple Docking Engines:**
  - **Action:** Implement concrete engine classes (e.g., `SminaEngine`, `AutoDockGPU_Engine`) extending the base `DockingEngine` class created in Phase 1.
  - **Action: Update GUI Selection.** Add a "Docking Engine" dropdown menu to the

setup tab, allowing users to switch algorithms per run (e.g., use Vina for speed, Smina for custom scoring, or AutoDock-GPU for massive libraries).

- **Galaxy Visualization (The Bulk View):**
  - **Action:** Implement a "Galaxy View" for batch results. Render results as floating orbs plotted by Binding Affinity (Y-Axis) and Structural Similarity (X-Axis), color-coded from Red to Neon Green.
- **Intelligent Database Integration:**
  - **Action:** Replace CSVs with SQLite (`project.db`) for efficient storage of large screening campaigns.
  - **New Feature: Graveyard Alert.** Query the database before a run to check if this exact simulation was run previously and failed.
  - **New Feature: Hypothesis Tags.** Add mandatory metadata tags (e.g., "Scaffold Hopping") to track intent.
- **Active Site Auto-Detection:**
  - **Action:** Implement an algorithm to scan the protein surface for deep cavities. Highlight these as glowing spheres in the GUI; clicking snaps the grid box to that location.
- **Fail-Fast Logic:**
  - **Action:** Implement early termination for simulations showing high steric clashes or non-convergence.

Phase 4: Web Transition & Ecosystem Integration

Goal: Evolve into a scalable web platform with distinct frontend/backend separation, task queues, and business logic.

- **Headless REST API & Task Queue:**
  - **Action:** Wrap core logic in a **FastAPI** backend with endpoints (`/api/dock`, `/api/score`).
  - **Action: Implement Task Queue (Celery + Redis).** Offload long-running docking jobs to a Redis-backed queue system to prevent server timeouts and enable massive scaling.
- **Modern Web Frontend & Visualization:**
  - **Action:** Build a Single Page Application (SPA) using **React or Vue** to interact with the API.
  - **Action: Embed Web Viewer.** Integrate **NGL Viewer** or **Mol*** directly into the web UI for zero-install 3D visualization.
- **Real-Time Collaboration:**
  - **Action:** Use WebSockets to enable live session URLs ("Multiplayer Mode"), syncing rotation/zoom between users.
- **Deployment & Business Intelligence:**
  - **Action: Cloud-Ready Containerization.** Dockerize the full stack (API, Worker,

    Frontend, DB) for deployment on AWS/GCP or self-hosted private clouds.
  - **Action:** Compute Cost Estimator & One-Click Report Generator (PDFs with 3D screenshots).

Phase 5: AI-Powered "Copilot", Plugins & Advanced Security

Goal: Add cutting-edge AI assistance, allow community extensibility, and ensure enterprise-grade security.

- **Extensible Plugin Architecture:**
  - **Action:** Architect a plugin system allowing the community to contribute new `DockingEngine` adapters or analysis modules without altering core code.
- **SimDock Copilot & AI Scoring:**
  - **Action:** Integrate a sandboxed, Local LLM to answer queries (e.g., "Summarize binding differences").
  - **Action: ML Re-Scoring Pipeline.** Add a post-docking step to re-score poses using trained Machine Learning models (e.g., RF-Score, NNScore) for higher accuracy than Vina alone.
  - **Action:** Generative Suggestions (e.g., holographic wireframes for structural modifications).
- **"Safety Net" (Real-Time ADMET):**
  - **Action:** Run background AI inference to predict Toxicity/Solubility with a glowing "Shield Icon" warning system.
- **Security & Governance:**
  - **Action:** Immutable Audit Logs (User ID, Timestamp, File Hash).
  - **Action:** RBAC (Role-Based Access Control) & SSO (Single Sign-On).
  - **Action:** 3D Annotation ("Sticky Notes" on specific atoms).