Genome Guide: Project Context & Architecture (Backend Focused)

1. Project Overview

Genome Guide is a bioinformatics engine that programmatically calculates answers to questions from *"A Short Guide to the Human Genome"* using raw genomic data.

Current Phase: Pure Backend Development (ETL + Analysis Pipelines).
Goal: Build a robust system to answer all questions in Chapters 2-10 before building any UI.

---

2. Technical Architecture

The Pipeline

1. **Raw Data Layer:**
    - **Genome:** `hg38.fa` (3GB) - Loaded into `chromosomes` table.
    - **Annotations:** `Homo_sapiens.GRCh38.gtf` (1GB) - Loaded into `genes` and `exons` tables.
    - **Repeats:** `simpleRepeat.txt` (UCSC) - Loaded into `simple_repeats` table.
    - **Future Inputs:** `cpgIslandExt.txt` (CpG Islands), `pep.all.fa` (Proteins), `rmsk.txt` (RepeatMasker).
2. **Storage Layer (SQLite):**
    - **File:** `backend/genome_guides.db`
    - **Schema (Current):**
        - `chromosomes` (id, name, length, sequence)
        - `genes` (id, gene_id, gene_name, gene_type, start, end, chrom_id)
        - `exons` (id, gene_id, start, end, exon_number)
        - `simple_repeats` (id, sequence, copy_num, unit_size, etc.)
        - `genome_stats` (name, data_json) - The final "Answer" store.
3. **Execution Engine (Snakemake):**
    - Orchestrates the dependency graph.
    - Example: `calculate_gene_density.py` waits for `genes_loaded.done`.

---

3. Implementation Roadmap & Specs

✅ Chapter 2: DNA & Chromosomes (Complete)

- All base composition and repeat statistics are implemented.

🚧 Chapter 3: Genes (Active Development)

- **Core Data:** Genes and Exons are loaded.
- **Pending Implementation Specs:**
  - ☐ **Is Gene Density Uniform?**
    - *Algo:* Create a "Sliding Window" script. Divide each chromosome into 1Mb bins. Count gene `start_pos` falling into each bin. Calculate variance/stdev of counts.
  - ☐ **Do Gene-rich Chromosomes Have Smaller Genes?**
    - *Algo:* Correlation analysis. For each chromosome: (Total Gene Count / Chromosome Length) vs (Average Gene Length). Pearson correlation.
  - ☐ **How Are CpG Islands Associated with Genes?**
    - *Data:* Need to download `cpgIslandExt.txt` from UCSC.
    - *ETL:* Create `parse_cpg_islands.py` -> Table `cpg_islands`.
    - *Analysis:* Spatial join. Count how many CpG islands overlap with `genes` (start/end) vs non-gene regions.
  - ☐ **Do Large mRNAs Have Large UTRs?**
    - *Data:* GTF file already contains `UTR` features.
    - *ETL:* Need to update `parse_gtf.py` to also parse `feature == 'UTR'` (currently only genes/exons). Store in `utrs` table.
    - *Analysis:* Scatter plot data: Total Transcript Length vs Total UTR Length per gene.
  - ☐ **Which Genes Are Located in the Introns of Other Genes?**
    - *Algo:* Interval Tree or Nested Loop (optimized). Find Genes A where `A.start > B.start` and `A.end < B.end` and `B` is a different gene.

🔮 Future Chapters (Technical Plan)

Chapter 4: RNA

- **Key Data Source:** `rmsk.txt` (RepeatMasker) from UCSC.
  - *Why:* rRNA, tRNA, snRNA are often better annotated in the RepeatMasker track than the standard GTF for structural analysis.
- **Task:** Parse `tRNA` and `rRNA` families from RepeatMasker into a `non_coding_rnas` table.

Chapter 5: Proteins

- **Key Data Source:** Ensembl `Homo_sapiens.GRCh38.pep.all.fa` (Protein Sequences).
- **ETL:** New script `parse_proteins.py`.
  - *Schema:* `proteins` table (gene_id FK, protein_id, sequence, length).
- **Analysis:**
  - *Amino Acid Composition:* Counter(sequence).
  - *Motifs:* Regex search for specific patterns (e.g., Zinc Fingers) if requested.

Chapter 10: Comparative Genomics

- **Requirement:** Multi-genome support.
- **Refactor:** The database currently assumes 1 genome. We will need to add `genome_version` column to tables or use separate DB files (`mm10.db`, `hg38.db`) and attach them in SQLite for cross-querying.

---

4. Development Standards

1. **No Raw File Reads in Analysis:** Analysis scripts must *only* query the SQL database. If data is missing, write a Parser first.
2. **Idempotency:** All parsers must check `if exists` or handle `DROP TABLE` gracefully.
3. **Modular Snakefile:** Every script gets its own Rule. No bundling multiple scripts into one shell command.

Backend Coding Agent Skills

Skill 1: Data Engineering (The "Parse" Step)

Goal: Ingest raw bio-formats (GTF, BED, FASTA) into SQLite.

1. **Check Prerequisites:**
   - If the file is missing, instruct the user to download it (provide UCSC/Ensembl URL).
2. **Create Model:**
   - Define SQLAlchemy model in `backend/app/models/`.
   - *Critical:* Always use appropriate indexes on `start_pos`, `end_pos`, and `gene_id` for performance.
3. **Write Parser (`backend/scripts/`):**
   - Use `pandas` for CSV/BED/GTF files (chunksize=100k for memory safety).
   - Use `Bio.SeqIO` for FASTA.
   - **Optimization:** Convert DataFrames to dicts and use `db.bulk_insert_mappings()`. Never use `db.add()` in a loop.
4. **Register:**
   - Import the new model in `backend/fix_schema.py` to ensure tables are created.

Skill 2: Computational Analysis (The "Analyze" Step)

Goal: Answer a specific biological question using SQL + Python.

1. **Strategy Selection:**
   - *Simple Aggregation:* Use SQL (`SELECT gene_type, count(*) ...`).
   - *Complex Statistics:* Fetch data into Pandas (`pd.read_sql`), calculate in memory, save result.
   - *Genomic Interval Operations:*
     - **Problem:** "Find intersection of A and B".
     - **Solution:** Do not fetch all rows. Use SQL ranges: `WHERE A.end >= B.start AND A.start <= B.end`.
2. **Output Format:**
   - Save results as a JSON dictionary to `genome_stats` table using `upsert_statistic()`.
   - Structure the JSON to be self-descriptive (e.g., `{"average": 12.5, "min": 1, "distribution": {...}}`).

Skill 3: Pipeline Integration (Snakemake)

Goal: Automate the execution.

1. **Define Rules:**
    - Input: `script.py` AND `previous_step.done` (dependencies).
    - Output: `analysis/stats/script_name.done`.
2. **Dependencies:**
    - If a script queries `genes`, it **must** depend on `genes_loaded.done`.
    - If a script queries `proteins`, it **must** depend on `proteins_loaded.done`.

Skill 4: API Verification

Goal: Verify the data is accessible without a Frontend.

1. **Endpoint:** Ensure `backend/app/main.py` has a generic getter for statistics.
2. **Test:**
    - Run `curl http://127.0.0.1:8000/api/v1/statistics/{stat_name}`.
    - Verify the JSON structure matches the analysis script output.

Common Data Sources & Formats

- **Genes/Exons/UTRs:** `GTF` or `GFF3` (Ensembl).
- **Repeats/CpG/Conservation:** `BED` or `txt` (UCSC Table Browser).
- **Sequences:** `FASTA` (Ensembl).