Subject: Implementation Directives for Project Chimera

To: Code Generation AI

From: Suyash, System Architect

Objective

Your primary directive is to implement the **Project Chimera** system in Python. This system is designed to parse a modern web application's frontend codebase (React with TypeScript/JSX) into a structured JSON representation, allow for programmatic modification of this representation, and automate the build and deployment process.

You will be creating three core Python scripts: `surveyor.py` (The Parser), `foreman.py` (The Updater), and `pipeline.py` (The Automation Engine). Adherence to the specified logic and data schemas is critical for system integrity.

Directive 1: Implement The Parser (`surveyor.py`)

1.1. Role: This script's function is to translate the source code files into a single, master `project_data.json` file based on the schema defined in Directive 4. It also serves a secondary function as the verification engine.

1.2. Required Libraries:

- `os`: For file system traversal (walking directories).
- `json`: For writing the final JSON output.
- **A robust TSX/JSX-to-AST Parser:** You must use a library capable of parsing modern React syntax into an Abstract Syntax Tree. A Python wrapper around a mature JavaScript-based parser like Babel (`@babel/parser`) or Acorn is recommended.

1.3. Core Logic:

1. Implement a main function that accepts an optional file path argument. If no argument is given, it operates on the entire project. If a path is given, it operates only on that file (for the verification loop).
2. **Scan:** Recursively walk the `/client/src` directory to find all files ending in `.jsx` and `.tsx`.
3. **Parse to AST:** For each file, invoke the chosen AST parsing library to generate a complete, in-memory Abstract Syntax Tree.
4. **Translate to JSON:**
    - Traverse the AST. Your traversal logic must be designed to identify key React/JSX patterns (e.g., `JSXOpeningElement`, `JSXAttribute`, `JSXText`, `JSXExpressionContainer`).

    - Map the AST nodes to the custom JSON schema defined in **Directive 4**. Extract component names, element types, props (attributes), children, and source line numbers (`sourceLocation`).
    - Populate a master Python dictionary with this structured data, maintaining the file system hierarchy as specified in the schema.
5. **Output:** Write the master dictionary to `project_data.json` using the `json.dump()` method with an indent of `2` for human readability.

Directive 2: Implement The Updater (`foreman.py`)

2.1. Role: This script executes high-level commands to modify the codebase. It orchestrates the "propose-execute-verify" workflow.

2.2. Required Libraries:

- `argparse`: To parse the command-line arguments.
- **The same AST Parser/Generator libraries** used by `surveyor.py`.

2.3. Core Logic:

1. **Input:** The script must accept a `--command` string as a command-line argument.
2. **Parse Command:** Implement logic to parse this command to extract the target element `id`, the `property` to modify, and the `newValue`.
3. **Generate Intended State:** Load `project_data.json`. Create a deep copy of this data in memory. Apply the commanded change to this in-memory copy. This is now the **"intended state."**
4. **Generate Code Strings:** Using the AST library, generate two code strings: one representing the element's state *before* the change, and one *after*.
5. **Execute Find-and-Replace:**
    - Read the target source file into memory.
    - Perform a string replacement of the "old code" block with the "new code" block.
    - Save the modified content, overwriting the original file. This is the only direct file write operation.
6. **Initiate Verification:**
    - Immediately execute `surveyor.py` using `subprocess.run()`, passing the path of the modified file as an argument.
    - Capture the JSON output from this execution. This is the **"actual state."**
7. **Compare & Conclude:**
    - Perform a deep comparison between the "intended state" object and the "actual state" object.
    - If they match, call `pipeline.py` to proceed.
    - If they do not match, print a critical error to the console detailing the discrepancy and exit with a non-zero status code.

Directive 3: Implement The Automation Engine (`pipeline.py`)

3.1. Role: A simple script to execute shell commands for building and deploying the application.

3.2. Required Libraries:

- `subprocess`: To run external commands.

3.3. Core Logic:

1. **Build Step:**
    - Execute `npm run build` using `subprocess.run()`.
    - Set `shell=True` (or handle pathing correctly) and `check=True`. The `check=True` flag will automatically raise an exception if the command returns a non-zero exit code, halting the script on build failure.
2. **Deploy Step:**
    - If the build step completes without error, execute `firebase deploy` using `subprocess.run()`.
    - Ensure that the `stdout` and `stderr` of the commands are streamed to the console so the user can monitor the progress.

Directive 4: The Master JSON Schema (Blueprint)

This schema is the definitive data structure. The `surveyor.py` script **must** generate JSON that conforms to this structure, and the `foreman.py` script **must** be able to parse it.

```
{
  "projectName": "LuxeCraft",
  "rootDirectory": "client",
  "tree": {
    "type": "directory",
    "name": "src",
    "path": "client/src",
    "children": [
      {
        "type": "directory",
        "name": "components",
        "path": "client/src/components",
        "children": [
          {
            "type": "Component",
            "name": "HeroSection",
            "fileName": "HeroSection.jsx",
            "path": "client/src/components/HeroSection.jsx",
            "definition": {
              "rootElementType": "section",
              "elements": [
                {
```

```
              "id": "hero-main-heading",
              "type": "Heading",
              "sourceLocation": {
                "startLine": 41,
                "endLine": 47
              },
              "props": {
                "level": "h1",
                "className": "text-6xl...",
                "children": [
                  { "type": "text", "content": "Timeless" },
                  { "type": "span", "content": "Elegance" }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}
```


End of directives.