Certainly, let's summarize the key architectural decisions and propose a potential directory structure based on our discussion.

**Hexagonal Architecture Directory Structure:**

file\_processing\_framework/  
├── adapters/   
│ ├── file\_readers/  
│ │ ├── csv\_reader.py  
│ │ ├── json\_reader.py  
│ │ └── <other\_file\_readers>.py  
│ ├── file\_writers/  
│ │ ├── csv\_writer.py  
│ │ ├── json\_writer.py  
│ │ └── <other\_file\_writers>.py  
│ ├── database/  
│ │ ├── config\_repository.py   
│ ├── logger/  
│ │ ├── file\_logger.py   
│ │ ├── database\_logger.py   
│ │ └── <other\_loggers>.py   
├── domain/  
│ ├── entities/   
│ │ └── <YourEntityClass>.py   
│ ├── value\_objects/  
│ │ └── <YourValueObject>.py   
│ ├── services/  
│ │ ├── data\_cleaner.py  
│ │ ├── data\_transformer.py  
│ │ ├── data\_validator.py  
│ │ ├── data\_duplicator.py  
│ │ ├── stats\_calculator.py  
│ │ └── business\_rule\_engine.py  
├── infrastructure/  
│ ├── config/   
│ │ ├── config.yaml   
│ │ ├── config.json   
│ ├── utils/   
│ │ ├── helper\_functions.py   
├── application/  
│ ├── use\_cases/  
│ │ ├── process\_file\_use\_case.py  
│ ├── main.py

**Explanation:**

* **adapters:** This layer handles interactions with external systems.
  + **file\_readers:** Contains classes for reading data from different file formats (CSV, JSON, etc.).
  + **file\_writers:** Contains classes for writing data to different file formats.
  + **database:** Contains classes for interacting with the database (e.g., config\_repository).
  + **logger:** Contains classes for logging events and metrics (e.g., file\_logger, database\_logger).
* **domain:** This layer contains the core business logic, independent of any external concerns.
  + **entities:** Represents core business objects (e.g., Customer, Order).
  + **value\_objects:** Represents immutable values with their own validation logic (e.g., Name, Email).
  + **services:** Contains classes for specific business operations (e.g., data\_cleaner, data\_transformer, data\_validator).
* **infrastructure:** This layer provides supporting utilities.
  + **config:** Contains configuration files (YAML, JSON).
  + **utils:** Contains helper functions and utilities used throughout the application.
* **application:** This layer orchestrates the flow of data and coordinates interactions between other layers.
  + **use\_cases:** Contains classes that define the use cases of the application (e.g., process\_file\_use\_case).
  + **main:** The entry point of the application.

**Key Points:**

* This structure promotes loose coupling between different parts of the application.
* The core business logic is isolated within the "domain" layer.
* Adapters handle interactions with external systems, making it easier to switch between different technologies.
* The "infrastructure" layer provides supporting utilities and configuration management.
* The "application" layer coordinates the flow of data and orchestrates the use cases.

**Note:**

* This is a suggested structure, and you can adapt it based on your specific needs and preferences.
* Remember to follow the principles of Hexagonal Architecture, ensuring that the core domain logic remains independent of external concerns.

I hope this provides a good starting point for your file processing framework. If you have any further questions or want to delve deeper into specific aspects of the architecture, feel free to ask!