Certainly, let's illustrate how these components interact with each other using a simple example: **Processing a CSV file and writing the transformed data to another CSV file.**

**1. adapters/file\_readers/csv\_reader.py**

Python

class CSVReader:  
 def read(self, file\_path: str) -> list[dict]:  
 """Reads data from a CSV file and returns a list of dictionaries."""  
 # ... implementation for reading CSV data ...  
 return data

**2. domain/services/data\_transformer.py**

Python

class DataTransformer:  
 def transform(self, data: list[dict]) -> list[dict]:  
 """Transforms the input data according to business rules."""  
 # ... implementation for data transformation (e.g., adding a new column, modifying existing values) ...  
 return transformed\_data

**3. adapters/file\_writers/csv\_writer.py**

Python

class CSVWriter:  
 def write(self, data: list[dict], file\_path: str) -> None:  
 """Writes data to a CSV file."""  
 # ... implementation for writing data to CSV file ...

**4. application/use\_cases/process\_file\_use\_case.py**

Python

from adapters.file\_readers.csv\_reader import CSVReader  
from domain.services.data\_transformer import DataTransformer  
from adapters.file\_writers.csv\_writer import CSVWriter  
  
class ProcessFileUseCase:  
 def \_\_init\_\_(self, csv\_reader: CSVReader, data\_transformer: DataTransformer, csv\_writer: CSVWriter):  
 self.csv\_reader = csv\_reader  
 self.data\_transformer = data\_transformer  
 self.csv\_writer = csv\_writer  
  
 def execute(self, input\_file\_path: str, output\_file\_path: str) -> None:  
 data = self.csv\_reader.read(input\_file\_path)  
 transformed\_data = self.data\_transformer.transform(data)  
 self.csv\_writer.write(transformed\_data, output\_file\_path)

**5. application/main.py**

Python

from application.use\_cases.process\_file\_use\_case import ProcessFileUseCase  
  
def main():  
 csv\_reader = CSVReader()  
 data\_transformer = DataTransformer()  
 csv\_writer = CSVWriter()  
 use\_case = ProcessFileUseCase(csv\_reader, data\_transformer, csv\_writer)  
 use\_case.execute("input.csv", "output.csv")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Explanation:**

* **main.py:** This is the entry point of the application. It creates instances of the necessary components (reader, transformer, writer) and initializes the ProcessFileUseCase.
* **ProcessFileUseCase:** This class orchestrates the flow of data. It calls the read() method of the CSVReader to read data from the input file, then calls the transform() method of the DataTransformer to process the data, and finally calls the write() method of the CSVWriter to write the transformed data to the output file.
* **Adapters:** The CSVReader and CSVWriter are adapters that handle the interaction with the CSV files.
* **Domain:** The DataTransformer contains the core business logic for transforming the data.

This simplified example demonstrates how the different components interact within the Hexagonal Architecture. The use\_case acts as a coordinator, and the adapters handle interactions with external systems. The core business logic resides within the "domain" layer, keeping it independent of external concerns.

I hope this provides a clearer understanding of the component interactions within your file processing framework!