CDAP Wrangler – Feature Enhancement Summary

1. Repository Setup

- Begin by forking the Wrangler GitHub repo.
- Clone your fork and navigate to the local project directory for development.

---

2. Grammar Extension (`Directives.g4`)

- Introduced new **lexer tokens** to represent:
    - `BYTE_UNIT` (e.g., KB, MB, GB)
    - `TIME_UNIT` (e.g., ms, s, min)
- Added new **parser rules**:
    - `byteSizeArg` for byte expressions
    - `timeDurationArg` for time values
- Regenerate the ANTLR parser using Maven:

```
Unset
mvn clean compile
```

---

3. API Layer Enhancements (wrangler-api)

- Implemented two utility classes inside `parser`:
    - `ByteSize.java` – parses strings like "2MB" into bytes

    - `TimeDuration.java` – converts "5s", "300ms" into milliseconds

- Modified `TokenType.java` to register new token types: `BYTE_SIZE` and `TIME_DURATION`

---

4. Core Parser Modifications (wrangler-core)

- Updated the **visitor pattern implementation** to support the two new token types.
- When matched, corresponding Java classes (`ByteSize`, `TimeDuration`) are instantiated and returned.

---

5. New Directive: `AggregateStats`

- Developed a new directive class named `AggregateStats`.
- This directive:
    - Accepts two columns: one with data sizes and one with durations
    - Converts and aggregates values
    - Produces a final result with:
        - Total size in MB
        - Total time in seconds

---

6. Testing

- **Unit Tests** cover:
    - ByteSize and TimeDuration parsing for various formats and edge cases.

- **Integration Test**:
    - Executes the `AggregateStats` directive on sample data rows to validate the aggregated output.

---

7. AI Involvement (`prompts.txt`)

- Prompts used during AI-based coding assistance (e.g., logic generation or bug fixes) should be added here.

---

8. Documentation (`README.md`)

- Updated the README with:
    - How to use the new directive
    - List of accepted formats (e.g., "10KB", "2s")
    - Code samples

Code Illustrations

`ByteSize.java`

```java
public class ByteSize extends Token {
    private long bytes;

    public ByteSize(String value) {
        if (value.endsWith("KB"))
            bytes = (long)
(Double.parseDouble(value.replace("KB", "")) * 1024);
```

```
        else if (value.endsWith("MB"))
            bytes = (long)
(Double.parseDouble(value.replace("MB", "")) * 1024 * 1024);
        else if (value.endsWith("GB"))
            bytes = (long)
(Double.parseDouble(value.replace("GB", "")) * 1024 * 1024 *
1024);
        // Add more cases as needed
    }

    public long getBytes() {
        return bytes;
    }
}
```

TimeDuration.java

Java

```java
public class TimeDuration extends Token {
    private long milliseconds;

    public TimeDuration(String value) {
        if (value.endsWith("ms"))
            milliseconds = Long.parseLong(value.replace("ms",
""));
        else if (value.endsWith("s"))
            milliseconds = Long.parseLong(value.replace("s", ""))
* 1000;
        // Extend as needed
    }

    public long getMilliseconds() {
        return milliseconds;
    }
```

```
}
```

AggregateStats.java

Java

```
public class AggregateStats implements Directive {
    private String byteCol, timeCol, sizeOutCol, timeOutCol;

    public List<Row> execute(List<Row> rows, ExecutorContext ctx)
{
        long totalBytes = 0, totalTime = 0;

        for (Row row : rows) {
            totalBytes += ((ByteSize)
row.getValue(byteCol)).getBytes();
            totalTime += ((TimeDuration)
row.getValue(timeCol)).getMilliseconds();
        }

        List<Row> output = new ArrayList<>();
        output.add(new Row()
            .add(sizeOutCol, totalBytes / (1024.0 * 1024))
            .add(timeOutCol, totalTime / 1000.0));
        return output;
    }
}
```

Unit Tests

Java

```
@Test
public void testByteSizeParsing() {
    ByteSize size = new ByteSize("10KB");
    assertEquals(10240, size.getBytes());

    size = new ByteSize("1.5MB");
    assertEquals(1572864, size.getBytes());
}

@Test
public void testTimeDurationParsing() {
    TimeDuration time = new TimeDuration("200ms");
    assertEquals(200, time.getMilliseconds());

    time = new TimeDuration("2s");
    assertEquals(2000, time.getMilliseconds());
}

@Test
public void testAggregateStatsDirective() {
    List<Row> rows = Arrays.asList(
        new Row().add("data_transfer_size", new
ByteSize("2MB")).add("response_time", new TimeDuration("1s")),
        new Row().add("data_transfer_size", new
ByteSize("3MB")).add("response_time", new TimeDuration("2s"))
    );

    List<Row> result = new AggregateStats().execute(rows, null);
    assertEquals(1, result.size());
    assertEquals(5.0, result.get(0).getValue("total_size_mb"),
0.001);
    assertEquals(3.0, result.get(0).getValue("total_time_sec"),
0.001);
```

```
}
```