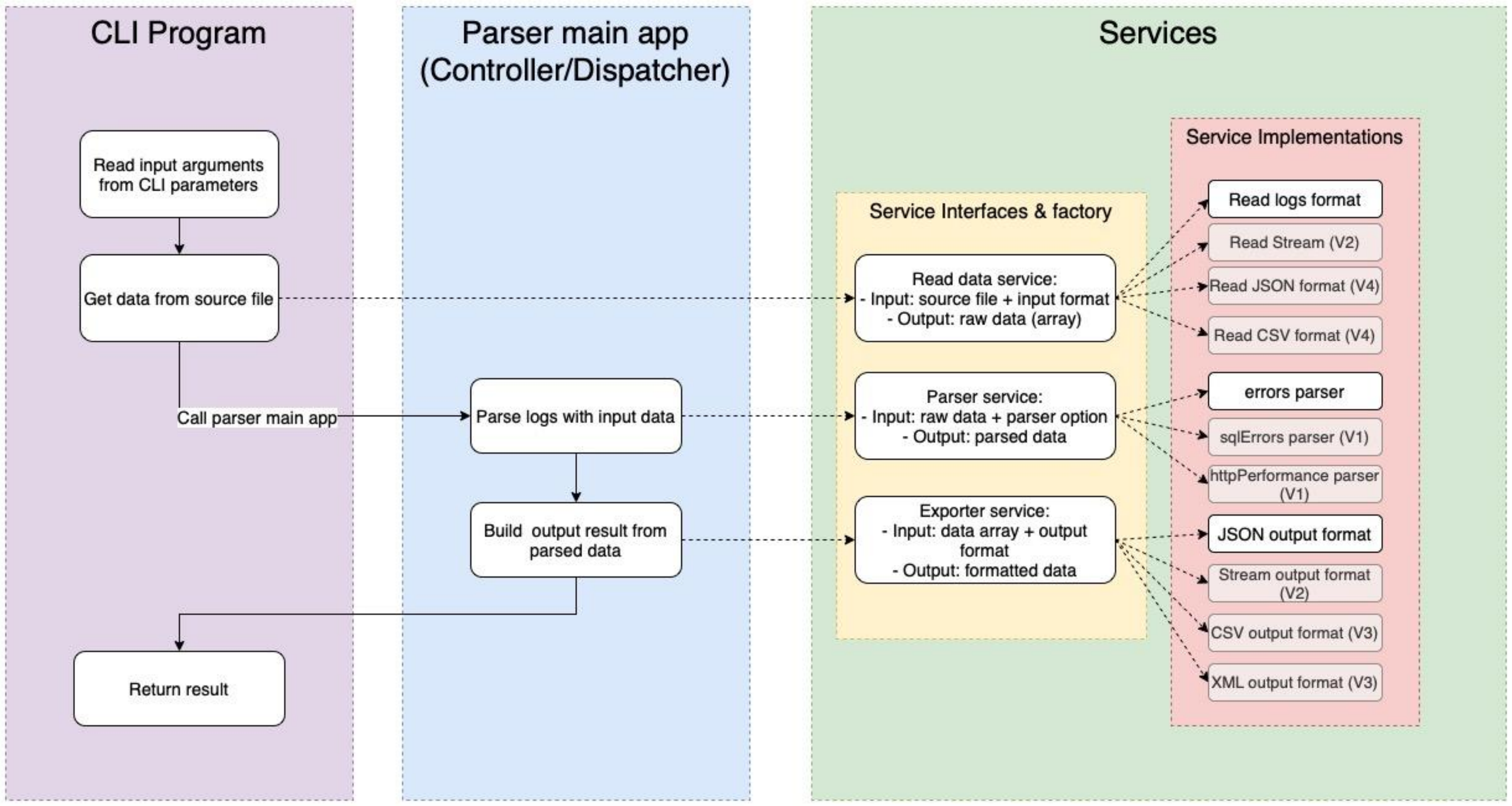
High-level Architecture

This document contains a high-level architecture overview for Log Parser application.

Overview architecture


CLI Program
Read input arguments from CLI parameters
Get data from source file
Call parser main app
Return result
Parser main app (Controller/Dispatcher)
Parse logs with input data
Build output result from parsed data
Services
Service Interfaces & factory
Read data service:
- Input: source file + input format
- Output: raw data (array)
Parser service:
- Input: raw data + parser option
- Output: parsed data
Exporter service:
- Input: data array + output format
- Output: formatted data
Service Implementations
Read logs format
Read Stream (V2)
Read JSON format (V4)
Read CSV format (V4)
errors parser
sqlErrors parser (V1)
httpPerformance parser (V1)
JSON output format
Stream output format (V2)
CSV output format (V3)
XML output format (V3)

Component layers

CLI Program

Entry point of the application, accept arguments from CLI and prepare the parameters for the main application.

Arguments:

```
--input                 input file path
--output                output file path
--parser                content of interest for parser (V1)
```

```
--formatter          format of output file (V3)
--logFormat          format of input log file (V4)
```

Parser main app

Work as Controller/Dispatcher that validates parameters, calls corresponding services and returns data as result.

Parameters:
- `data`: input raw data that will be parsed
- `output`: output file path
- `parser`: content of interest for parser
- `formatter`: format of output file

Response:
- Output file as specified format & file path, with parsed data as content.

Services

Include all services that are used in application. Each service has an interface for controllers to call, and a factory that init the implementations based on constructor parameters.

Read Data Service

Parameters:
- `input`: input file path
- `logFormat`: format of input log file

Response:
- Array of raw data

Processing:
Initiate (factory) implementation class from `logFormat` and call the service function with `input` for data reading. Add a default implementation of `"log"` format for the MVP version.

Parser Service

Parameters:
- `data`: array of raw data to be parsed
- `parser`: content of interest for parser

Responses:
- Array of parsed data

Processing:

Initiate (factory) implementation class from `parser` and call the service function with `data` for parsing. Add a default implementation of `"errors"` parser for the MVP version.

Exporter Service

Parameters:

- data: array of data to be exported
- formatter: format of output file

Responses:

- Output file with given data in specified format

Processing:

Initiate (factory) implementation class from `formatter` and call the service function with `data` for parsing. Add a default implementation of `"json"` parser for the MVP version.

Implementation plan

V1

- Support `parser` argument from CLI program
- Add Parser Service implementation for `"sqlErrors"` and `"httpPerformance"`

V2

- Support empty `input` and `output` argument from CLI program and recognize them as `"stream"`
- Add Read Data Service implementation for `"stream"`
- Add Exporter Service implementation for `"stream"`

V3

- Support `formatter` argument from CLI program
- Add Exporter Service implementation for `"xml"` and “csv”

V4

- Support logFormat argument from CLI program
- Add Read Data Service implementation for `"json"` and `"csv"`