Transport workflow

At Ferovinum we transport stock from one warehouse location to another on clients behalf. Your task is to implement a simplified transport workflow described by the sequence diagram on the next page.

A 3rd party server implementation is provided to represent 3rd party API behaviors of Carrier and Warehouses. Details of this server are found after the sequence diagram.

You will create a simple HTTP server that interacts with the 3rd party server via HTTP to demonstrate your understanding of HTTP request / response, asynchronous tasks and polling.

You can write this server in any language you choose, we recommend Javascript, Python as they require minimal boilerplate.

Your Server Specification

Your server needs to support a single POST endpoint `/request-transport`

It should have a request body containing all parameters needed to complete the transport workflow. No parameters should be hard coded and you can expect multiple requests to be sent to this endpoint at the same time when we test your work..

The end result should be 2 files written to the file system for each request. File names and their content structures are:

File 1: `collection-confirmation-<jobId>.json`
(where `<jobId>` is generated by the 3rd party server)

```
{
  productsId: string
  quantity: number
  collectionTime: string
}
```

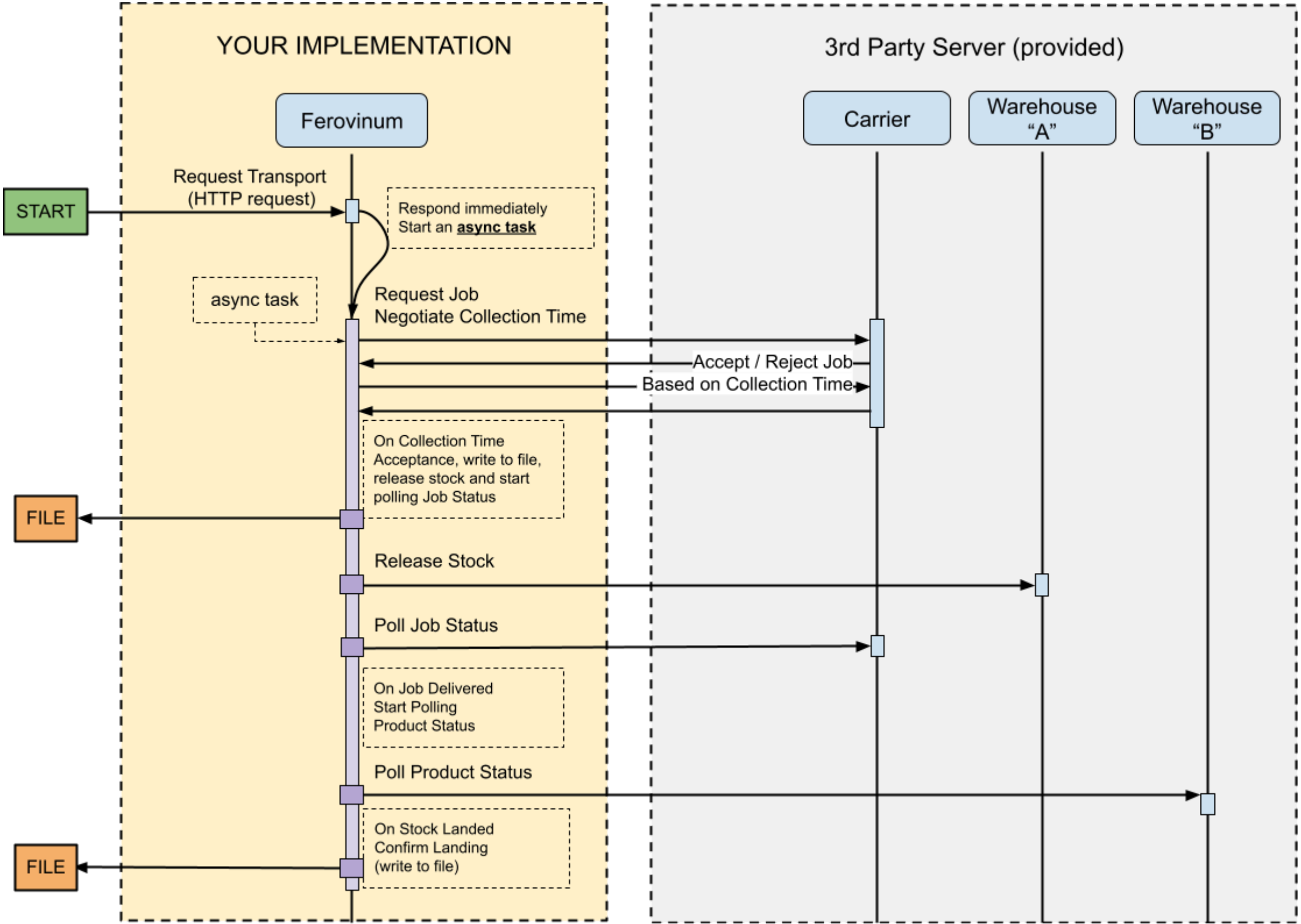
File 2: `landing-confirmation-<warehouseId>-<productId>.json`
(where `warehouseId`, `productId` and `quantity` are part of the request sent to your endpoint)

```
{
  quantity: number
}
```

The 3rd party server will also generate expected confirmation files to help you verify your work.

Hints:

- It may be helpful to implement a “ThirdPartyClient” to handle interactions with the 3rd party server..
- Feel free to reverse engineer the 3rd party server code to understand how to interact with it. You can insert additional logs or hard code certain conditions to help you test individual interactions.
- You can use CURL, Postman or other tools to send test requests to your server.
- Feel free to use any AI tools such as Copilot and ChatGPT.


YOUR IMPLEMENTATION
3rd Party Server (provided)
Ferovinum
Carrier
Warehouse "A"
Warehouse "B"
START
Request Transport (HTTP request)
Respond immediately
Start an async task
async task
Request Job
Negotiate Collection Time
Accept / Reject Job
Based on Collection Time
On Collection Time Acceptance, write to file, release stock and start polling Job Status
FILE
Release Stock
Poll Job Status
On Job Delivered Start Polling Product Status
Poll Product Status
On Stock Landed Confirm Landing (write to file)
FILE

3rd Party Server

The server is designed to assist you in completing this task. Implemented in NodeJS, it will log helpful messages to assist in your work.

```
npm install
node 3rd-party-server.js
```

1. POST carrier/request-job

Request carrier to arrange transportation from origin to destination, with a specific collection time.

Accepts payload with the following schema:

```
{
  clientId: string // can be any string
  productId: string
  quantity: number
  origin: 'A' | 'B' // warehouse ID
  destination: 'A' | 'B' // warehouse ID
  collectionTime: string // in DD/MM/YYYY HH:MM:SS format
}
```

Possible return values

- `{ status: "ACCEPT", jobId: string, collectionTime: string }`
- `{ status: "REJECT" }` `// if delivery time is not suitable`
- `{ status: "ERROR", error: string }` `// if input is invalid`

> NOTE: In order to find a suitable collection time, you should start with a collectionTime from 9am the next day and increment it by an hour each time it is rejected. You can assume there is always a suitable time on the following day. The following day can be a weekend or a holiday, we are not factoring in working days in this exercise.

2. GET carrier/job/#job-id/status

Query carrier job status for a given job id.

Possible return values

- `{ status: "PENDING" }`
- `{ status: "COLLECTION FAILED" }` `// if product was not "released" from the warehouse`
- `{ status: "DELIVERED" }`
- `{ status: "NOT FOUND" }`

NOTE: You should poll this endpoint once a second to monitor the status, the 3rd party server is designed to change status within seconds when conditions are met.

3. POST warehouse/#warehouse-id/release

Release product from the warehouse so the carrier can collect the stock, without this call, the carrier will return a **"COLLECTION FAILED"** status.

Accepts payload with the following schema:

```
{
  productId: string
  quantity: number
  collectionTime: string // in DD/MM/YYYY HH:MM:SS format
}
```

NOTE: The product id, quantity and collection time must match the job details.

4. GET warehouse/#warehouse-id/product/#product-id/status

Query product status in the warehouse
Possible return values

- **{ status: "NOT FOUND" }**
- **{ status: "NOT LANDED" }**
- **{ status: "LANDED", quantity }**

NOTE: You should poll this endpoint every second or so to monitor the status