Code Challenge: Funds Transfers

Context

You are tasked with implementing an application capable of transferring funds between two savings accounts based on some business rules. Please read the instructions below, and feel free to ask for clarifications if needed.

General notes

- You should submit your solution source code to us as a compressed file containing the code and documentation. Please do not to include unnecessary files such as compiled binaries and libraries;
- Do not upload your solution to public repositories in GitLab, BitBucket, GitHub, etc

Our expectations

We at YellowPepper value **working, elegant code that follows best practices over complex code**. The solution should reflect it.

Your solution is expected to have **quality**, be **maintainable** and **extensible**.

Delivery

Provide a document with descriptions of relevant code design choices.

The application must be runnable as a stand-alone or in a docker container. It must be written in Java with Spring Framework.

Feel free to use any open source library you deem appropriate.

Operation

The solution must be based on microservices architecture, and expose the RESTful APIs that are necessary to achieve the objective.

If the `origin_account` does not have enough funds to execute the transfer for the requested value, add an `insufficient-funds` error.

If the daily limits are reached, an error should be added: `limit_exceeded`.

At the end of the transfer operation, the `origin_account` must reflect the new available balance as well as the `destination_account` with the increased value of the transfer.

** Here are some APIs that you will need. Please define any other additional APIs or Services if you see necessary to complete the task. **

Input/Output of Transfer Service

Your program should accept a `json` object as input and output a serializable object. Example:

```
input:
    {
        "amount": 5000,
        "currency": "USD",
        "origin_account": "12345600",
        "destination_account": "12345601",
        "description": "Hey dude! I am sending you the money you loaned to me last
week."
    }

output:
    {
        "status": "OK",
        "errors": [],
        "tax_collected": 50.00,
        "CAD": 66,928861615
    }
```

Input/Output of Account Service

Your program should accept a `json` object as input and output a serializable object. Example:

```
input:
    {
        "account": "12345600"
    }

output:
    {
        "status": "OK",
        "errors": [],
        "account_balance": 70000.00
    }
```

Business rules

- Accounts cannot have negative balances.
- A source account can only make 3 transfers per day.

- The only currency supported at the moment is dollars, but this could change in the future.
- The origin account will be charged with a tax corresponding to 0.5% in case the amount of the transfer is greater than $ 100USD or 0.2% otherwise
- Use the public API https://api.exchangeratesapi.io/latest?base=USD to add rate exchange for CAD currency

Examples

```
input:
    {
        "amount": 5000,
        "currency": "USD",
        "origin_account": "12345600",
        "destination_account": "12345601",
        "description": "Loan Pay",
    }

output:
    {
        "status": "OK",
        "errors": [],
        "tax_collected": 50.00
        "CAD": 66,928861615
    }
```

```
input:
    {
        "amount": 1000,
        "currency": "USD",
        "origin_account": "12345601",
        "destination_account": "12345600",
        "description": "hi"
    }

output:
    {
        "status": "ERROR",
        "errors": ["insufficient-funds"],
        "tax_collected": 0.00
    }
```

```
input:
    {
        "amount": 10,
        "currency": "USD",
        "origin_account": "12345603",
        "destination_account": "12345604",
        "description": "hi"
    }

output:
    {
        "status": "ERROR",
        "errors": ["limit_exceeded"],
        "tax_collected": 0.00
    }
```

Error handling

Please consider using common patterns to propagate errors and exceptions within your business logic.

Persistence Layer

Please use any database that can be embedded with Spring Boot as H2 Database. Make sure you have a mechanism to preload some basic test scenarios.