Background: This is a practical data-engineering assignment within the recruiting process for the Data Engineer position. The practical assignment is prepared by the applicant beforehand and is presented during the technical (second) interview. The solution serves as a basis and an inspiration for further discussions during the technical interview.

Goal:

- get insights into how the candidate approaches and solves a problem
- see the working style of the applicant
- get an impression about how much attention to detail the applicant has
- see how the candidate presents the results

Preamble: There is no “right” or “wrong” when presenting your technical solution. It’s not expected that you have a fancy presentation - you can also present the solution at the command line, if it is your strength. Every human is different - someone is better in presenting, explaining and understanding users; someone is better in problem-solving, programming and writing beautiful code. Everyone has strong and weak sides. Show us your strong sides! **We** want to understand how you work and behave in a professional environment - and if you can think out of the box. So, **please,** do it your way and within your personal style! But to be clear: *A Jupyter Notebook is not the best tool for these **data engineering** tasks!*
Last but not least: *Try to prepare it like a Software Project that a colleague, whom you really like, will take over without any doubts or fears.*

E1 - Use external Exchange Rate API and load exchange rates into a database

Goal:

- implement the API and show how you downloaded the data (any observations, pitfalls?)
- implement, show and explain the data model you created
- be prepared to show that you can convert multiple source (target) prices from an SQL query result (see SQL “data example” below), into each required base price

Background - Use case:

- imagine you have a dashboard with currency-related numbers
- you have different users for that dashboard: All local dealers (e.g. Russia, Europe, US), but also the boss of that dealers
- All use the same dashboards, but want to see within their region their own numbers (the numbers are in the original values and (target) currency stored) in RUB, EUR, USD, etc.
- Their Bosses want to see over ALL locations the numbers in EUR or USD
- To-Do: sketch out a (data) solution, that would solve this problem (do NOT build the dashboard, do not build another API, instead build the load into a (data) storage to get all the raw data for calculation, sketch out the problem and show a solution (e.g. with SQL) how you calculate on the fly into the two base currencies EUR or USD from some dummy data (e.g. raw data example SELECT below) from RUB, GBP whatever)

How-to:

- at exchangeratesapi.io you find an exchange API for currency rates on a daily basis
- load all available exchanges for the current year into a database

- ensure that you can easily convert multiple source (target currency) prices into at least 2 base currencies: USD or EUR, as example target currency data as PostgreSQL that needs to be calculated (Hint: Expected best results are two nearly identical SQL's where you do a translation in each execution either [EUR|USD]) for the following example target data):

```
SELECT CAST('2021-01-01' AS DATE) AS date_date, CAST('EUR' AS CHAR(3)) AS target_currency, CAST('11.11' AS
NUMERIC(14,4)) AS a_price UNION
SELECT CAST('2021-02-01' AS DATE) AS date_date, CAST('GBP' AS CHAR(3)) AS target_currency, CAST('12.12' AS
NUMERIC(14,4)) AS a_price UNION
SELECT CAST('2021-02-08' AS DATE) AS date_date, CAST('RUB' AS CHAR(3)) AS target_currency, CAST('333.33' AS
NUMERIC(14,4)) AS a_price;
```

Or as tabulated result for EUR (example, not real numbers):

```
| date_date  | target_currency | a_price   | base_currency | converted_price |
| 2021-02-01 | GBP             | 12.1200   | EUR           | 14.1510         |
| 2021-02-08 | RUB             | 333.3300  | EUR           | 3.8468          |
| 2021-01-01 | EUR             | 11.1100   | EUR           | 11.1100         |
```

E2 - crawling, data extraction of a static website

- A user asks about how he can obtain data of a website without spending days extracting the data by hand
- Look at https://www.urparts.com/index.cfm/page/catalogue and as an example for the overall structure: check for a manufacturer, then look into the categories, then the models and the result set where the part (number) before ‘ - ‘ and the part category
- crawl the ***whole*** at this point in time existing catalogue, for all manufacturers, and create a CSV file that will look like this (example of the first 10 lines of a complete output):

```
manufacturer,category,model,part,part_category
Ammann,Roller Parts,ASC100,ND011710,LEFT COVER
Ammann,Roller Parts,ASC100,ND011758,VIBRATOR
Ammann,Roller Parts,ASC100,ND011785,RIGHT COVER
Ammann,Roller Parts,ASC100,ND017673,ECCENTRIC
Ammann,Roller Parts,ASC100,ND017675,ECCENTRIC
Ammann,Roller Parts,ASC100,ND018184,HUB
Ammann,Roller Parts,ASC100,ND018193,BRACKET
Ammann,Roller Parts,ASC100,ND018214,LEFT SHAFT
Ammann,Roller Parts,ASC100,ND018216,RIGHT SHAFT
```

E3 - write a unit test

Write at least one simple unit test for the code that you developed in E1 or E2 with any framework you want!

Hint: It should be a test written or at least explained in a way that runs on your local dev and (!potential) remote test env. (e.g. consider “real” unit test or if not a pure unit then a “mocked” integration test)

Next Hint: You also need this to be better prepared and understand the pairing session during the technical interview where we iterate into real unit testing.