API

The Fortune-Teller API is at the core of all Fortune-Teller functionality. While the App provides a means to guide users through the functionality and provide an attractive User Interface, the API should be doing the work. Deeper logic should be limited to the API. The API is hosted as a GitHub Project and based on Ruby on Rails. It can be installed separately and the architecture considers key settings to be saved in files rather than database entries. However, the API is also deployed on a server under api.fortune-teller.me. Within the API there are two major parts: the public part with no intention of saving requests or calculations and the registered part.

API Concepts

General Scope and Goals

Following the general goals as published on the Project Website, the API intends to provide easy access to calculations and information relevant for personal finance and financial planning of individuals. Any extension of the app with further modules / functionalities is welcome - as long as the core principles: Non-Commercial, Independence, Ease of Use, Privacy are maintained. Go to GitHub for the project site.

The public part of the API covers basic functionalities that can be queried without a UID (registration string) and that (besides server logs) do not save data.

The registered part of the API is set up in a way to avoid obtaining personal data to the highest degree possible. Fortune-Teller does not intend to monetize data or proactively contact users, but in order to provide a better user experience, value flows and entries are saved in this working mode.

International Scope: At the time of intiation, the API provides some generic functionalisties and specific functionalities for the legal setup in Germany as of 2021. The architecture considers internationalization / country localization.

Outlook on Functionality

- Public API
  - Generic
    - Inflation on named line items
    - Interest-Calculator
    - Valueflows for Financial items over time (interest, fees, tax)
  - DE-specific
    - Income Tax Tariff
    - Pension-Calculator (DRV)
    - Social Insurance rates
- Registered API (Generic functions with localization for certain items)
  - UID management (UID, Pin, token management, key calculation information)
  - Spending at given timeframe (named and sorted line items)
  - Assets at given timeframe
  - Debt & Valueflows
  - Savings Valueflows
  - Summary Cashflow

Public API

Registered API

JSON-Calculation Scheme

This concept is used in a number of calculation formats in the Fortune-Teller API.

Generally speaking, a JSON-File, that is committed in the GIT project contains a calculation structure, that is to be applied on a base value. The structure provided in the JSON-File provides the raw-data for the calculation scheme, so that it is highly adjustable, without impacting the APIs code.

API calculation engine

The scheme calculator is called with Localization (i.e. DE), with the calculation type, a base value and a number of variables. The input items are checked, the versions are identified and errors / alternatives raised if not available. As a result the calculation is performed and a number of labelled items and the values are given back, along with messages raised, disclaimer and sources.

Step Elements	Possible Values	Example
type	Absolute *if range is met, this value or the referenced labelvalue is added* Percent *a percentage is applied on an amount in the range* StepPercent *a percentage is applied for each part of the range that is below the base value* Add *if range is met, the label value is added* Multiply *if range is met, the label value is added*	
base	label id *any label available as input or in the calculation*	
from, to	Decimal value *lower and upper range of applicable base value. Range includes these values.*	
part	Decimal value of 1 or below *applied on the range of the base amount (only percent or absolute amount)*	
label	label id *result is cumulatively saved under this label*	
var	Decimal value *i.e. an amount or a decimal below 1 for a rate to be applied.*	

labelvar	label id *any label available as input or in the calculation*	

Core Elements of the JSON file (in folder jsonlib)

1. The name of the file is defined in the respective module, but begins with the Country Code (2-Letter)
2. The Json begins with a context depending on the scope of the settings included, any number of generic items can be listed.
3. The Input section declares expected input variables and whether these are obligatory
4. The actual type of calculations is named and provides a collection of versions
5. Each version includes a collection of steps, that will be followed consecutively
6. Each calculation scheme ends with a Disclaimer and Source tag, this must contain any limitations to the accuracy of the calculation and the source of the information included in the scheme.

Example for a file (not making a lot of sense)

```
{
  "country":"Deutschland",
  "comment1": "Order versions from recent to old, followed by Disclaimer",
  "input":
  □{□
    □[
□□□"label":"basevalue",
            "obligatory":"no"
□□],
        [
□□□"label":"prepayment",
            "obligatory":"yes"
□□]
  □},
  "income":
  □{□
□□"2021": [
□□□□□...
□□□□],
□□"2020": [
□□□□□{
□□□□□□"type":"absolute",
                        "base":"value",
```

```
        "from":"0",
        "to":"1200",
                        "part":"1",
        "label":"gez",
        "var":"120"
      },
      {
        "type":"percent",
                        "base":"value",
        "from":"0",
        "to":"1200",
                        "part":"1",
        "label":"est",
        "var":"0.12"
      },
      {
        "type":"percent",
                        "base":"est",
        "from":"0",
        "to":"999999999",
                        "part":"1",
        "label":"soli",
        "var":"0.05"
      }
    ],
  "Disclaimer":"Dieser Text erklärt Besonderheiten",
        "Source":"EStG, Dejure.org 31.7.2021"
  }
    }
 }
```