Introduction

Brief introduction:

Regestum will be an application where people will be able to use it as a registry for their expenses. They also be able to set a desirable amount of saves to achieve in a period of time and that way control their expenses.

A brief list of features:

> Save data on server (REST API)
> Quick understand of layout (good UI and UX)
> Mobile-first (Web with RWD similar aspect as a native)
> Simple and worryless use (No banks and cloud account sync)

Design and Implementation

The REST API Specification

+ Create, modify, save, and delete their accounts.
+ Record, modify, and delete their payments and expenses.
+ Set a period of time or instance to record the payments.
+ Set a desired amount of savings to achieve at end of period of time.

Endpoints

+ /account (with validation)
+ /saves (taking the id of account, else returns back to /account)
+ /saves/:saveId
+ /periods (taking the id of account, save, else returns the list of saves)
+ /periods/:periodId
+ /entries (taking the id of account, save, and period, else returns the list of saves)
+ /entries/:entryId
+ /expenses (taking the id of account, save, and period, else returns the list of saves)
+ /expenses/:expenseId

Database Schemas, Design and Structure, and Communication

```
+ Entries Schema: /entries (array)  >> Click >> /entries/:entryId

        - amount: String, required          //Entry amount

        - date: Date, default               //Date that the entry has been done

        - comment: String, default          //Some comment or explanation for the entry


+ Expenses Schema: /expenses (array) (taking the id of account, save, and period) >>
Click >> /expenses/:expenseId

        - place: String, default            //Place or shop where the expense has been done

        - amount: Number, required          //Expense amount

        - date: Date, default               //Date that the expense has been done

        - comment: String, default          //Some comment or explanation for the expense


+ Periods Schema: /periods (array)  >> Click >> /periods/:periodId       //All completed
periods and current periods that each onw shows its currentSave, if the desired save has
been achieved or not (show the resultant currentSave), its total entries and total
expenses
```

```
        - startDate: Date, required          //The star of the period of time

        - endDate: Date, required            //The end of the period of time

        - desiredSave: Number, required      //How much I wish to save in a period of time

        - availableMoney:  =( savesModel.totalSaves - desiredTotalSave )
        //The available money that I have at the moment

        - currentSave: Number, required => =(desiredSave + availableMoney)
        //The save that I have (including the desired save) Restart in each new period

        - expenses: sum([expensesModel].amount) >> Click >> /expenses (array) >> Click >>
/expenses/:expenseId
        //All expenses done during a period

        - entries: sum([entriesModel].amount) >> Click >> /entries (array) >> Click >>
/entries/:entryId
        //All entries done during a period

        - comment: String, default           //Some comment or explanation or planning for the
period


+ Saves Schema: /saves (array) (taking the id of account) >> Click >> /saves/:saveId

        - totalSaves: Number, default => =( totalSaves - (lastPeriod).expenses.last ) ||
=(totalSaves + (lastPeriod).entries.last)
        //The total save that I have collected during all periods until now (add and extract each new
money transaction)

        - desiredTotalSave: sum([periodsModel].desiredSave)
        //The desired savings that I should have been collected during all periods.

        - periods: [periodsModel].count >> Click >> /periods (array) >> Click >>
/periods/:periodId
        //The periods quantity in the account

        - comment: String, default           //Some comment or explanation or planning for the
save


+ User Account Schema: /account (with validation)

        - name: String, required             //The user name of the account

        - password: String, required         //The password to log in the account

        - image: String, default             //The img of the account (just for customize)

        - saves: Number, default => [savesModel].count >> Click >> /saves
        //The saves instances of the account

        - availableMoney: Number, default => savesModel.(lastPeriod).availableMoney
        >> Click >> /periods/:periodId
        //The available money that I have at the moment in the current period

        - totalSaves: Number, default => savesModel.totalSaves >> Click >> /saves/:saveId
        //The total save that I have collected during all periods until now

        - totalExpenses: Number, default => savesModel.(lastPeriod).expenses >> Click >>
/periods/:periodId
        //The total money amount expended at the moment in the period

        - totalEntries: Number, default => savesModel.(lastPeriod).entries >> Click >>
/periods/:periodId
        //The total entry amount earned at the moment in the period

        - periods: Number, default => savesModel.periods >> Click >> /periods (array)
        //The periods quantity in the save
```

Conslusion

Users can creates its own account, create a new save instance and use it as a registry for their expenses and entries

The properties like saves:

```
Number, default => [savesModel].count
```

or

```
availableMoney: Number, default => savesModel.(lastPeriod).availableMoney
```

means that at the begining will be a default value of number type and then it will be modificated by front-end interaction, data created by user and saved in the database. Or calculated values from database.