Self-saving for your retirement

Retirement planning remains a critical challenge in emerging markets, where individual savings rates often fall short of long-term financial security needs.

Behavioral economics research demonstrates that automated savings mechanisms—particularly micro-savings through expense rounding—significantly improve accumulation rates by reducing decision friction. This challenge addresses the technical infrastructure required to operationalize such a system at a scale.

Objective

Design and implement production-grade APIs that enable automated retirement savings through expense-based micro-investments. The system must handle complex temporal constraints, validate financial transactions, calculate investment returns across multiple investment vehicles, and provide inflation-adjusted projections.

This challenge evaluates your capability in financial systems design, algorithmic complexity management, data validation rigor, and enterprise-grade software engineering practices.

Problem Description
From a series of n Expenses made during a year (in any order), use the auto-saving strategy with multiples of 100 to determine the amount to invest at the end of the year.
Each expense is represented as expense[i] = {date: t, amount: x};
with $0 \leq n < 10^6$, 0 ≤ i, j ≤ n, t ∈ {w: w is a string of characters indicating a date-time in a fixed format}, $t_i \neq t_j$, $x < 5 \times 10^5$).

To perform auto-saving, each expense must be rounded up to the next multiple of 100 (e.g., for expense [0] = {date: 2021-10-01 20:15:00, amount: 1519}, the next multiple of 100 is 1600, so that the remanent is 81). The remanent (the difference between the next multiple of 100 and the original transaction) is the amount that will be invested for each expense.

The amount to invest can be affected by the following cases/situations:

1. **q moments** are recorded during the year when you can only invest a fixed amount, fixed between the dates start and end within the year (with $0 \leq q < 10^6$, fixed < 5×10^5, {start, end} ∈ {w: w is a string of characters indicating a date-time in any fixed format}, min(t_i) ≤ start ≤ end ≤ max(t_i)). Thus, the rounding remnants to be invested change according to what is recorded (the date ranges can intersect between different moments, and they are inclusive ranges).

2. **p moments** are recorded during the year when, concerned about your future, you add an extra amount to the amounts to be invested form each expense between the dates start and end within the year (with $0 \leq p < 10^6$, extra < 5×10^5 , {start,end} ∈ {w: w

is a string of characters indicating a date- time in any fixed format}, min(t_i) ≤ start ≤ end ≤ max(t_i)). Thus, the rounding remnants to be invested change according to what is recorded (the date ranges can intersect between different moments, and they are inclusive ranges).

Additionally, at the end of the year, it is evaluated to only invest the rounding remnants within **k ranges** of the dates start and end within the year (with $0 \leq k < 10^6$, {start, end} ∈ {w: w is a string of characters indicating a date-time in any fixed format}, min(t_i) ≤ start ≤ end ≤ max(t_i), the date ranges can intersect between different moments, and they are inclusive ranges).

Period Processing Rules

Processing Order

Transaction processing follows this sequence:
Step 1: Calculate ceiling and remanent
Step 2: Apply q period rules (if applicable)
Step 3: Apply p period rules (if applicable)
Step 4: Group by k periods
Step 5: Calculate returns

q Period Rules (Fixed Amount Override)

- When a transaction date falls within a q period (including start and end dates), replace the calculated remanent with the q period's fixed amount.
- If multiple q periods match: use the one that starts latest (start date is closest to transaction date). If they start on the same date, use the first one in the list.

p Period Rules (Extra Amount Addition)

- When a transaction date falls within a p period (including start and end dates), add the p period's extra amount to the remanent.
- If multiple p periods match: add all their extra amounts together. p periods always add to the remanent; they never replace it.
- p is an addition to q rules which means if a transaction falls in q and then p, we apply both.

k Period Rules (Evaluation Grouping)

- For each k period: sum up the remanent of all transactions whose dates fall between the k period's start and end dates (including both).
- A transaction can belong to multiple k periods. Each k period calculates its sum independently.
- Date ranges include both start and end dates.
- Any k range is within a calendar year and not spanning multiple years.

Timestamp Format Specification

- All temporal data uses a timestamp format (date with time component).
- Format: "YYYY-MM-DD HH:mm:ss".
- Periods use start and end timestamps (inclusive).

Constraints:

n	$0 \le n < 10^6$
i, j	$0 \le i, j \le n$
t, start, end	{t, start, end} ⊂ {w: w is a string of characters indicating a date-time in a fixed format}
t_i, t_j	$t_i \ne t_j$
x	$x < 5 \times 10^5$
q	$0 \le q < 10^6$
p	$0 \le p < 10^6$
k	$0 \le k < 10^6$
fixed	$fixed < 5 \times 10^5$
extra	$extra < 5 \times 10^5$
start, end	$\min(t_i) \le start \le end \le \max(t_i)$

Investment Options:

Once the amount to invest is determined, there are two options for making the investment:

- NPS (National Pension Scheme) at an interest rate of 7.11% compounded annually. Assuming a monthly income equal to the salary in Indian Rupees, for the NPS (National Pension Scheme), you can only invest up to 10% of the annual income or max of 2L annually to take advantage of the tax incentive. In this way, a tax rebate is obtained on the invested amount additionally, prioritizing liquidity (this tax rebate is obtained at the end of each period and does not generate interest).

Tax Benefit Calculation (NPS Only)

Eligible NPS Deduction:
NPS_Deduction = min(invested, 10% of annual_income, ₹2,00,000)

Tax Benefit Formula:
Tax_Benefit = Tax(income) - Tax(income - NPS_Deduction)

Assumptions & Simplifications:

1. Salary (wage) is considered as pre-tax
2. Using simplified tax slabs mentioned below and ignoring regimes for simplicity
3. Standard deduction of ₹50,000 and other deductions (80C, 80D, etc.) are not considered

4. NPS deduction limits:
    a. Maximum ₹2,00,000
    b. Limited to 10% of annual income
    c. Based on invested amount
5. Tax benefit is returned separately from profits (not added)

Tax Slabs (Simplified):

- ₹0 to ₹7,00,000: 0%
- ₹7,00,001 to ₹10,00,000: 10% on amount above ₹7L
- ₹10,00,001 to ₹12,00,000: 15% on amount above ₹10L
- ₹12,00,001 to ₹15,00,000: 20% on amount above ₹12L
- Above ₹15,00,000: 30% on amount above ₹15L

- Index Fund (e.g., NIFTY 50) at an interest rate of 14.49% compounded annually. For this fund, there are no restrictions or tax rebates.

- To calculate the final value of the investment after the remaining years until the age of 60, the compound interest formula is applied.

- Once the final return is calculated, it must be adjusted for an inflation to obtain the real return.

Investment Instrument	Annual Rate	Constraints
NPS (National Pension Scheme)	7.11%	tax rebate of up to ₹2,00,000
Index Fund (e.g., NIFTY 50)	14.49%	None

Compound interest formula:

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

Where:

A is the final amount of the investment.
P is the initial amount (remanent invested).
r is the annual interest rate (7.11% for the retirement plan, 14.49% for the NIFTY 50 fund).
n is the number of times the compound interest is applied per year (in this case, annually).

t is the number of years (the difference between 60 years and your age, assuming it is less than 60, otherwise 5).

Inflation adjustment formula:

$$A_{real} = \frac{A}{(1 + inflation)^t}$$

Where:

Areal is the inflation-adjusted amount.
Inflation is the annual inflation rate

Example:
Inputs

Current age: 29 years
Monthly salary: 50,000 Indian Rupees
Yearly salary: 50,000 x 12 = 6,00,000

Expenses:
[
{date: 2023-10-12 20:15, amount: 250},
{date: 2023-02-28 15:49, amount: 375},
{date: 2023-07-01 21:59, amount: 620},
{date: 2023-12-17 08:09, amount: 480}
]

q: [

{fixed: 0, start: 2023-07-01 00:00, end: 2023-07-31 23:59}

]

p: [

{extra: 25, start: 2023-10-01 08:00, end: 2023-12-31 19:59}

]

k: [

{start: 2023-03-01 00:00, end: 2023-11-30 23:59},
{start: 2023-01-01 00:00, end: 2023-12-31 23:59}
]

Explanation

Determine the amount saved based on the Expenses
For this example, 4 Expenses are considered (n = 4):

The expense of 250 is rounded to 300, and the remanent of 50 is invested.
The expense of 375 is rounded to 400, and the remanent of 25 is invested.
The expense of 620 is rounded to 700, and the remanent of 80 is invested.
The expense of 480 is rounded to 500, and the remanent of 20 is invested.
The amount to invest if no extraordinary situation is recorded is **175**.

Update the amount saved based on fixed amounts
For this example, one update is considered (q = 1):
During the month of July, you can only invest a fixed amount of 0 Indian Rupees per expense.

Date range:
start = 2023-07-01 00:00,
end = 2023-07-31 23:59
fixed = 0

Application:
For any expense within this range, the remnants to invest are 0.
Applied example:
The expense of 250 is rounded to 300, and the remanent of 50 is invested.
The expense of 375 is rounded to 400, and the remanent of 25 is invested.
The expense of 620 is not considered for investment.
The expense of 480 is rounded to 500, and the remanent of 20 is invested.
The amount to invest is **95**.

Update the amount saved based on increases
For this example, one increase is considered (p = 1):
Between October and December, you decide to add an extra 25 Indian Rupees to each remanent as a proactive measure.

Date range:
start = 2023-10-01 08:00,
end = 2023-12-31 19:59
extra = 25

Application:
An additional 25 Indian Rupees is added to each calculated remanent.

Applied example:
The expense of 250 is rounded to 300, and the remanent of 50 is increased by 25 and 75 is invested.
The expense of 375 is rounded to 400, and the remanent of 25 is invested.
The expense of 620 is not considered for investment.
The expense of 480 is rounded to 500, and the remanent of 20 is increased by 25 and 45 is invested.
The amount to invest is **145**.

Evaluate the amounts saved in periods
For this example, 2 evaluations are considered (k = 2):
At the end of the year, it is evaluated to invest the remnants for the entire period and from March to November.

Date range:
start = 2023-03-01 00:00,
end = 2023-11-30 23:59

Application:
In each range to be evaluated, the corresponding remnants are taken.

For the total period, the amount to invest is 145.

For March to November (2023-03-01 to 2023-11-30):

- 2023-10-12 (250): INCLUDED, remanent=75 (after p period)
- 2023-02-28 (375): EXCLUDED (before March 1)
- 2023-07-01 (620): INCLUDED but remanent=0 (q period excluded it and it does not fall in any p range to add extra)
- 2023-12-17 (480): EXCLUDED (after November 30)

Total for this k period: 0 + 75 = 75

Date range:
start = 2023-01-01 00:00,
end = 2023-12-31 23:59

Application:
For the full year (2023-01-01 to 2023-12-31):

- 2023-10-12 (250): remanent=75 (p period added 25)
- 2023-02-28 (375): remanent=25 (no q/p periods apply)

- 2023-07-01 (620): remanent=0 (q period excluded)
- 2023-12-17 (480): remanent=45 (p period added 25)

Total for this k period: 25 + 0 + 75 + 45 = 145

Output
{start: 2023-03-01 00:00, end: 2023-11-30 23:59, amount: 75}
{start: 2023-01-01 00:00, end: 2023-12-31 23:59, amount: 145}

Invest the saved amount
Now let's calculate returns for the full year range (2nd k range Jan-Dec) with amount=145:
If the investment period is 31 years (considering the retirement age and capitalization, 60 - 29), the final value of each invested remanent will be calculated using the compound interest formula.

NPS (National Pension Scheme) option:

A = 145 × (1 + 0.0711) ^31
A = 145 × (1.0711) ^31
A = 145 × 8.41
A = 1219.45

NPS_Deduction = min (145, 10% of 50,000, 2,00,000)
= 145

Tax_Benefit = Tax (6,00,000) - Tax (6,00,000 – 145)
= 0 (because the person falls in 0 tax slab in the example)

NIFTY 50 Fund option:

A = 145 × (1 + 0.1449) ^31
A = 145 × (1.1449) ^31
A = 145 × 66.34
A = 9619.7

Finally, the return will be adjusted for an inflation of 5.5% to obtain the final value.
NPS Real Value = 1219.45 / (1.055)^31
= 1219.45 / 5.258
= 231.9

NIFTY 50 Real Value = 9619.7 / (1.055)^31
= 9619.7 / 5.258
= 1829.5

Output
{start: 2023-01-01 00:00, end: 2023-12-31 23:59, amount: 145.0, profit: 86.88, taxBenefit: }

OR
{start: 2023-01-01 00:00, end: 2023-12-31 23:59, return: 1829.5}

API
Build an API that allows transforming, validating, and calculating financial returns from a list of Expenses. This API will be part of an auto-saving system that optimizes investment decisions under certain constraints. The required endpoints, their inputs and outputs, as well as the associated business rules are described below.

1. Transaction Builder

Endpoint: /blackrock/challenge/v1/transactions:parse
Description:
Receives a list of Expenses and returns a list of transactions enriched with the fields of ceiling and remanent. The total invested, remanent, and expense can also be calculated.

Input

```
[
  {"date": "2023-10-12 20:15:30", "amount": 250},
  {"date": "2023-02-28 15:49:20", "amount": 375},
  {"date": "2023-07-01 21:59:00", "amount": 620},
  {"date": "2023-12-17 08:09:45", "amount": 480}
]
```

Where:

timestamp: type datetime
amount: type number (double)

Output

```
[
  { "date": "2023-10-12 20:15:30", "amount": 250, "ceiling": 300, "remanent": 50 },
  { "date": "2023-02-28 15:49:20", "amount": 375, "ceiling": 400, "remanent": 25 },
  { "date": "2023-07-01 21:59:00", "amount": 620, "ceiling": 700, "remanent": 80 },
  { "date": "2023-12-17 08:09:45", "amount": 480, "ceiling": 500, "remanent": 20 }
]
```

Where:

date: type datetime (format: "YYYY-MM-DD HH:mm:ss")
amount: type number (double)
ceiling: type number (double)
remanent: type number (double)

2. Transaction Validator

Endpoint: /blackrock/challenge/v1/transactions:validator

Description:

Validates a list of transactions based on the wage and the maximum amount to invest. Returns valid, invalid, and duplicate transactions.

Input:

```
{
  "wage": 50000,
  "transactions": [
    {"date": "2023-01-15 10:30:00", "amount": 2000, "ceiling": 300, "remanent": 50},
    {"date": "2023-03-20 14:45:00", "amount": 3500, "ceiling": 400, "remanent": 70},
    {"date": "2023-06-10 09:15:00", "amount": 1500, "ceiling": 200, "remanent": 30},
    {"date": "2023-07-10 09:15:00", "amount": -250, "ceiling": 200, "remanent": 30}
  ]
}
```

Where:

wage: type number (double)
transactions: list of transaction objects with the fields:
date: type date
amount: type number (double)
ceiling: type number (double)
remanent: type number (double)

Output:

```
{
  "valid": [
    {"date": "2023-01-15 10:30:00", "amount": 2000.0, "ceiling": 300.0, "remanent": 50.0},
    {"date": "2023-03-20 14:45:00", "amount": 3500.0, "ceiling": 400.0, "remanent": 70.0},
    {"date": "2023-06-10 09:15:00", "amount": 1500.0, "ceiling": 200.0, "remanent": 30.0}
  ],
  "invalid": [
    {"date": "2023-07-10 09:15:00", "amount": -250.0, "ceiling": 200.0, "remanent": 30.0, "message": "Negative amounts are not allowed"}
  ]
}
```

Where:
valid: list of valid transactions (transaction)
invalid: list of invalid transactions (invalid_transaction), which extends transaction with an additional field:

message: type string (explanation of the error)

3. Temporal Constraints Validator

Endpoint: /blackrock/challenge/v1/transactions:filter
Description:
Validates transactions according to the periods defined as “q”, “p”, and “k”. Each period has specific rules that affect the validity of transactions. Returns
valid and invalid transactions.

Input:

```
{
  "q": [{"fixed": 0, "start": "2023-07-01 00:00:00", "end": "2023-07-31 23:59:59"}],
  "p": [{"extra": 30, "start": "2023-10-01 00:00:00", "end": "2023-12-31 23:59:59"}],
  "k": [{"start": "2023-01-01 00:00:00", "end": "2023-12-31 23:59:59"}],
  "wage": 50000,
  "transactions": [
    {"date": "2023-02-28 15:49:20", "amount": 375},
    {"date": "2023-07-15 10:30:00", "amount": 620},
    {"date": "2023-10-12 20:15:30", "amount": 250},
    {"date": "2023-10-12 20:15:30", "amount": 250},
    {"date": "2023-12-17 08:09:45", "amount": -480}
  ]
}
```

Where:
q: list of objects with:

fixed: number (double)
start, end: type datetime

p: list of objects with:

extra: number (double)
start, end: type datetime

k: list of objects with:

start, end: type datetime

transactions: list of transactions with:

timestamp: type datetime amount, ceiling, remanent: type number (double)

Output:

```
{
  "valid": [
    {"date": "2023-02-28 15:49:20", "amount": 375.0, "ceiling": 400.0, "remanent": 25.0, "inKPeriod": true},
    {"date": "2023-10-12 20:15:30", "amount": 250.0, "ceiling": 300.0, "remanent": 80.0, "inKPeriod": true}
  ],
  "invalid": [
    {"date": "2023-10-12 20:15:30", "amount": 250.0, "message": "Duplicate transaction"},
    {"date": "2023-12-17 08:09:45", "amount": -480.0, "message": "Negative amounts are not allowed"}
  ]
}
```


Where:

valid: list of valid transactions (transaction)

invalid: list of invalid_transaction, which extends transaction with an additional field:

message: type string (explanation of the error).

4. Returns Calculation:

Endpoints:

/blackrock/challenge/v1/returns:nps

/blackrock/challenge/v1/returns:index

Description:

Calculates the return on investments from transactions, salary, age, inflation, periods, and return rates. Returns financial metrics and investment breakdown by k periods.

Input:


```
{
  "age": 29,
  "wage": 50000,
  "inflation": 5.5,
  "q": [{"fixed": 0, "start": "2023-07-01 00:00:00", "end": "2023-07-31 23:59:59"}],
  "p": [{"extra": 25, "start": "2023-10-01 08:00:00", "end": "2023-12-31 19:59:59"}],
  "k": [{"start": "2023-01-01 00:00:00", "end": "2023-12-31 23:59:59"},
        {"start": "2023-03-01 00:00:00", "end": "2023-11-31 23:59:59"}],
  "transactions": [
    {"date": "2023-02-28 15:49:20", "amount": 375},
    {"date": "2023-07-01 21:59:00", "amount": 620},
    {"date": "2023-10-12 20:15:30", "amount": 250},
    {"date": "2023-12-17 08:09:45", "amount": 480},
    {"date": "2023-12-17 08:09:45", "amount": -10}
  ]
}
```


Where:

age: integer number

wage, inflation: type number (double)

q, p, k: lists of periods as defined previously

transactions: list of transactions

Output:

```
{
    "totalTransactionAmount": 1725.0,
    "totalCeiling": 1900.0,
    "savingsByDates": [
        {
            "start": "2023-01-01 00:00:00",
            "end": "2023-12-31 23:59:59",
            "amount": 145.0,
            "profit": 86.88,
            "taxBenefit": 0.0
        },
        {
            "start": "2023-03-01 00:00:00",
            "end": "2023-11-31 23:59:59",
            "amount": 75.0,
            "profit": 44.94,
            "taxBenefit": 0.0
        }
    ]
}
```


Where:

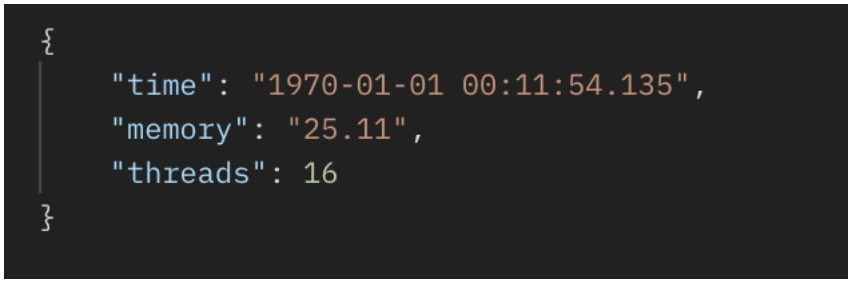
transactionsTotalAmount: type number (double) - sum of valid transaction amount
transactionsTotalCeiling: type number (double) - sum of valid transaction ceiling
savingsByDates: list of objects (of the same size as the k value list) with:

- start, end: type datetime
- amount: type number (double)
- profits: type number (double)
- taxBenefit: type number (double) - 0 in case of index)

5. Performance Report
Endpoint: /blackrock/challenge/v1/performance
Description: Reports system execution metrics such as response time, memory usage, and number of threads used.

Input: NOT APPLICABLE

Output:



Where:

time: type string (format: "HH:mm:ss.SSS" or duration in milliseconds)
memory: type string (memory usage in megabytes, format: "XXX.XX MB")
threads: integer number (number of threads used)

Deployment

As part of the development and release of the solution, the server that includes the business logic must be containerized with Docker and meet the following requirements:

1. Container Configuration:
   a. Application must run on port 5477 inside the container
   b. Docker port mapping: -p 5477:5477 (host:container)
   c. Expose port 5477 in Dockerfile

   Example Dockerfile: EXPOSE 5477
   Example Docker Run: docker run -d -p 5477:5477 blk-hacking-ind-{name-lastname}

2. The operating system must be based on a Linux distribution and include the selection criteria as a comment in the OS definition.

3. The first line of the Dockerfile must include the command to build the image. The image name must follow the convention with the participant's details:
   blk-hacking-ind-{name-lastname}

4. If there are dependencies on other services, a YAML file with the definitions and configurations must be included to be executed with Docker Compose.
   File naming convention: compose.yaml

Testing

Any included software testing will be considered by the judges as a bonus to the evaluation score.
If tests are included, they must be placed in the project under a folder named “test”, and the files must specify the following data as a comment:

1. Test type
2. Validation to be executed
3. Command with the necessary arguments for execution.

Finally think beyond boundaries—be creative, challenge assumptions, and deliver more than what's asked for. Innovation starts where expectations end!.

Submission

To complete the submission of your code challenge, once you finish your solution, share your response with the link to a public repository on a Git-based version control system (e.g., GitHub, GitLab, Bitbucket, Azure Repos). This link must allow access to the repository without the need for additional permission. Otherwise, in case of duplicate submissions or empty repositories, the submission will not be considered for evaluation (changes to the repository after the code challenge deadline will also not be considered).
Your repository must contain (in the default branch):

- All the source code of your solution to the challenge, including the Dockerfile, test automation, and a README.md file with clear and detailed instructions on how to configure, run, and test the solution. Be sure to explain any additional dependencies or requirements.

Good luck with your code challenge!