## Overview

### Problem Description

Each season, clients purchase products on credit, and over the course of a season, they repay their credit, and so clients have credit associated with them on a season-by-season basis.

In our data model, we associate a client’s payments with a season for which they have outstanding credit. These payments are made either through cash to a Field Officer or through a mobile money platform (ex. M-Pesa). A member of our HQ staff then takes a list of payments that clients have made, and uploads them into our application, which saves each payment, and each payment must be associated with the correct client and season in our database.

When a client makes a payment, we need to know which season the payment will be applied to, as sometimes clients can have outstanding credit (debt) in more than one season. Traditionally, we required each payment in a payment upload to be associated with a season, so that the application doesn’t have to figure out to which season(s) the client’s payment needs to be applied to. But to make it easier on clients, Field Officers, and HQ staff, we’re moving away from this model, and dropping the requirement that seasons must be associated with each payment contained in a list of payment uploads. Since we still require that each saved repayment must be associated with a season in our database, this means we need ‘seasonless repayment’ logic to determine which season(s) the client’s payment should be applied to.

### Technical Details

1. The Cascade - If a client makes a payment that exceeds the outstanding credit (debt) for his/her oldest season, then additional repayment records must be created until all of the client’s remaining payment has been applied to any/all seasons with outstanding credit, from oldest to newest. See the **Simple Example** section below:

2. The Overpaid - If a client does NOT have outstanding credit in a later season, or does NOT have outstanding credit in ANY season, then NO additional repayment records are created, and the full amount is applied to the client’s most recent (max) season.

3. The Override - If a client’s payment specifies a season (i.e. contains a non-zero, non-null SeasonID value), then NO additional repayments are created, and the full amount is applied to the specified season.

### Project Deliverables

There are 4 deliverables for this project:

1. **Service class** - The service class should input a list of ([RepaymentUploads](#_3rdcrjn)) client repayments, and the logic should determine which season(s) the repayment upload should be applied to. The service class should also *output* a list of ([Repayments](#_17dp8vu)) for the correct client, season, and amount for each repayment, and this list should be inclusive of any adjustments repayments that must be made. Additionally, every adjustment repayment should be associated back to the original repayment.
2. **Basic interface** - The interface should display the state of the application’s data *before* the repayment upload, and the interface should display the state of the application’s data *after* the repayment upload; this display should be inclusive of any adjustment repayments that must be made. Additional UI enhancements are welcome, but not required. A stretch goal would be to display the proposed changes before saving, with the understanding that during the display of the proposed changes, the underlying application data (or other application data) may have been edited or changed in the meantime.
3. **Test data** - The list of RepaymentUploads & CustomerSummaries should be considered a *starting point* for testing purposes, and should be *updated* to handle any unhandled corner cases or edge cases as needed.
4. **Post-mortem** - The post-mortem should outline the following:
   1. Current project status
   2. Estimate on the outstanding work
   3. Successes/what went well
   4. Bumps/what you wished went better
      1. How you would improve your approach in future projects
   5. Improvements/enhancements to this project for future consideration
5. **Final project** - Before the final interview meeting, this should be a link or a zipped file of #1-4. Feel free to link to or attach your final project to the “final interview” event, or by replying to the welcome email containing the original project files.

### Additional Considerations

You will be provided with sample data. You may use whatever local development environment, framework, or language you feel comfortable coding in. The team will be most comfortable in Angular, JavaScript, C#, and SQL, but are conversant enough in other languages to meaningfully interact with your code and evaluate your work.

You will spend about 30mins each day pair programming with at least one member of the Dev team. You will be driving, so use this time to ask questions and get feedback on your code.

### Final Note

This is not primarily a coding test, but the quality of your code will have some relationship to the success of your time management and pair programming. You do not have to present a completed project, but you will be expected to present a post-mortem on what went well and what you would have changed, in hindsight, as well as an estimate of how much work remains to be done, and possible improvements or ‘stretch goals’ for future enhancements to the project.

This final interview is designed to assess:

* Your communication and critical-thinking style
* Your ability to collaborate, when appropriate
* Your problem-solving skills and strategies
* Your ability to make and execute a project plan

### Data Schema

#### Seasons

SeasonID (int)

SeasonName (string)

StartDate (date)

EndDate (date)

#### Customers

CustomerID (int)

CustomerName (string)

#### CustomerSummaries

CustomerID (int)

SeasonID (int)

TotalRepaid (dec)

TotalCredit (dec)

#### RepaymentUploads (INPUT)

CustomerID (int)

SeasonID [optional] (int)

Date (date)

Amount (dec)

#### Repayments (OUTPUT)

RepaymentID (int)

CustomerID (int)

SeasonID (int)

Date (date)

Amount (dec)

ParentID (int)

### 

### Simple Example

If a client starts out with 2 seasons of outstanding credit (debt):

* CustomerSummary (Client owes 20)
  + Season = 2011
  + TotalRepaid = 80
  + TotalCredit = 100
* CustomerSummary (Client owes 90)
  + Season = 2012
  + TotalRepaid = 30
  + TotalCredit = 120

When the client makes a payment of 60, we would expect to save 3 repayment records:

* Repaymend record #1 - Season = **2011**, Amount = **+60** - original repayment record
* Repayment record #2 - Season = **2011**, Amount = **-40** - adjustment repayment record
* Repayment record #3 - Season = **2012**, Amount = **+40** - adjustment repayment record

And we would also expect to see updated customer summaries:

* CustomerSummary (Client owes 0)
  + Season = 2011
  + TotalRepaid = 100
  + TotalCredit = 100
* CustomerSummary (Client owes 50)
  + Season = 2012
  + TotalRepaid = 70
  + TotalCredit = 120