Title: Meter Reading API

(any assumptions I have made I have marked with PO Confirmed, to show that this would be a decision from the Product Owner)

Story Points: 5 (medium complexity)

Problem statement:

As an Energy Company Account Manager,
I want to be able to load a CSV file of Customer Meter Readings
So that We can monitor their energy consumption and charge them accordingly

Acceptance Criteria:

- The url must be /meter-reading-uploads
- The endpoint must be able to process a CSV
- Valid entries must be stored in a database
- The response must contain the total of successful and failed readings
- Validation:
    - Duplicate entries are not allowed
    - The reading must have a valid AccountId
    - Reading must be numerical to 5 places NNNNN
    - Negative values are invalid

Technical details:

Csv Processing

- Additional columns should be ignored and not cause errors *(found in test data - PO confirmed)*
- The original meter reading value should be preserved and then parsed to conform to the validation rules
- CsvHelper has been identified as a library to process the CSV files for us

Validation

- The test data contains a considerable number of values that are too short and do not conform to the required format, these will be considered invalid *(PO confirmed)*
- The meter reading has a potential to loop back to zero, this was raised by the team and considered out of scope for this initial development *(PO confirmed)*

Database design

- There is potential for a foreign key constraint on accountId, the team made the decision not to tightly couple the tables at this point

Test Cases:

1. Given a POST request
   And the csv is not present
   Then return bad request

2. Given a POST request
   And the csv is present
   When the data is proceeded
   And all data is correct
   Then the success count equals the uploaded count

3. Given a POST request
   And the csv is present
   When the data is proceeded
   And all data is incorrect
   Then the failure count equals the uploaded count

4. Given a POST request
   And the csv is present
   When the data is proceeded
   And some data is correct
   Then the failure and success values are correct
   And total the number uploaded

Tasks:

Bootstrapping

- [x] ~~B1: Create skeleton Project~~
    - [x] ~~Build project~~
    - [x] ~~Add docker orchestration~~
    - [x] ~~Check project loads~~
    - [x] ~~Add testing project~~
- [x] ~~Research and select database engine (assuming Sql Server)~~
    - [x] ~~Install ef core~~
    - [x] ~~Deploy database to docker-compose~~
    - [ ] Connect successfully to the Db

Account Data

- [ ] A1 Create Table
- [x] ~~A2 Create Table seeding~~
- [x] ~~A3 Create Accounts Repository~~

Meter reading Api

- [x] ~~M1 Create minimal Api~~
- [x] ~~M2 Create Meter Reading Service~~
- [ ] M3 Create Validator
    - [x] ~~Create basic validator skeleton~~
    - [x] ~~Create accountId validation~~
    - [x] ~~Create meter reading validation~~
        - [x] ~~Validate too short~~
        - [x] ~~Validate too long~~
        - [x] ~~Validate non only numeric to 5 digits~~
        - [x] ~~Validate positive value~~
    - [ ] Make date of meter reading validation
    - [ ] Create duplicate row validation
- [ ] M4 Create MeterReading Repository

Coding challenge retrospective:

Good:

- The challenge was good, taxing, complex and required thinking of edge cases up front to fix issues.
- Up front planning helped focus the development and identify the paths that gave the biggest value
- Identified additional edge cases during development (code gracefully handles a pdf upload)

Frustrating:

- SqlServer in docker did not set up as well and as quickly as I had hoped for the timeframe, a networking issue prevented the seeding.
- The lack of the Db prevented me developing sections of the code I wanted to

Completion notes:

With more time to complete this project I would have moved on to:

- Implementing a Unit of Work pattern over the top of the database layer
- Converting and saving the meter readings into the database
- In the validator, looking up the last meter reading to validate against old entries
- In the validator, beginning to plan and implement a check for duplicate rows (in code rather than relying on a database unique key exception)
- Database rollbacks

If I had all the time in the world:

- Implement a queue mechanism so the user is not waiting on the response for large imports
- Adding logic for duplicate file uploads
- Reporting all errors in the response (maybe with a /job/{id}/details endpoint)