I'll provide you with some guidance on how to approach each step and the optional challenges.

Step 1 – Database storage

You'll need a database to store the meter and meter reading information. You can use a relational database like MySQL for this purpose. Create two tables: one for storing meter information and another for storing meter readings. Here's a basic structure for each table:

**Meters Table:**

id (Primary Key)

identifier (MPXN)

installation\_date

Meter type (electricity or gas) - ENUM or VARCHAR

**Meter Readings Table:**

id (Primary Key)

meters\_id (Foreign Key referencing Meters table)

reading\_value

reading\_date

**Estimated Readings**

id (Primary Key)

meters\_id (Foreign Key referencing Meters table)

estimated\_reading

Step 2 – Meters

For adding a new meter and displaying a list of meters, you can create a simple HTML form that sends data to a PHP script for processing. Similarly, for displaying meter information and readings, you can create a page that lists the meter information and a table displaying the readings associated with that meter.

Step 3 – Meter Readings

To add a new meter reading, you can create a form in the meter information display page. This form would send data to a PHP script that adds the reading to the database. Use validation to ensure the reading is an integer.

# Optional A – Estimated Readings

A.1 – Storage and Calculations

Modify your Meters table to include a new field for storing the estimated annual consumption.

To calculate the estimated reading, you can use the formula:

**Estimated Reading = Previous Reading + (Days since Previous Reading \* Estimated Daily Consumption)**

**Where Estimated Daily Consumption = (Estimated Annual Consumption / 365)**

A.2 – User Interface

Add a date input and submit button to the meter detail view. When the user selects a date and submits the form, use the logic from A.1 to calculate the estimated reading and save it to the database as a new meter reading associated with the selected meter.

A.3 – Validation

When a customer submits a reading, you can implement the validation by comparing the submitted reading with the expected reading calculated using the logic from A.1. If the submitted reading falls outside the acceptable range (e.g., ±25% of the expected reading), show an error message and prevent the submission.

**<?php**

**// Check if the provided reading falls within the expected range**

**$expected\_reading = calculateExpectedReading($previous\_reading, $previous\_reading\_date, $estimated\_consumption, $reading\_date);**

**$acceptable\_range = $expected\_reading \* 0.25; // 25% range**

**if (($reading\_value - $expected\_reading) > $acceptable\_range) {**

**// Reading is outside acceptable range, show an error message**

**} else {**

**// Reading is within acceptable range, proceed with adding to the database**

**// ...**

**}**

**?>**

# Optional B – Bulk Uploading

When we need to add lots of readings to the system, a bulk upload process can simplify this.

A basic example of how you could achieve this in a Laravel application.

1. **Setting Up Your Laravel Project:**

Assuming you already have a Laravel project set up, let's start by creating a route, a controller, and a view.

1. **Create a Route:**

In your routes/web.php file, add a route to handle the form display and CSV processing:

**Route::get('/meter-readings', 'MeterReadingController@index');**

**Route::post('/meter-readings/upload', 'MeterReadingController@upload');**

1. **Create a Controller:**

Generate a new controller named MeterReadingController:

**php artisan make:controller MeterReadingController**

Then, open the app/Http/Controllers/MeterReadingController.php file and define the following methods:

**use Illuminate\Http\Request;**

**use Illuminate\Support\Facades\Validator;**

**class MeterReadingController extends Controller**

**{**

**public function index()**

**{**

**return view('meter-readings.index');**

**}**

**public function upload(Request $request)**

**{**

**$validator = Validator::make($request->all(), [**

**'readings' => 'required|string',**

**]);**

**if ($validator->fails()) {**

**return redirect('/meter-readings')**

**->withErrors($validator)**

**->withInput();**

**}**

**$readingsData = $request->input('readings');**

**$lines = explode("\n", $readingsData);**

**foreach ($lines as $line) {**

**$data = str\_getcsv($line);**

**if (count($data) !== 3) {**

**continue; // Skip invalid lines**

**}**

**$reading = $data[0];**

**$readingDate = $data[1];**

**$meterIdentifier = $data[2];**

**// Here you can perform additional validation on $reading, $readingDate, and $meterIdentifier**

**// Assuming you have a MeterReading model to persist data to the database**

**// MeterReading::create([...]);**

**}**

**return redirect('/meter-readings')**

**->with('success', 'Readings uploaded successfully.');**

**}**

**}**

1. **Create a View:**

Create a view file named index.blade.php in resources/views/meter-readings directory:

**Additional Steps:**

Make sure you have a database configured in your .env file and run migrations if necessary.

Create a MeterReading model and configure its corresponding database table.

Customize the validation and data insertion logic in the controller to match your specific requirements.

Implement better error handling, security measures, and authentication in your application.

Remember that this is a simplified example and might not cover all edge cases or security considerations. Always ensure proper validation, sanitation, and security measures in your real-world application.

We'll create a job to handle the processing of the uploaded data and persist the valid readings to the database.

Create a Job:

**Generate a new job named ProcessMeterReadings:**

**php artisan make:job ProcessMeterReadings**

This will create a job file at app/Jobs/ProcessMeterReadings.php. Modify the handle method to process the uploaded data and persist valid readings:

**use Illuminate\Bus\Queueable;**

**use Illuminate\Contracts\Queue\ShouldQueue;**

**use Illuminate\Foundation\Bus\Dispatchable;**

**use Illuminate\Queue\InteractsWithQueue;**

**use Illuminate\Queue\SerializesModels;**

**class ProcessMeterReadings implements ShouldQueue**

**{**

**use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;**

**protected $readingsData;**

**public function \_\_construct($readingsData)**

**{**

**$this->readingsData = $readingsData;**

**}**

**public function handle()**

**{**

**$lines = explode("\n", $this->readingsData);**

**foreach ($lines as $line) {**

**$data = str\_getcsv($line);**

**if (count($data) !== 3) {**

**continue; // Skip invalid lines**

**}**

**$reading = $data[0];**

**$readingDate = $data[1];**

**$meterIdentifier = $data[2];**

**// Here you can perform additional validation on $reading, $readingDate, and $meterIdentifier**

**// Assuming you have a MeterReading model to persist data to the database**

**// MeterReading::create([...]);**

**}**

**}**

**}**

**Modify the Controller to Dispatch the Job:**

In your MeterReadingController, modify the upload method to dispatch the ProcessMeterReadings job:

**use App\Jobs\ProcessMeterReadings;**

**class MeterReadingController extends Controller**

**{**

**// ...**

**public function upload(Request $request)**

**{**

**$validator = Validator::make($request->all(), [**

**'readings' => 'required|string',**

**]);**

**if ($validator->fails()) {**

**return redirect('/meter-readings')**

**->withErrors($validator)**

**->withInput();**

**}**

**$readingsData = $request->input('readings');**

**ProcessMeterReadings::dispatch($readingsData);**

**return redirect('/meter-readings')**

**->with('success', 'Readings are being processed in the background.');**

**}**

**}**

**Queue Configuration:**

Make sure your queue driver is set to a suitable option (e.g., database or redis) in your .env file.

**Running the Queue Worker:**

In a terminal, start a queue worker to process the queued jobs:

**php artisan queue:work**

The worker will process the jobs in the background, allowing the front-end to remain responsive.