D3.4 FIWARE-enabled applications for customers

Author: Gareth Lewis (UNEXE)
Co-Authors: Danillo Lange (EUT)

May 2022

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant agreement No. 821036.

Disclaimer

This document reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Intellectual Property Rights

Project Consortium

Executive Summary

Deliverable 3.4 describes the role of data within demo case #4: Smart Meters and Customers (United Kingdom) and specifically concerns data produced / stored in the Stellio broker, as captured from Diehl Altair V4 15MM smart meters in the Great Torrington pilot and SWW customer data from various sources.

Section I of the report details the key values in the Diehl Altair V4 15MM smart meter dataset (waterConsumption, minFlow and alarmFlowPersistence) and describes how minFlow and alarmFlowPersistence are used to determine and measure customer-side leakage events.

Section II details the approach taken for dealing with missing waterConsumption readings, typically due to over-the-air meter data capture failing due to parked vehicles on, or near to, the smart meter transponder, where 'missing' readings are estimated and in-filled with actual data on a linear scale.

Section IV details the approaches taken to extract customer 'cluster' data, i.e. grouping customers together by consumption, environmental and property data, using a range of data sources provided by SWW (Table 2).

Section V describes approaches to the visualisation of smart meter data through the Stellio context broker, with both the utility and customer-side applications presented. The over-arching goal of data visualisation was to provide the two different user groups (SWW staff and SWW customers) with different presentations of the same data to both: 1) ensure data consistency between both user groups and 2) provide data presentations that met the needs of both user groups.

Section VI details optimisation approaches that were undertaken in order to minimise the time taken to process Stellio data and present it to users. This was particularly evident in functions that processed the entire Stellio data set, e.g. collecting leak data for the utility app and collecting overall population consumption for the customer app. Optimisations were centred around moving Stellio data onto a traditional database and employing 'traditional' SQL operations and caching frequent daily requests.

Section VII records the conclusions and perspectives of the application development project and highlights the need for a better understanding of the FIWARE/Stellio/Docker ecosystem when starting a project of this nature, the generally positive suitability of Django and Flutter as a full-stack platform for mobile development, and the pivotal role of IT support in a large organisation.

Section VIII records the recommendations for future work in this area.

Related Deliverables

D1.1 Requirements from Demo Cases

D4.5 FIWARE4_ Smart Metering and Citizen Engagement

Document Information

Programme	H2020 – SC0511-2018
Project Acronym	**Fiware4Water**
Project full name	FIWARE for the Next Generation Internet Services for the WATER sector
Deliverable	**D3.4: FIWARE-enabled applications for customers**
Work Package	**WP3: Smart Applications and Devices**
Task	Task 3.4
Lead Beneficiary	University of Exeter
Author(s)	Gareth Lewis (UNEXE)
Contributor(s)	Danillo Lange (EUT)
Quality check	Alex van der Helm (WATNL)
Planned Delivery Date	30/04/2022
Actual Delivery Date	24/05/2022
Dissemination Level	Public (Information available in the Grant Agreement)

Revision history

Version	Date	Author(s)/Contributor(s)	Notes
Draft 1	18/04/22	Gareth Lewis (UNEXE), Danillo Lange (EUT)	First version of the document
Draft 2	22/04/22	Alex van der Helm (WATNL)	Internal review of the document
Draft 3	20/05/22	Gareth Lewis (UNEXE), Danillo Lange (EUT)	Final version of the document
Final	24/05/22	Gareth Lewis (UNEXE), Danillo Lange (EUT)	Addition of an executive summary
V2	11/07/22	Gareth Lewis (UNEXE)	Addition of recommendation section

Table of content

List of figures

List of tables

List of Acronyms/Glossary

Text: Calibri, 11 (alphabetic order). Acronyms in bold. Full name, normal. Definitions (if needed) in italics.

F4W Fiware4Water project

NGI Next Generation Internet

The Next Generation Internet (NGI) initiative, launched by the European Commission in the autumn of 2016, aims to shape the future internet as an interoperable platform ecosystem that embodies the values that Europe holds dear: openness, inclusivity, transparency, privacy, cooperation, and protection of data.

OTA **Over the air, wireless transmission of data**

WPL Work Packages Leaders

Introduction

The goal of the SouthWest Water demo case was to implement a FIWARE enabled pipeline to retrieve consumption data from smart meters and provide this data to customers via a smartphone application to drive positive changes in water use behaviour, reduce consumption, and reduce customer water bills. The data will also be used operationally to detect customer leaks.

Figure 1 presents the conceptual model of the proposed smart metering solution. Customer consumption data is collected from Sigfox smart meters daily and stored in a FIWARE-Stellio context broker. Customers can access their data through a mobile phone app. Value-add data is available to customers through the interrogation of existing customer data, both the customer's data and the pilot case data, as a whole or subsets thereof, and potentially augmented with data from other sources.

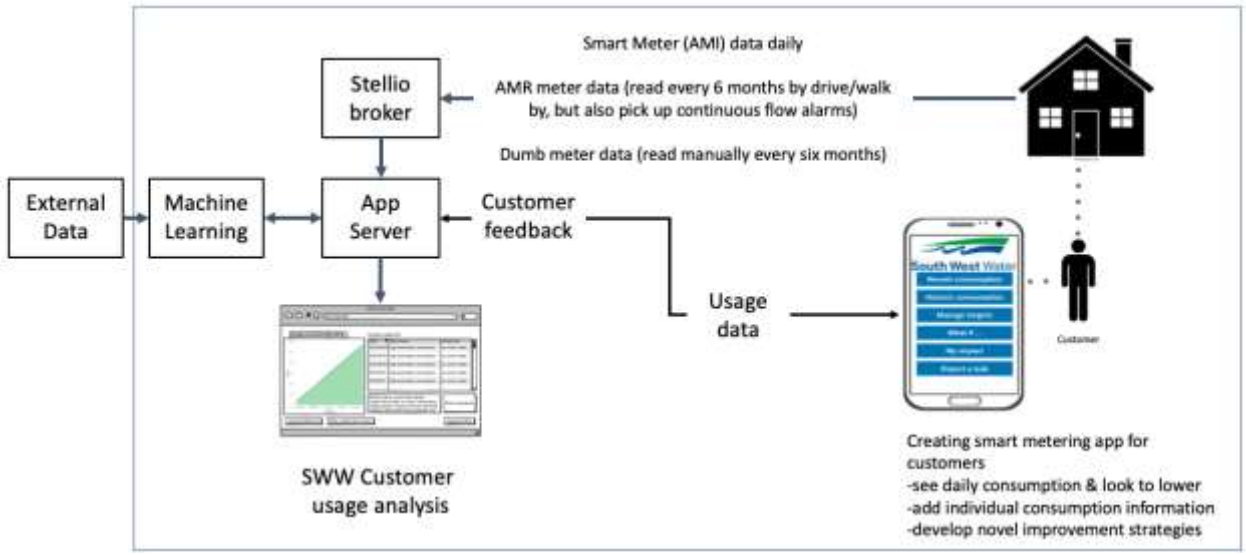

Figure 1: Conceptual architecture

The demo case has been implemented as a collection of Docker containers, with one container (provided by EGM) implementing the Stellio context broker and the other implementing the customer-side and utility-side applications. Both applications are implemented using Django, with the customer-side app providing functionality primarily through a mobile app implemented in Flutter. The Flutter app is also available as a development-side web application, though this is not intended for customer use. The utility-side app is realised as a web app.

I. Data Integration

For this pilot, a water consumption smart model was developed as a digital proxy of the physical smart meters used in the pilot, Diehl Altair V4 15MM. Data was collected from the physical smart metersdaily, with key attributes of the water consumption model being waterConsumption, minFlow and alarmFlowPersistence.

I.1. waterConsumption

This attribute records total water consumption since the meter has been installed and can be considered functionality equivalent to traditional 'elapsed' meters. Periodic consumption can be

determined by subtracting daily consumption figures. However, data is not collected with 100% reliability due to OTA (over the air) data transmission failures, typically from parked vehicles blocking data transmission, and other operational issues, so care must be taken when determining daily consumption figures.

I.2. minFlow

This attribute records the minimum flow level observed by the meter during a 24hr period. Ideally, this should be zero as non-zero values are indicative of continuous flow, 'leak'. This value is used with alarmFlowPersistence to determine the severity of leak.

I.3. alarmFlowPersistence

This attribute translates the minFlow attribute into a qualitative term and is used for determining the severity of a continuous flow / leak situation. The Sigfox meter generates the following terms: 'Nothing to report', 'no persistence', 'In progress impacting persistence', 'In progress persistence' and 'Past Persistence during the period'].

In the application, 'In progress impacting persistence' and 'In progress persistence' are used to define a leak for the daily reporting period.

II. Data processing

Smart meter data is collected on a rolling daily basis (with several meters being processed every hour, rather than processing all meters at a given point in the day) and stored in Stellio using the WaterConsumption entity.

Although the OTA daily data collection was generally reliable (with data being collected far more often than not), there were periods where data was not collected, with two broad use cases of collection failure; 1) data was not collected due to an OTA transmission failure, which presented as isolated days of no data collection for a given meter or 2) data was not collected due to IOT Agent failure, which presented as multi-day data collection failures across all of the meters.

Whilst this was not a huge issue for the system, as consumption is measured as total since meter installation, rather than absolute daily consumption, it did make visualization and data processing over given periods awkward due to the potential for arbitrary gaps in data.

To address this, functionality was developed to automate the production of 'infill' data that would create estimated consumption values between known start and end points, Figure 2. This approach was particularly useful to visualize historic consumption data for the customer app.

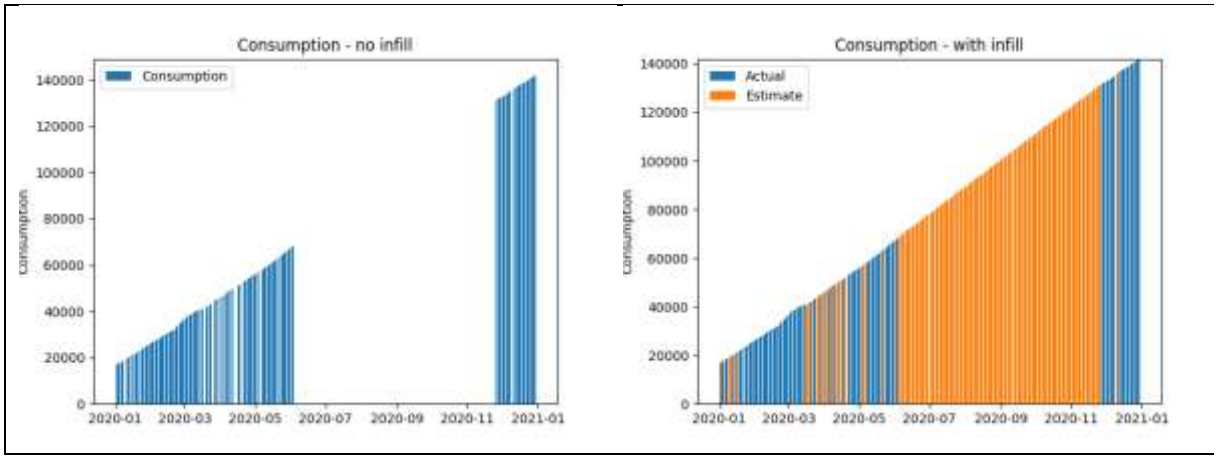

Figure 2: Water consumption infill in operation

Infill was only undertaken for water consumption as its role was primarily to visualize rather than logical/analytical. Figure 2 illustrated the two failure modes for data collection, with the individual orange estimates shows isolated data collection failure occurrences and the large band from 2020-7 to 2020-11 showing where longer term data collection failure had occurred, due to IoTAgent failure.

III. Implementation of Smart Data Models & Context Broker

The pilot maintains two smart data models: water consumption and consumer info. The water consumption model captures all the relevant data from the smart meters, in particular waterConsumption, minFlow and alarmFlowPersistence.

The consumer information model is a self-reporting model that is used for user clustering. It comprises of two parts: customer information, and customer settings. Given the experimental nature of clustering, the information component stores a set of cluster properties and value choices that users may select, whilst the settings component stores the current selection of those properties. This approach allows cluster properties to be changed on demand and provides a data-driven scaffold to allow users to set novel properties in the customer-side app without needing to update the app codebase.

The context broker was provided by EGM and realised as a Stellio broker.

IV. Water use clustering

The presented work aims at providing the water supply organization with an insight into identified behaviour across urban households, by analysing historical values of consumption and other factors such as economic, social, and weather parameters. An Artificial Intelligence (AI) data-driven ensemble approach is proposed, which combines multiple clustering algorithms with time-series data analytics and advanced dimensionality reduction techniques. As a result, a predictive model is obtained which provides a real-time classification, among different behaviour groups, based on any given customer's short-mid and long-term historical and static information.

IV.1. Data

The provided datasets were composed of multiple files, they are available in .csv format and are stored off-line, as can be seen in Table 1:

Table 1 Water consumption dataset description

Title	Description	Method of acquisition
Sampled Properties.csv	A table of sampled properties, sample types and metadata.	Received by email
Sampled Property Occupancy.csv	A table of occupancy records for each sampled property.	
Rainfall Stations 4200m Grid.csv	A table of 674 rainfall stations for which data has been retrieved since January/2019.	
Temetra Consumption Data.sqlite	Consumption data from customers in the Temetra sample.	
AMRConsumptionData.sqlite	Consumption data from customers in the AMR sample.	
PITConsumptionData.sqlite	Consumption data from customers in the PIT sample aggregated to 15 minute time period.	
WaterUse_SampledCustomers_DataSummary.csv	Dataset containing financial, social and demographic properties of households.	
PIT__Data_15min_2018_2021_NoOverlaps.sqlite	Consumption from PIT sample with gaps filled.	

The data set used to demonstrate our developments, shown in Table 2, composes of high-resolution time-series water consumption data collected from 3 different types of smart sensors scattered across the area, with dates in the range from 2016 to 2021 at an interval between 15-minutes and a day.

Table 2 Sensor reading grouped information

Feature	Number of samples	Description	Unit	No. of registers
Sensor type 1 regadings	562	Water consumption registered from the sensor with the highest definition, in a 15-minute interval.	Litres/15 minutes	42,414,797
Sensor type 2 readings	2111	Water consumption registered from sensor 2 in a 30-minute interval.	Litres/day	5,630,035
Sensor type 3 readings	335	Daily water consumption in litres registered from sensor 3.	Litres/30 minutes	820,313
Rain information	-	Rain historic information and location of rainfall stations	mm, string	2,047,668

Moreover, it also includes a rainfall dataset containing the height in millimetres with irregular intervals. Lastly, the data set containing demographic characteristics and financial level segmentation of said households that are presented in Table 3. Moreover, all presented features were linked to the households and were supplied along with the consumption data sets.

Table 3 Demographic and financial features

Feature	Description	Unit	N. of registers
Financial category	Powerful individual level and financial services segmentation, built based on IPSOS's Financial Research Survey.	Categorical	4,918
Net worth category	Individual classification of consumers based on their net worth, combines income, family composition and financial assets.	Categorical	4,918
Demographic category	Demographic and neighbourhood characteristics	Categorical	4,592
Garden Size	The floor size of the property's garden. Units are in m².	m²	5,090
Building Size	The floor size of the property's building. Units are in m².	m²	5,090
Household Size	Estimated number of persons that resides on the property.	Integer	5,111
Children at home	Estimated number of children residing on the property.	Integer	5,111

Data preparation

While working with time-series data, some pre-processing steps are necessary because either error during data collection may occur or some specific behaviour registered in the data can negatively affect the development of ML models.

The tasks performed in the earlier steps of the data analysis were the analysis of outliers and consecutive zeroes in the historic water demand data. It was decided to compare every value of the historic data set against a defined threshold at a local (property) scope to deal with outliers. The values that meet this criterion, as seen in equation (1), are then replaced by its local median value; this process was also applied to household characteristics data fields as well.

$$Value > \left(\overline{X} + 3 \times \sigma\right) \quad (1)$$

where $\overline{X}$ = mean value and σ = standard deviation, both calculated locally

From data analysis, it was found that 30 or fewer days with consecutive zeroes in readings can be considered vacations, so it was decided to delete from the time series any period with more zeroes than that.

Moreover, missing and negative values were treated as well. In this case, such values were replaced by the average value of the corresponding data field.

Assemble of daily consumption time series joined data set

The strategy at this point was to employ Dynamic Time Warping (DTW) as a distance-metric into the K-Means algorithm. DTW can be defined as an algorithm used for measuring similarity between two temporal sequences (time-series), which may vary in shape and length. As described by [1] it is possible to efficiently employ DTW into a K-Means to cluster time series of different shapes and lengths, enabling a dynamic and more flexible comparison.

The choice of combining the time series of each household (including all different smart sensors) resulted in a training dataset where each register corresponded to a consumption time series of a given household. The frequency of the readings was resampled to be at a daily level, to make use of all sensors and achieve records with a reasonable number of registers. Table 4 shows the data model of the joined data set at this point, and later, it will be adapted to the accepted shape of the clustering algorithm.

Table 4 Data model for the first clustering algorithm

Data field	Description	Unit
Readings	Water consumption value, joined between all sensor and resampled to a daily value.	Litres/day
Date	Date and time of registered event	Datetime
Reference	The household identification	String
Season	Which season the register belongs to	String

Before applying the clustering, the *Reference* and *Season* data fields will be used to identify different time-series and use them on its respective model. In Figure 3

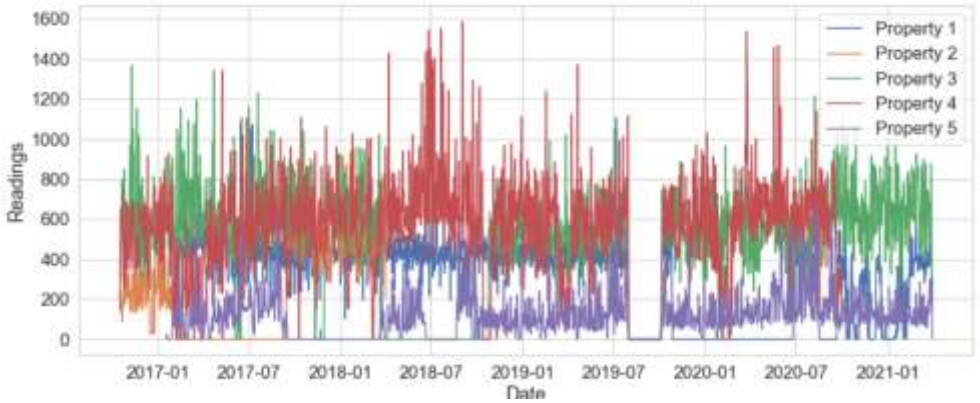

Figure 3 Time-series grouped by properties

Assemble of the household characterization data set

The purpose of the household characterization data set of length (n) of 2715, being each register a reference for a household, is to represent all quantitative and qualitative static (non-time-series) data that is available for each household.

Almost all available data fields were considered resulting in a high-dimensional data set, including for each register (household): a) Average consumption of weekdays and weekend, and weekly values for each season; b) Properties characteristics such as building and garden size; c) Rainfall data; d) Occupancy, social and financial categories (which were converted from categorical variables into binary fields); the data set achieved in the end had around 147 different features. Some of the latter data fields that was previously described can be observed in Figure 4:

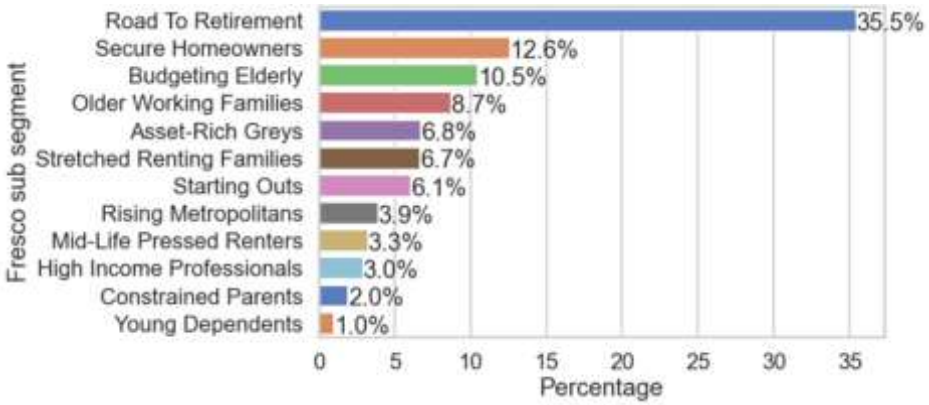

Figure 4 Financial category sub-segment types

Furthermore, this data set is composed of both numeric features and categorical data fields, as can be observed in Table 5 that follows, the categorical features were later converted to indicator variables.

Table 5 Aggregated consumption values

Data field	Description	Unit
Seasonal weekday average	Water consumption average value on weekdays (for each season).	Litres/day
Seasonal weekend average	Water consumption average value on weekend days (for each season).	Litres/weekend
Seasonal weekly average	Weekly water consumption average value (for each season).	Litres/week
Daily rainfall average	Daily average rainfall calculated for each household, based on the closest rainfall station.	mm/day
Max value	Maximum consumption value.	Litres
Mean value	Mean consumption value.	Litres
Percentiles (25, 75)	Percentile values of consumption (25% and 75%)	Litres
Property characteristics	Characteristics include garden and building size, household size, number of children at home, etc.	m^2, integer
Socio-economic categories	General demographic and neighbourhood characteristics (6 categories and 49 sub-categories).	Categorical
Financial categories	Individual power's value, based on a research survey (10 categories and 50 sub-categories).	Categorical

IV.2. Models

The solution is composed of three modules. The first module covers a household consumption clustering based on time-series data similarity using the Dynamic Time Warping (DTW) distance metric, allowing time-series of different lengths and shapes to be compared [2]. In parallel, the second module

processes the static sampled household characteristics and meteorological features aiming to reduce its dimensionality via its latent representation identification. This step is conducted using a Deep Neural Network (DNN) implementation of a type of feedforwarded neural networks called Autoencoder. In this case, it is specifically denominated a DNN because this Autoencoder have more than three layers when considering the encoder and decoder.

Finally, the outputs of modules one and two are combined and clustered to generate groups that are representative of the high-resolution input data set and that are suitable for understanding the household's consumption behaviour.

The solution's architecture is represented in a flowchart format, where each section or model is composed of the input data model and the respective algorithm, as can be seen in Figure 5.

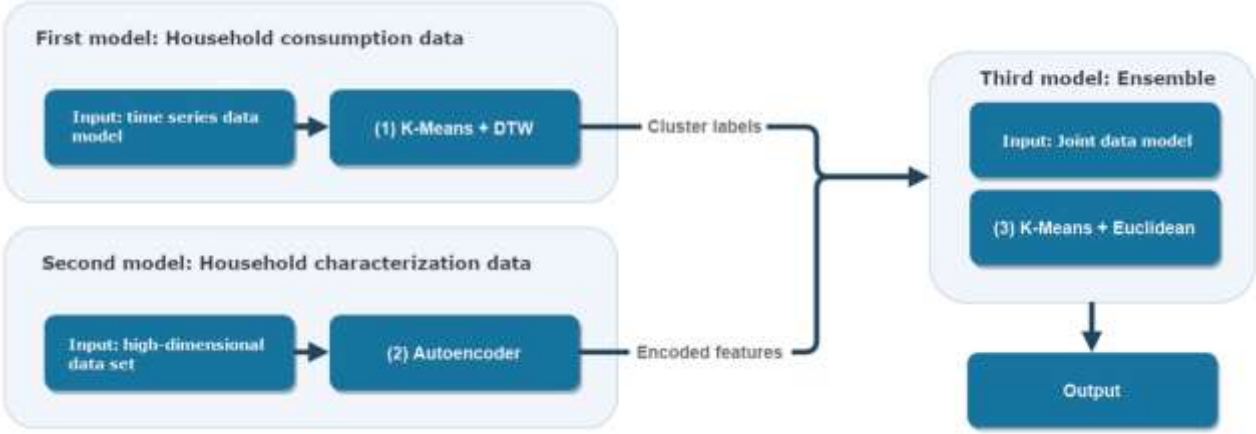

Figure 5 Solutions flowchart

Cluster Assessment (K-Means + DTW)

Different seasons can influence water consumption behaviour, because of that, daily consumption time-series data is segmented into seasons, and individual ML models were generated for each period [3]. The split of seasons was executed following United Kingdoms' season definition as follows: a) Spring was the months of March, April, and May; b) Summer, June, July, and August; c) Autumn, September, October, and November; and lastly, d) Winter was composed of December, January, and February.

Additionally, the time series data were also separated by year. We have decided to work exclusively with the years 2017 and 2018, mostly because 2019 had some gaps on it during the crucial season of summer, and 2020 was thought to be not a representative year due to the behaviour change that the sanitary crisis brought [4].

K-Means is an unsupervised learning algorithm widely used for data clustering. K-Means works by randomly assigning k cluster points in the hypervolume of the data set, then it assigns each data register to the closest cluster centre and finally recomputes the cluster centres based on current memberships, this process is repeated until it converges on the defined criterion.
To find the ideal number of clusters, the following scoring metrics were used:

- **Elbow method (EM):** Heuristic used in determining the number of clusters, consists of plotting the explained variation (also known as the Sum of Squares of the distance between data

points, SSE) against the number of clusters, where it is visible that after the inflexion curve (resembling an elbow) the model fits better.

- **Silhouette Coefficient (SC)**: Compares how similar a sample is within its cluster when in comparison with other clusters, take values between -1 and 1, with values closer to 1 meaning well-separated clusters.

Additionally, this K-Means model was trained using random initialization of centroids.

Dimensionality Reduction and latent space identification (Autoencoder)

It is proposed the use of an *AutoEncoder* (AE), a specific DNN architecture, for the dimensionality reduction task [5]. An AE is widely used for noise reduction and image compression. It is composed of three main segments which are the encoder, responsible for shrinking the original data into compressed data into a second segment known as the bottleneck, and a third one called the decoder, which does the reverse work of the encoder, transforming the low-dimensional data into the original shape. After this process, the output is then compared to the input to see if the DNN managed to reconstruct the data based on its compressed state reasonably well. In this schema, the bottleneck layer is trained to identify the latent space of the input information and can be used for dimensionality reduction purposes with non-linear capabilities.

The strategy is to extract the low-dimensional data from the bottleneck, the latent space, to use as an encoded feature that represents every property. Moreover, this extraction was only made after that the AE evaluation (based on the reconstruction of data) was acceptable.

The layers defined for our DNN were the following: a) Encoder, composed of *Linear* and *ReLU* layers with the input of 147 features and output of 3 features; b) Bottleneck data; c) Decoder set of layers including *Linear* and *ReLU* with the input of 3 features and output of 147 features. This structure is normally adopted for dimensionality reduction. Usually, the bottleneck layer is of a smaller size for ensuring that the dimensionality will be reduced after the encoder layer. Aside from the structure, some hyperparameters were optimized for this NN:

- **Optimizer:** Adam, is an extension of stochastic gradient descent that is broadly used for deep learning applications.
- **Learning Rate:** 1e-2, parameter tuned to achieve an acceptable performance after 1000 epochs, values considered on hyperparameters optimization: 1e-2, 1e-3 and 0,1.
- **Weight decay:** 1e-5, a parameter used to penalize complexity, being a way to contour problems of underfitting or overfit in NN. Values considered on optimization: 1e-5, 1e-4, and 1e-3.
- **Criterion (Evaluation):** MSE Loss, measures the loss as Mean Squared Error between each element in input x and target y.

Clustering Ensemble (K-Means + Euclidean)

Ensemble techniques applied to supervised and unsupervised learning algorithms can result in a better performance overall [6]. More specifically, combining algorithm outputs for the execution of a K-Means can provide clusters with a higher similarity and quality.

Joining the data is an important step of the study, and that is accomplished by converting the cluster labels from the first model (FM) to indicator variables (binary) and appending the encoded features

extracted from the latent space of the AE. The joining is made at a household level, meaning that each register that represents a property, have both outputs combined and used as training data for the next K-Means model. The respective data model can be observed in Table 6.

Table 6 Autoencoder datamodel

Data field	Description	Unit
Encoded feature 1	First AE encoded feature.	Float
Encoded feature 2	Second AE encoded feature.	Float
Encoded feature 3	Third AE encoded feature.	Float
Label 2017	Cluster label that was assigned to the 2017 model (K-Means + DTW)	Integer
Label 2018	Cluster label that was assigned to the 2018 model (K-Means + DTW)	Integer

The two latter variables were converted into binary indicator variables, so, finally, the data set would be composed of 18 features. Once again, one data set per season was made to train the individual models (one for each season).

Like the FM, this K-Means implementation adopted the random initialization of centroids and selection of value *k* made by analysing the EM, SC, and Calinski Harabasz (CH) metrics. Moreover, this model uses the Euclidean method as a distance metric which is widely adopted by default in the industry.

Another different step in this model was the necessity of scaling the data set beforehand due to the different nature and numeric scale of joined outputs which composes the training data set.

IV.3. Evaluation

First model (K-Means + DTW)

As defined earlier, four different K-Means models have been generated, each one with its own data set (separated by season). The choice of the value *k* came based on the analysis of the values of the EM (SSE) and the SC. The value can be observed in Table 7.

Table 7 Evaluation of the first model

Season	SSE for k = 6	SSE for k = 7	SSE for k = 8	SC for k = 6	SC for k = 7	SC for k = 8
Spring	1.3342	1.2845	**1.2760**	0.0535	0.0650	**0.0476**
Summer	1.3383	1.2841	**1.2868**	0.0573	0.0536	**0.0456**
Autumn	1.3534	1.2821	**1.2797**	0.0567	0.0587	**0.0609**
Winter	1.3228	1.3233	**1.2874**	0.0505	0.0458	**0.0471**

The values marked in bold were the choices for the number *k* of clusters. Internal discussions were held with the water utility in which was decided to keep a low number of *k* clusters, with the objective of making the analysis easier and simpler to understand. So, based on it and the metrics obtained (higher value for SC and lower value of SSE), the chosen *k* value for each model was: 8.

Second model (Autoencoder)

The data that resides in the bottleneck is the one that will be used as a low-dimension version of the data, given that the decoder made a good job reconstructing it. The Autoencoder also allows for use of non-linear data. On Table 8 it is possible to see the number of trainable parameters for each layer of the DNN, the values marked in bold represents, respectively: dimensions of input, bottleneck layer dimension and the decode output dimension (same as original data).

Table 8: Autoencoder's Trainable parameters table

Module	Trainable parameters
Input	**148**
`encoder.0.weight`	9472
`encoder.0.bias`	64
`encoder.2.weight`	768
`encoder.2.bias`	12
`encoder.4.weight`	36
`encoder.4.bias`	**3**
`decoder.0.weight`	36
`decoder.0.bias`	12
`decoder.2.weight`	768
`decoder.2.bias`	64
`decoder.4.weight`	9472
`decoder.4.bias (output)`	**148**

The AE was evaluated by measuring the loss metric obtained by comparing the reconstruction made by the decoder and the input data, through all the epochs of training, as defined earlier. Moreover, an additional measuring during the validation phase was implemented. Those values can be observed in Table 9 and in Figure 6. Additionally, 70% of the data set was used to train the model, while the other 30% was used to validate it.

Table 9 Autoencoder epoch evaluation

Epoch	Train loss (MSE)	Validation loss (MSE)
1	0.9893	0.4063
200	0.4872	0.3547
600	0.4071	0.3285
1000	0.3889	0.3131

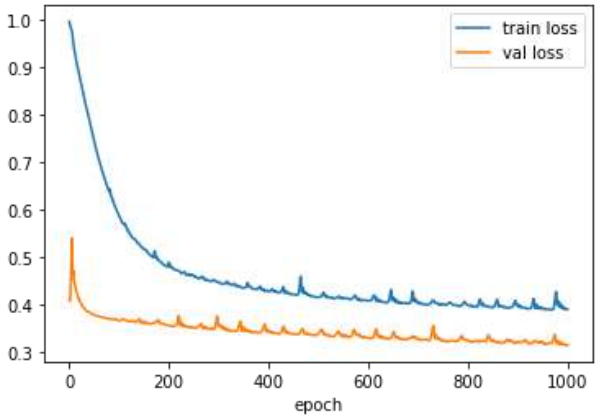

Figure 6 Autoencoder loss curve

Third model (K-Means + Euclidean distance)

To evaluate the ensemble model, three different metrics are employed. The previously discussed EM and SC, but also the Calinski Harabasz index (CH), which is a metric best suited for Euclidean distance-based clustering.

- **Calinski Harabasz Index (CH):** Also known as Variance Ratio Criterion (VRC), it represents the ratio between the within-cluster dispersion and the between-cluster dispersion, higher values mean well-separated clusters.

As can be observed in Figure 7, the SC values for each model differ, but, overall, they represent well-defined clusters while maintaining a reasonable low value for *k*. Moreover, the same figure also shows the EM values for the ensemble model.

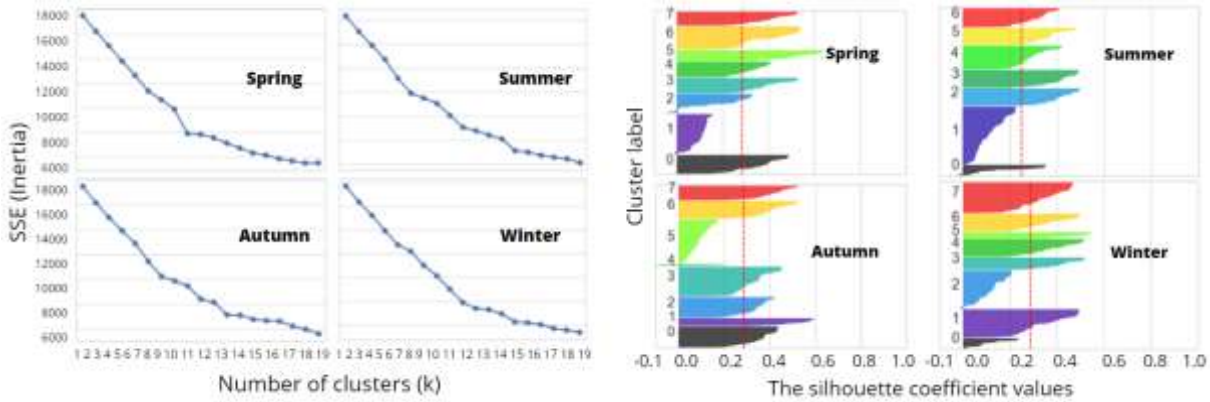

Figure 7 (a) Elbow and (b) Silhouette evaluation for the different clusters

Given those analysed metrics and discussions related to the number of clusters, the optimal number of k chosen for the third model (TM) was, respectively: 8, 7, 8 and 8.

Next, in Table 10, a summary of the evaluation metrics for each ensemble model is represented, based on that, it is possible to conclude that the spring and autumn models achieved the highest metrics and the summer model, had the worst performance overall.

Table 10 Evaluation of the final model

Season	No. of clusters	SSE	SC	CH
Spring	8	9855.568	0.28443	112.6678
Summer	7	11484.491	0.25049	106.3353
Autumn	8	9871.417	0.28479	111.5321
Winter	8	10120.917	0.28838	109.7568

On the other hand, one chosen form of validation to compare the first clustering result (K-Means + DTW) with the ensemble model, and to identify and observe the impact that the *encoded features* finally had on the final set of clusters, is the Sankey Diagram (SD). The SD is useful to observe continuity between distinct sources, and it is represented in Figure 8, where FM means First Model and TM means Third Model.

The TM cluster 2 is a brand-new cluster, being composed of many properties that before were assigned to another cluster, but, in general, we can affirm that the remaining clusters were assigned close to the result of the FM. Moreover, we can conclude from the SD that the encoded features had an impact on the new assignation of clusters, but without changing most of the clusters defined by the FM.

It is important to note that, this proposed solution can be easily expanded by adding more years of data, which can result in a better clustering result.

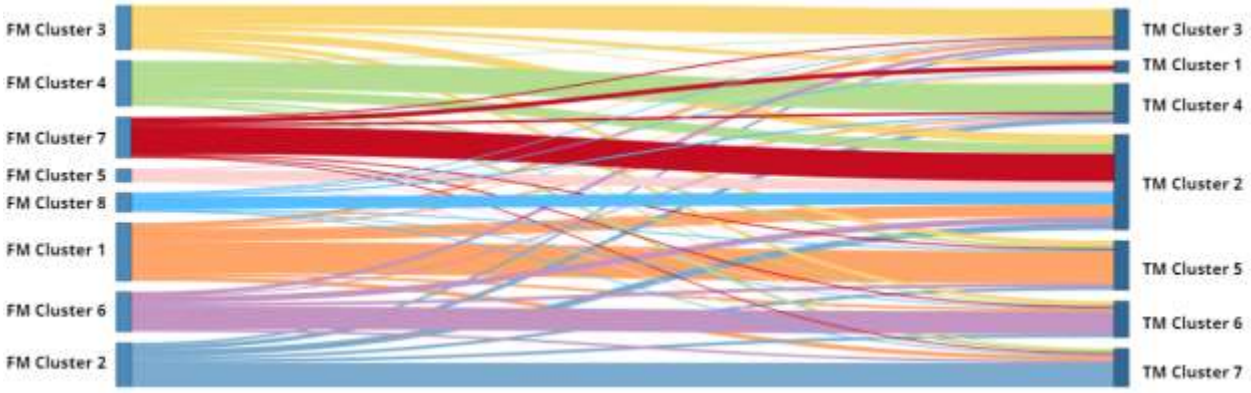

Figure 8 Sankey Diagram, continuity of clusters between models (considering summer models)

To expand the analysis, Figure 9 was developed by calculating the mean value of some properties per cluster, to produce some comparative base between the different assigned clusters.

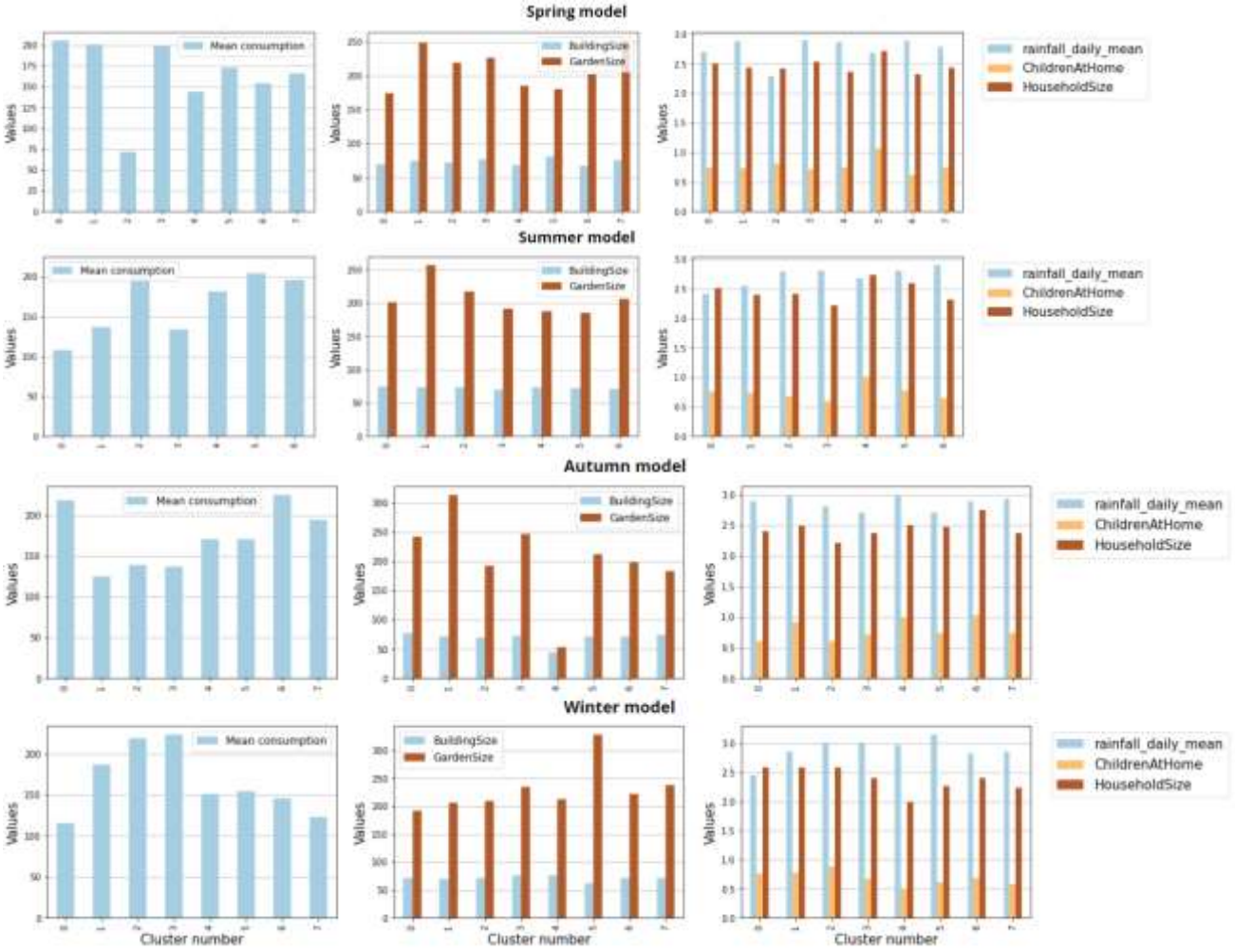

Figure 9 Aggregated features of the different output clusters for each season

The values were calculated by taking the mean value of a given feature considering all properties that resides within the respective cluster.

Lastly, a plot showing the distribution of kind kernel density estimation (KDE), which is a non-parametric way to estimate the probability density function of a given variable, in our case of the water consumption variable in each cluster. On the x-axis we can observe the different models for each year and on the y-axis each cluster, and can be observed in Figure 10.

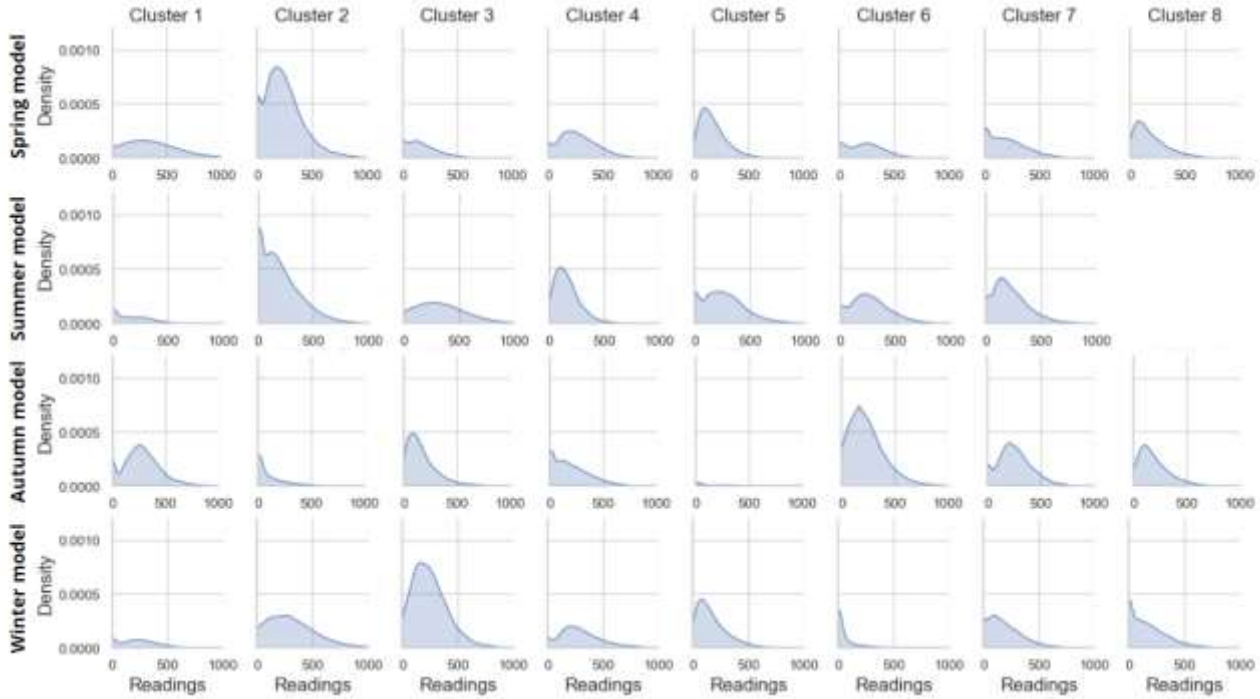

Figure 10 Cluster distribution for each season

Through the normal distribution of this variable, it is possible to observe which clusters concentrate the properties with the highest consumption in each model/year and can serve as a base for comparation when considering only the water consumption feature.

V. Data Visualisation

A key goal of data visualisation and presentation in the demo case was to share data between utility and customer side apps such that customers would not be surprised by data plurality, i.e. SWW staff reporting different values to those that customers saw on the customer app.
To achieve this goal, Stellio was the data authority for both utility and customer-side applications.

V.1. Utility Side Data Visualization

Key goal of the utility app was to provide utility staff with a way of determining which consumers were likely to have leaks in their properties. Although the alarmFlowPersistence meter property can give some indication into potential leaks, utility staff developed an approach that also incorporated the consideration of multiple properties:

1. Number of leak alarms (alarmFlowPersistence) over a given period
2. Consumption over a period compared with previous period
3. Weekly consumption over a given value

These considerations were wrapped into queries for Stellio with the results presented in numerical and graphical formats, Figure 11.

Figure 11: Utility app leak reporting

V.2. Customer Side Data Visualization

The key goals of this app were: 1) give customers an insight into their daily and historic consumption and 2) give customers an insight into how their consumption measures within the overall consumer group of Gt. Torrington, with consumption data presented numerically, and graphically.

Figure 12 details screenshots of the customer app. The image on the left shows the daily consumption, with the customer's consumption for the previous day (50 litres on 15/11/2021) and a summary of consumption for the entire pilots, detailing the total consumption for that day, the average consumption per customer and where the customer's consumption ranks within the Great Torrington group (2nd highest out of the ten). In addition, the daily usage records the lowest (above zero) consumption for the group and the highest, to give customers a clear idea of where they sit within the group.

The centre image shows historic consumption (set over the last 31 days) and provides the customer with a similar breakdown of consumption in the group, but over a longer timeframe.

The image on the right shows a graph of consumption over the same period.

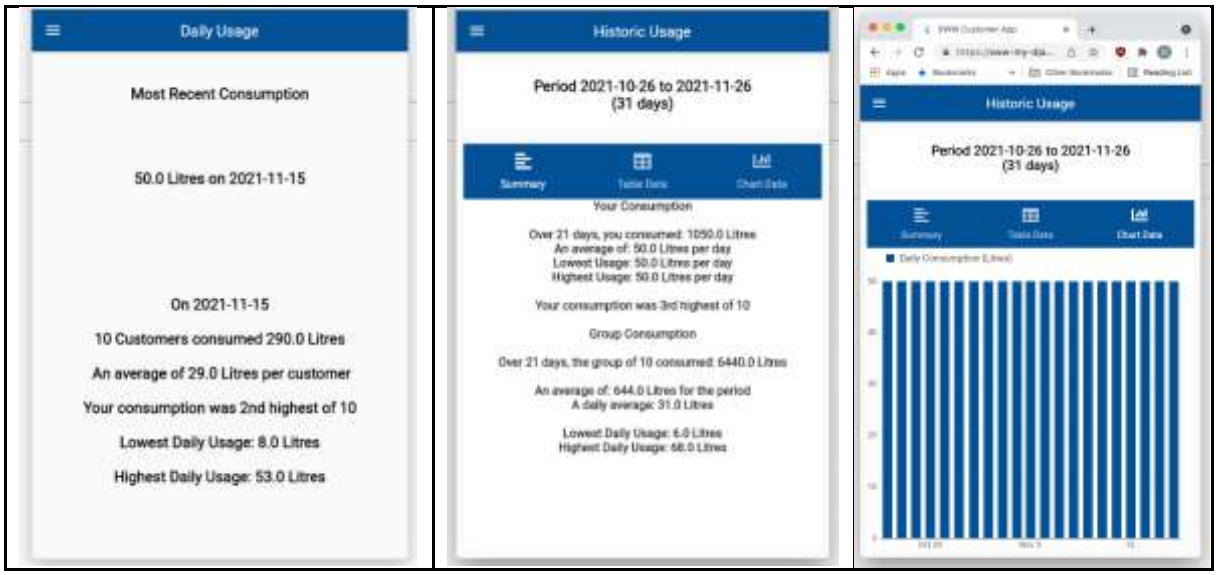

Figure 12: Customer App showing daily consumption (left), historic consumption (centre) and consumption graph (right)

VI. Platform Optimization

VI.1. Data processing

Initial development work with Stellio was centred on a 'single customer' approach, with requests for a customer's current or historic consumption, using the ngsi-ld/v1/entities and ngsi-ld/v1/temporal.entities requests.

However, iterating through customers to collect all current or historic data resulted in long periods of data collect as http requests were queued up. This was largely addressed by threading http requests, though this provided to be largely insufficient when multiple requests were required, particularly for building the customer app 'consumption narrative' where a customer's consumption was ranked against group consumption.

One approach to address this in the future, in particularly with the open-ended queries of the utility-side app was moving Stellio data into a PostgreSQL database and using SQL to query data. Although Stellio and ngsi-ld/v1 do provide some support to query the underlying context broker, missing consumption data makes this a non-trivial exercise and running a daily PostgreSQL building process allows data to be suitably processed.

It is assumed that there is a more efficient data management approach in extending the Stellio dataset to store daily consumption in an entity that is separate to the WaterConsumption model, though our limited understanding of Stellio and a desire to avoid updating the live Stellio broker left this approach untried.

VI.2. Data visualisation: Utility-side App

The utility-side application was designed and implemented to be run in a standard desktop landscape mode browser within the South West Water infrastructure and has been implemented as a fairly

responsive application, insofar page content will adjust to reflect differences in screen size and aspect ratio to a large degree, Table 11: Utility-side app responsive presentation, near portrait (left) and landscape (right), and where this is not possible, the browser will make the screen scrollable.

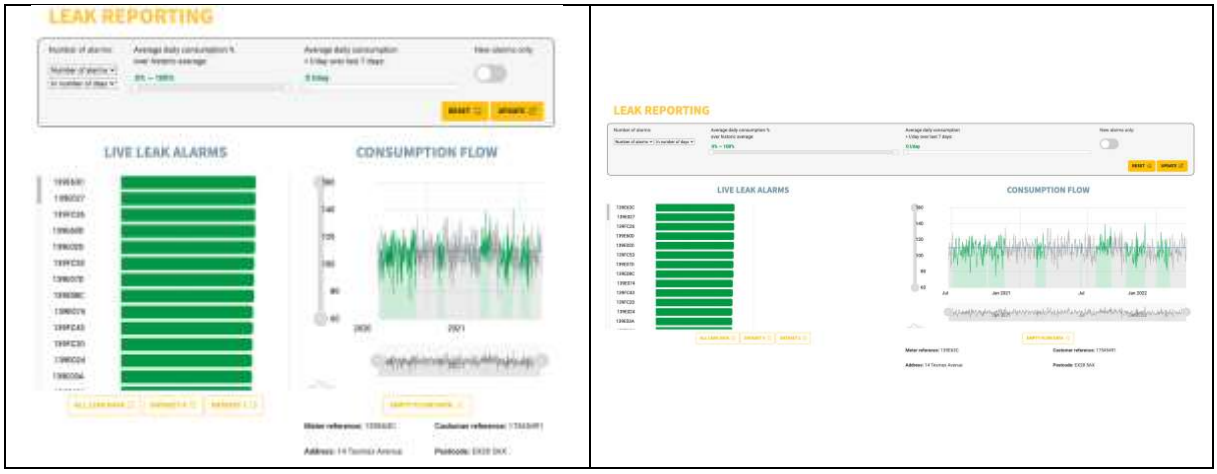

Table 11: Utility-side app responsive presentation, near portrait (left) and landscape (right)

VI.3. Data visualisation: Customer-side App

The customer-side app was designed (D4.5, V.4) as a portrait app for smart phones and implemented in Google's Flutter API which supports responsive UI presentation to a large degree. The app has been configured to present only in portrait mode and is unresponsive to changes in orientation, i.e. the UI will remain in portrait mode if the phone is rotated to landscape mode. The app does not support tablet presentation modes (4:3 / 3:4).

Much of the Flutter UI was developed using the flexible(flex) class which allows UI screens to be subdivided in logical, rather than physical, estate (like Bootstrap's grid system). This allows Flutter to resize screens around 'similar' aspect ratios, e.g. broad portrait of 9:16 to 9:19.5, without losing content off the edges of the screen. Table 12 shows how content is scaled across portrait devices of differing resolutions and aspect ratios

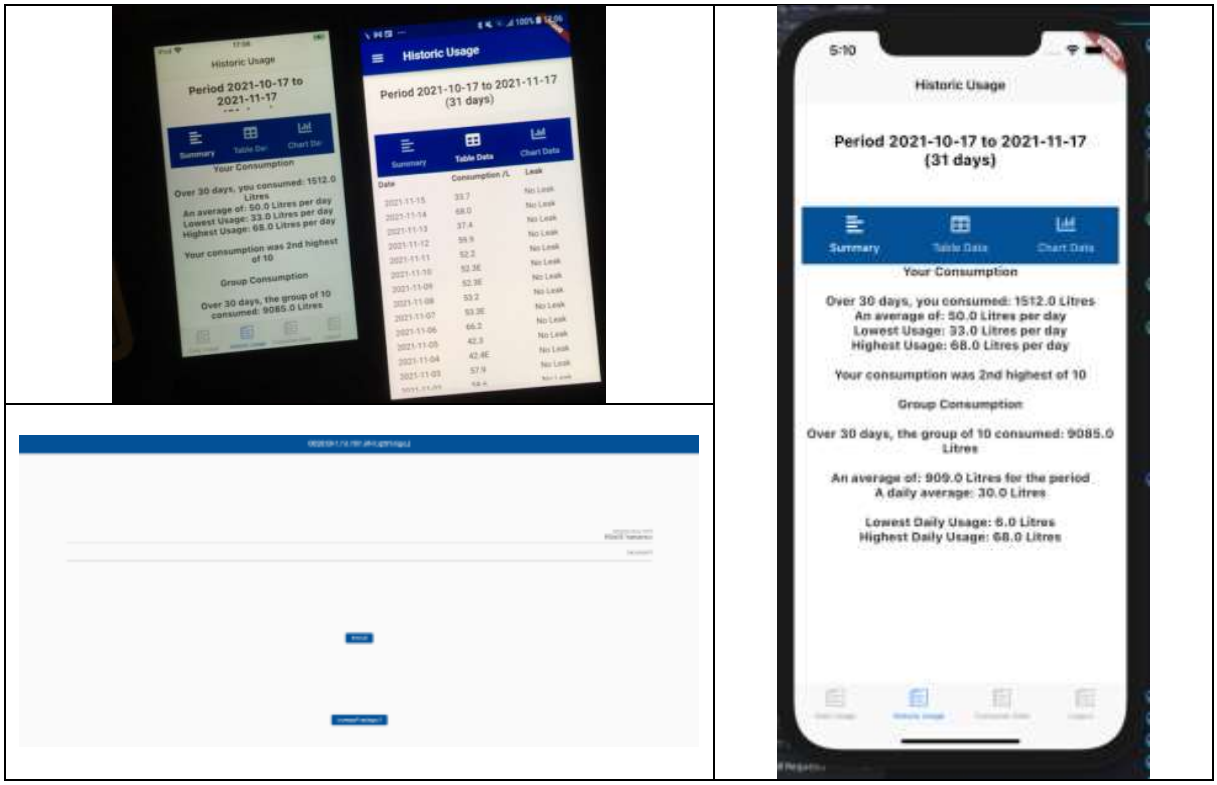

Table 12: Flutter responsive apps, android and iOS(top and right), browser flexing content (bottom)

A 'development testbed' version of the Flutter app was created as a testing web page for the customer-side Django app. As a landscape mode application, the portrait mode Flutter app was effectively stretched to fit the real estate available (as part of the flexible approach), Table 12. This could be addressed if the app was to be provided as a web page

VII. Conclusion and Perspectives

VII.1. FIWARE / Stellio

In general, Stellio worked well for the project and FIWARE appears to be a very useful framework for hosting smart metering data. In particular, storing entities as JSON data completely removed the need to create explicit database tables and the temporal evolution (ngsi-ld/v1/temporal/entities) removed much of the need to do complex data management.

There were little to no issues with working with the Stellio context that EGM set up, though there appeared to be occasional issues with the Sigfox / IOT Agent connection where data updates were 'lost' for several periods in the project.

The main operational issue with Stellio stem from requirements to perform multiple temporal requests across the entire smart meter dataset in order to determine total consumption or total leak counts. Attempting to perform these requests in a 'for each smart meter 'approach took a significant amount of time, >1min for 100 smart meters, as requests were all queued on completion. Whilst this was reduced by threading calls to the broker, it suggests either a poor choice of algorithm or a weak point for the broker, it is suspected that it's more an algorithmic issue that attempts to use the broker in a way it is not designed.

VII.2. Development environment

For this project, the Stellio broker environment was treated as a 'live' service and care was taken to not 'break' the service, with much of the app development being undertaken either using the live Stellio service as a data source or working in custom development environments.

Using the live service as a data source worked for much of the development process, except where there were needs to create edge case data in order to generate leak alarms, excessive consumption and so on. For these situations, a separate Stellio broker was created and populated with synthetic data. This was achieved by creating local Docker applications running the Stellio docker-compose settings.

The resource requirements of Stellio did make it difficult to run remote Stellio instances, with Stellio's failures being difficult to interpret.

VII.3. Limited understanding of FIWARE/docker

Many of the development issues encountered in this project stem from limited understandings of FIWARE and containerisation and it would have been extremely helpful to have had more orientation at the beginning of the project in both areas.

Although having Stellio and Sigfox setup at the beginning of the project was a great timesaving, it did create a mindset of a 'live' environment that should not be developed on, for fear of breaking it and losing live data. This led to development approaches that were not optimal, in particular, in how the demo case deals with processing Stellio data.

In hindsight, it would have been preferable to perform infill and leak processing as notification when Stellio received new meter data for each meter, rather than taking the approaches that have currently been implemented. However, to use notifications, URLs would need to be provided for the broker which couldn't have been done easily in a development environment, and it would have meant writing entities to the live service.

The project would have benefited from developing tools to archive and rebuild broker contents in order to have data security, which would have made it far less of an issue to add to and extend Stellio entities as well as setting up development and test environments. It also would have been extremely helpful to have been able to capture and duplicate Sigfox data.

The broad solution to these issues would have been to create a more typical dev-test-live environment where functionality could be developed in the development environment, then moved to a test environment, and finally published to the live environment. Docker can be an enabling technology for this approach to devops and it was a technology we came too late in the project.

VII.4. Django (app server development)

Alongside Stellio, the project used Django as a server framework for both the utility and customer-side applications and was selected as South West Water's preferred web development framework. In general, Django worked very well for the task and provided a lot of functionality to aid development.

As an established and industry-standard framework, there is a lot of support for Django on the internet and most of the 'novel' problems that we encountered had already been encountered before with a multitude of solutions and work arounds available, particular in areas such as integration with Flutter and mobile http requests.

The project used independent Django servers for both apps (utility and customer) as a broad security measure to ensure that the customer request framework could not be used to access utility only data. Both servers used a shared python package that initially provided an interface to Stellio, so that apps could not directly communicate with the Stellio broker. This was revised towards the end of the project with the implementation of an intermediary postgres server that stored daily customer data to enable the utility broker to make more complex leak / usage queries.

VII.5. Flutter (mobile app development)

Flutter was used to develop the mobile app and was selected, in part, for its ability to create reactive, cross-platform (android, iOS and web) UI.

In general, Flutter worked well and has a lot of support (mainly from Google) but does give the impression that it superseded by react.native.

Flutter's USP is that it is a responsive and cross-platform framework and whilst the amount of platform-specific code is fairly small, there are still platform-specific requirements, particularly with how both

platforms (iOS and Android) deal with app layout. This can be seen in Table 13, where the iOS application to the left has 4 ‘action’ icons at the bottom of the screen, whilst the Android application to the right has the same functionality in the hamburger menu to the top left of the application.

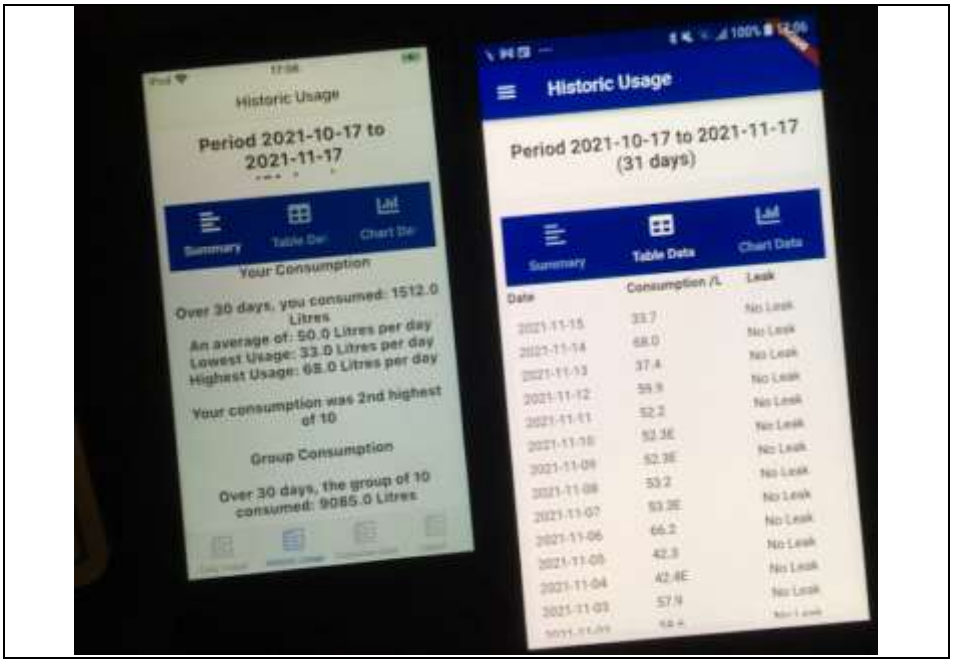

Table 13: Flutter layout differences, iOS (left) and Android (right)

Flutter’s responsive presentation engine works well with content generally being resized and organised correctly for different resolution and aspect ratio mobile screens, though this can lead to large amounts of white space and UI designed for portrait mode screens tends not to automagically resize for landscape mode, but this is to be expected. The cost of Flutter’s responsive presentation is that UI components are developed in mark-up code which tends to be time-consuming to create and iterate and it would make more sense to use a visual editor to layout UI in a more aesthetic manner.

Flutter uses Dart as its underlying programming language which is transpiled into JavaScript. In general, Dart is a fairly clunky language which concentrates on the wrong things (code layout) and obfuscates straightforward JavaScript concepts (promises) into ‘Future’, which results in awkward http.request processing workflows.

Flutter also highlights development issues with iOS and Android development. Whilst AndroidStudio allows debug apps on both mobile platforms to be debugged directly, the apps only exist during debugging sessions and the applications must be built onto the phones to be used outside of AndroidStudio. For Android, this is a very simple process, the build option is selected and an apk file is generated. This can then be installed on any android device. For iOS, the build option results in the creation of an xCode project which must then be loaded into xCode on a mac and the project built. To install the app onto iOS developers, they must be either setup for development or available through TestFlight. Generally, this makes developing for iOS a time-consuming activity. Android should be considered as the primary development platform, just purely to speed of iteration.

VII.6. IT policies

For this kind of development (context brokers, servers, and mobile apps) it's apparent that development needs to be geared around the availability of internet enabled, and secure, servers that developers have access to. For much of the project, development was only really possible on local development machines and low-performance servers that were incapable of hosting Stellio.

Development experiences with Heroku showed that whilst it's possible to host services on remote infrastructure-as-a-service services, costs ramp quickly and the services tend not to provide quite what is really needed for development. Some development experiments were undertaken with Azure, but nothing enough to provide any concrete conclusions.

VII.7. VII.7. EU added-value and upscaling possibilities

Within F4W project, a lot of actions, being technical (WP3 and WP4) or non-technological (WP5), were carried out in the city of Great Torrington. Both the water utility South West Water and the citizens have directly benefited from the good results of these actions which constitutes an important EU-added-value.
For example, having installed around 100 smart meters in households of the city of Great Torrington (medium class, not metered a lot) after having informed and then convinced volunteer citizens notably by organising public meetings was the starting point for raising awareness of citizens about the value of water. Then, having enabled pipeline to retrieve consumption data from smart meters and provide extracted analytics to customers via a smartphone application to drive positive changes in water use behaviour, reduce consumption and reduce the customers water bill directly contributed to the sustainable development goals SDG 1 "end poverty" and SDG 6.b "Citizen participation in decision-making". In addition, having explored, looking at the use of this data by the utility provider, South West Water, to detect customer leaks and manage the repair/replace work lifecycle using an interactive web application linked to the FIWARE system fully participated at the green-digital transition of Europe.

In terms of replication and upscaling of the models and tools developed in this EU demo case during the F4W project, it can be envisaged. Hence, the work completed as part of this F4W demo case in development of a utility application and customer smart meter application will be directly re-useable by South West Water for other customers across their region. South West Water has already committed to the installation of an additional 65,000 smart meters across their North Devon region by 2025 since this project completed.
The Utility application is built so that any new smart meter installations will be visible within the application and therefore the underlying approach of searching, filtering and prioritising based on leakage volumes will continue to be used. Similarly, the underlying code behind the customer smart meter application could be re-useable, with minimum effort, should South West Water or another utility continue with the development and deployment of the customer smart meter application.

VIII. Recommendations

This section of the reports draws on the experiences of the development project to highlight recommendations for best/better practice for future projects

VIII.1. Working with FIWARE

FIWARE has been an incredibly useful technology for this project, providing a straightforward and generally easy-to-use environment for managing data, in particular time series data, without the need to resort to SQL/no SQL databases. However, the FIWARE paradigm can be difficult to follow. It is recommended that developers that are new to FIWARE should look to develop their familiarity with FIWARE through small learning activities prior to engaging on project-based activities. The Stellio 'beehive' API walkthrough (https://stellio.readthedocs.io/en/latest/API_walkthrough.html) is a good starting point and the CIM NSGI-LD specifications (https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.05.01_60/gs_CIM009v010501p.pdf), though it is worth noting that not all context broker implementations follow the specification exactly.

VIII.2. Data Collection and interpretation

There was an initial assumption within the project that data collection would be completely reliable, though this was soon abandoned given gaps in data collection, typically caused by vehicles parking over ground-based meters blocking data transfer. For this project, it was not a huge issue as the key data, 'total consumption', would eventually be transmitted to the FIWARE broker and the nature of the data was such that missing consumption data could be 'infilled' along a straight-line estimate. However, the same could not be said for the continuous flow attributes which would limit the performance of leak detection.

The resulting recommendation is that consideration needs to be given to the data being collected, the impact of data loss and the likelihood and approaches, if appropriate, for data reconstruction and interpolation.

VIII.3. Development enviroments

Much of the initial development in this project was geared around applications interacting with the FIWARE broker. Whilst this worked well for early development activities, it did become increasingly difficult to work in testing, particularly with edge case generation and detection as there was a strong desire to keep testing data away from the live data in the broker.

It is recommended that developers working with FIWARE should look to implement a devops style environment, such that functionality can be developed, tested and operated in separate, but functionally and structurally environments. We found that moving to a containerised environment with Docker enabled a far more compartmentalised approach to development. It is also recommended for developers that are looking at containerised environments should look to develop their Docker expertise before moving onto a live development project and avoid using a current project as a learning environment.

VIII.4. Data security

This recommendation is closely related to section VIII.3, in that developing in a purely live environment raises clear issues about data security, in that if errors are made to the live FIWARE broker, the result could be the loss of live data. This creates a broad recommendation that consideration should be given to data security, to ensure that live data is not lost.
In addition, a data policy should be developed to determine how data can be duplicated from a live server into test environments and how data can be backed up, achieved and recovered as necessary

References

[1] X. Huang, Y. Ye, L. Xiong, R. Lau, N. Jiang and S. Wang, "Time series k -means: A new k -means type smooth subspace clustering for time series data," *Information Sciences,* Vols. 367-368, pp. 1-13, 2016.

[2] S. Aghabozorgi, S. Seyed Shirkhorshidi and T. Ying Wah, "Time-series clustering – A decade review," *Information Systems,* vol. 53, pp. 16-38, 2015.

[3] A. Ibrahim, F. Memon and D. Butler, "Seasonal Variation of Rainy and Dry Season Per Capita Water Consumption in Freetown City Sierra Leone.," *wATER,* vol. 13, no. 4, p. 499, 2021.

[4] A. Kalbusch, E. Henning, M. Brikalski, F. Luca and A. de and Konrath, "Impact of coronavirus (COVID-19) spread-prevention actions on urban water consumption," *Resources, Conservation and Recycling,* vol. 163, 2020.

[5] Y. Wang, H. Yao and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing,* vol. 184, pp. 232-242, 2016.

[6] Z.-H. Zhou and W. Tang, "Clusterer ensemble," *Knowledge-Based Systems,* vol. 19, no. 1, pp. 77-83, 2006.