A novel smart grid architecture that facilitates high RES penetration through innovative markets towards efficient interaction between advanced electricity grid management and intelligent stakeholders

H2020-GA-863876

Final version of FLEXGRID S/W prototype

Deliverable D6.3

Document Information

Scheduled delivery	30.06.2022
Actual delivery	15.07.2022
Version	Final
Responsible Partner	ETRA

Dissemination Level

CO Confidential, only for members of the consortium (including the Commission)

Contributors

Lars Herre (DTU), Rahul Nellikkath (DTU), Spyros Chatzivasileiadis (DTU), Elena Leal (ETRA), Ana Isabel Martinez (ETRA), Prodromos Makris (ICCS), Dimitrios Vergados (ICCS), Konstantinos Steriotis (ICCS), Nikolaos Efthymiopoulos (ICCS), Maria-Iro Baka (UCY), Marios Kynigos (UCY), Andreas Kyprianou (UCY), Stylianos Loizidis (UCY), Christina Papadimitriou (UCY), Spyros Theocharidis (UCY), Domagoj Badanjak (UNIZG-FER), Vesna Županović (UNIZG-FER)

Internal Reviewers

Gesa Milzer (NODES), Prodromos Makris (ICCS)

Acknowledgements

The research leading to these results has received funding from the EC Framework Programme HORIZON2020/2014-2020 under grant agreement n° 863876.

Glossary of Acronyms

Project management terminology

Acronym	Definition
D	Deliverable
HLUC	High Level Use Case
MS	Milestone
WP	Work Package
UCS	Use Case Scenario

Technical terminology

Acronym	Definition
AFAT	Automated Flexibility Aggregation Toolkit
API	Application Programming Interface
ATP	Automated Trading Platform
B2B/B2C	Business to Business / Business to Consumer
DFMCT	Distribution Flexibility Market Clearing Toolkit
DSO	Distribution System Operator
ES	Energy Service
ESP	Energy Service Provider
FMCT	Flexibility Market Clearing Toolkit
FMO	Flexibility Market Operator
FSP	Flexibility Service Provider
FST	FlexSupplier's Toolkit
GUI	Graphical User Interface
MTU	Market Time Unit
PV	Photovoltaic
REST	REpresentational State Transfer
TSO	Transmission System Operator

Specific terminology definition

Word	Definition
Baseline	The baseline shows the scheduled demand during the selected date
Cost	Amount of money an actor will have to pay for something
Price	Monetary value of something
Revenue	Money generated from participating in a market or a service
Benefit	Difference between costs and revenues

Table of Contents

List of Figures

List of Tables

Document History

This prototype deliverable (DEM) includes the final version of the S/W integration and validation results of FLEXGRID platform. An initial version was demonstrated during the 1st official review meeting (i.e. Month 20), while the release of the first integrated FLEXGRID system prototype took place in Month 24. The final version will be demonstrated during Period 2 review meeting and in several workshops/webinars/special sessions/events within M33-M36 period.

Table 1: Document History Summary

Revision Date	File version	Summary of Changes
28/01/2022	v0.1	Draft ToC
18/02/2022	v0.2	ToC approved for all partners
24/05/2022	v0.3	First round of contributions
25/05/2022	v1	First version for revision
20/06/2022	v1.2	Second round of contributions
05/07/2022	v1.3	Final round of contributions
08/07/2022	v1.4	Final version for revision
15/07/2022	v2	Final version for submission in ECAS

Executive Summary

This report is an official deliverable of H2020-GA-863876 FLEXGRID project dealing with the final version of FLEXGRID S/W prototype. It includes the outcomes of task 6.2 “Design of APIs and S/W Development” and task 6.3 “GUIs and integration activities” and the work reported also in D6.2. Along with this document, the final version and GUIs for the UCS presented on FLEXGRID and the main ATP are deeply explained.

For this final version showing the final design and results of the ATP and its modules, it has been integrated the UCS and algorithms developed along the whole project. The results on this document are based on the results from task 6.1, and WP3, WP4 and WP5 algorithms’ developments.

It should be highlighted that part of the work done during the first phase of the project (until M18) and reported on D6.2 (M24) is the basis for the developments done until M33 and delivered here. The work has been performed within T6.2 and T6.3 involving the different research partners on the adaptation of the mockups and APIs to be fully integrated on ATP developed by ETRA.

To address all relevant aspects to achieve the scope of both tasks, the deliverable is structured in 6 different chapters.

The first chapter deals with the executive summary of the deliverable contents, the description and definition of the FLEXGRID APIs and the toolkit of the GUIs that are being developed within the project, as well as the definition of the methodology used for the implementation of the task here executed.

The second chapter sums up the developed Use Cases Scenarios (UCS) showing the relation between them and the main ATP configuration available for each one with the objective of being an introduction for understanding the API and GUI used for the main platform of the project.

The third chapter is the core of the document, where the APIs and central data base integration on the ATP is described, both from the UCSSC perspective and the ATP. The Flexgrid SW architecture is defined and deeply explained to show how all the modules in ATP are interrelated.

Chapter 4 includes the developments regarding the GUI presentation, in particular considering the different functionalities that the ATP as a whole have and specifically the main tools that the owner of the ATP (i.e. administrative user) can use to manage users and services.

Chapter 5 shows per module and UCS the different GUIs (configuration, results and historical) to run the algorithms developed within the WP3, WP4 and WP5. The results of the algorithms developed and integrated on the ATP with an example per UCS is also here presented

showing how the results from algorithms tested on previous WPs are validated also via the ATP.

Chapter 6 describes the FLEXGRID software architecture to correctly implement the required APIs in T6.2. Furthermore, this chapter explains the integration of all UCS within the ATP and the current status of each API based on the methodology defined based on the work done by research partners in WP3 “Automated flexibility aggregation energy market development and management as a service”, WP4 “Innovative ESS aware Business Modelling for ESPs and interaction with advanced RES & Market Forecasters” and WP5 “Optimal Power Flow and interaction between network operators and markets”.

Finally, some conclusions are presented, containing a summary of the main results of the work performed and presented in the current deliverable.

1 Introduction

1.1 Purpose of the document

The main objective of this report is to demonstrate the final version of the FLEXGRID software, and the results obtained from the algorithms developed and included on the ATP interface. Through the different APIs it is possible to establish the communication protocol to allow interaction and data expansion over the algorithms and interfaces included on the ATP and show the results in the correspondent GUI making possible to the different type of users to interact in an easy way with all the functionalities that the FLEXGRID platform offers.

The report shows the final version of the ATP software as well as the GUIs developed to monitor the offline results (i.e. based on "what-if" simulation scenarios) obtained from the algorithms developed in WP3, WP4 and WP5. To this purpose a clear definition of the UCS implemented and their functionalities are defined.

As in previous deliverables, throughout the entire document, the different market actors and their present and future options are presented. They are categorized as FMO, DSO, ESP and aggregator users providing for each one of them a proper graphical interface (GUI) so that they can interact with current markets and future ones to enhance the flexibility and resilience of the grid. The principal goals of the project are: i) to provide access easily and effectively to advanced Energy Services (ESs), ii) to facilitate a dynamic and efficient interaction with the electricity grid and the stakeholders, and iii) to automate and optimize the planning and the operation of their ESs. All these are covered thanks to the API integration.

1.2 Scope of the document

This document presents the final version of the S/W integration and validation results of the FLEXGRID ATP into the context of the tasks 6.2 and 6.3. Both tasks officially started on M18 and M25 respectively. Based on the first version of the software and the information included in D6.2 a final version of the ATP is here described and will be demonstrated during the final review of the project (M36).

As in D6.2 the outcomes of tasks 6.2 and 6.3 described in this deliverable are based on previous work mostly based on the requirements definition and the information included in D6.2 and others:

- D2.1 – "FLEXGRID use case scenarios, requirements' analysis and correlation with innovative models"
- D2.2 – "The overall FLEXGRID architecture design, high-level model and system specifications "
- D6.1 – "Data Model of FLEXGRID architecture
- WP3 – "Automated flexibility aggregation energy market development and management as a service"
- WP4 – "Innovative ESS aware Business Modelling for ESPs and interaction with advanced RES & Market Forecasters
- WP5 – "Optimal Power Flow and interaction between network operators and markets"
- D6.2 – "First version of FLEXGRID S/W prototype"

Figure 1 Deliverable and work related with D6.3

1.3 Implementation Methodology

Following the approach according to D6.2 a simple methodology was defined to give the developers the opportunity to improve or redefine the APIs (already defined on the first version of the ATP) of the algorithms running on the ATP integrated by ETRA. As the API integration and the S/W development is an iterative process that continued based on the work performed during the first stages of the project and documented in D6.2.
The steps followed by the research partners to create their APIs are divided into two phases, the first one (phase A) for the definition of the API and the second one (phase B) for the implementation:

PHASE A (already done before M24)

1. Definition of the services
2. Definition of how each service will work implying the definition or adaptation of the following information:

   - **Input**: data needed to execute the service (provided by the user).
   - **Output**: data returned by the services that the specific algorithms return. For FLEXGRID it considers what is the goal of the service, the results to show and the market actors using the service.
   - **Data loaded**: Request data from the DB for the algorithm to run.
   - **Type of operation** (GET, POST, PUT…): Depending on the goal of the service.

PHASE B (Finished in M33)

3. Creation of your services (for those not created on firsts stages of the projects).
   - Transformation of the algorithm code(s) for services with json as input and output.
   - Launching the services on a local computer.
4. Upload to an online server and test the result: select between **Azure or partners own server.**

2 Use Case Scenarios

2.1 Use Case Scenarios definition

UCS 1.1: Distribution network aware flexibility market clearing via FLEXGRID ATP
In UCS 1.1, the FMO wants to efficiently clear (a set of) FlexRequests and FlexOffers for **energy maximizing social welfare** while considering network constraints. By using FLEXGRID ATP and the respective FMCT, the DSO user is able to clear the market in two different ways as an auction or in a continuous setup.

UCS 1.2: Market-based local congestion management using FLEXGRID ATP
In UCS 1.2, the FMO wants to efficiently clear (a set of) FlexRequests and FlexOffers for active **power reserve to maximize social welfare** while considering network constraints.
By using FLEXGRID ATP and the respective FMCT, the DSO user is able to clear the market in two different ways, depending on the choice. The DSO user can either clear the market in an auction, or in a continuous setup.

UCS 1.3: Market-based local voltage control in distribution network operation
In UCS 1.3, the FMO wants to efficiently clear (a set of) FlexRequests and FlexOffers for reactive (and active) power reserve that maximize social welfare while considering network constraints. In that way, UCS 1.3 includes UCS 1.2.
By using FLEXGRID ATP and the respective FMCT, the DSO user is able to clear the market in two different ways, depending on the choice. The DSO user can either clear the market in an auction, or in a continuous setup.

UCS 2.1: ESP minimizes its OPEX by optimally scheduling its FlexAssets
In the centre of the problem, we observe scheduling actions from an Energy Service Provider's (ESP) perspective. In the scope of the FLEXGRID project, ESP is considered as a profit-oriented market participant which, in the most general case, may make contractual arrangements with various types of flexibility assets (e.g. DSM, RES, storage). Furthermore, it may participate in energy and capacity wholesale markets, sell the energy on the retail market and take part in the near-real-time flexibility markets. For the purposes of UCS 2.1., the model is not network aware, so the exact location of Battery Storage Units (BSUs) is not relevant, nor are other grid constraints. The optimal scheduling algorithm is the base for the operational expenditure minimization problem. Detailed research results for this UCS are provided in D4.2 (chapter 3) and D4.3 (chapter 3).
By using FLEXGRID ATP and the respective FST service offering #1, the ESP user is able to run exhaustively online and "what-if" simulation scenarios via running an optimal scheduling algorithm to identify how to achieve minimum OPEX. Finally, the ESP user is able to visualize the results, which include the expected ESP's revenues and the optimized energy/flexibility offer curves for each market.

UCS 2.2: ESP minimizes CAPEX by making optimal investments on RES and FlexAssets
Optimal CAPEX strategy may present an important comparative advantage over the rival companies. Furthermore, optimal resource allocation may benefit the overall social welfare,

assuming that the greater competition raises market efficiency and that the greater number of players will have the opportunity to enter the market and increase the competition with each other. In that sense, a profit-seeker ESP, whose portfolio may consist of various controllable and uncontrollable assets is the focus of interest of this research problem. It uses a CAPEX minimization tool to determine the optimal investment strategy in terms of: i) size and ii) location of the different assets to fulfil its own goals and network requirements. Within the FLEXGRID project's context, optimal sizing and siting algorithm is used to ensure optimal investment strategy considering the given constraints and the objective function. In addition to the existing markets, the development of a DLFM is proposed and its influence on ESP's market behaviour alongside the conventional power markets is observed. Taking into account possible actions on all of the observed markets (DAM, RM, DLFM and BM), CAPEX minimization algorithm proposes the optimal investment strategy to participate in the energy market(s) in a preferable fashion. Meaning that for a specific one-time capital investment, operational expenses may be reduced.

UCS 2.3: ESP maximization of stacked revenues

In UCS 2.3, we consider a profit-seeker Energy Service Provider (ESP), who owns a set of Battery Storage Units (BSUs) located at various nodes of a distribution network. In order to maximize its revenues, the ESP may participate in several energy markets (i.e. day-ahead energy market, reserve market, DLFM, balancing market) and co-optimize its bidding strategy. In this way, the ESP can provide services to both the system-wide grid (TSO) and the local distribution network (DSO). Detailed research results for this UCS are provided in D4.2 (chapter 5) and D4.3 (chapter 5).

By using FLEXGRID ATP and the respective FST service offering #3, the ESP user is able to run exhaustively "what-if" simulation scenarios via running a stacked revenue maximization algorithm to identify how it can achieve maximum expected revenues. More specifically, the ESP user can provide several input parameters such as the set of markets to participate, the technical characteristics of the BSUs including their location in the distribution grid and the timeframe within which the market participation scenario takes place. Finally, the ESP user is able to visualize the results, which include the expected ESP's revenues and the optimized energy/flexibility offer curves for each market.

UCS 4.1: Manage a FlexRequest

In UCS 4.1, an independent aggregator efficiently responds to FlexRequests, received by the flexibility market or by bilateral contracts, by optimally re-scheduling (centralized manner) the flexibility assets of its portfolio. The aggregator's objective is to maximize its profits from participating in flexibility markets, while simultaneously respecting end-user preferences and constraints and avoiding imbalances. A more detailed description of this UCS can be found in the deliverables of WP3, D3.1 (Chapter 3), D3.2 (Chapter 2) and D3.3 (Chapter 2). The integration of this UCS in the FLEXGRID ATP GUI as AFAT service #1 allows the aggregator user to select between available flexibility portfolios of shiftable and adjustable assets and provide inputs such as flexibility request/s and timestamp. The results that are visualized are the aggregator's reward and cost and the deviations of the assets.

UCS 4.2: Manage a novel B2C flexibility market

In UCS 4.2, an aggregator/retailer operates an ad-hoc B2C flexibility market with its end energy prosumers by employing advanced pricing models and auction-based mechanisms.

The aggregator user runs various "what-if" simulation scenarios via running an advanced retail pricing algorithm (Behavioural Real Time Pricing – BRTP) to identify how it can recommend a new (more beneficial) FlexContract to a set of end energy prosumers. Detailed research results for this UCS are provided in D3.2 (chapter 4) and D3.3 (chapter 4).
By using FLEXGRID ATP GUI and the respective AFAT service offering #2, the aggregator user is able to provide several input parameters such as the set of consumers that participate in the proposed B2C flexibility market, a given FlexRequest, the set of FlexOffers made by the FlexAssets and the technical specifications of FlexAssets. Finally, the aggregator user is able to visualize the results, which include the: i) aggregator's revenues, ii) aggregated end users' welfare, iii) quantity of flexibility offered to the system, iv) individual end user's welfare.

UCS 4.3: Create a FlexOffer

In UCS 4.3, we propose a generic method (more information in WP3 work) for constructing aggregated FlexOffers that best represent the aggregator portfolio's actual flexibility costs, while accounting for uncertainty in future timeslots. Once trained, the machine learning algorithms can make fast decisions about the portfolio's FlexOffer in the near-real-time balancing market. The performance evaluation results show that the proposed method performs reliably towards minimizing the aggregator's imbalances (see more technical details in section 3 of D3.2).
By using FLEXGRID ATP GUI and the respective AFAT service offering #3, the aggregator user is able to make efficient FlexOffers in near-real-time balancing markets and DLFMs. The user can submit these aggregated FlexOffers from many individual FlexAssets in the ATP and this information is then available for both FMO and DSO/TSO users. The aggregator user can also visualize the expected revenues for a given timeframe given the fact that the aggregated FlexOffer will be accepted.

UCS 4.4: Forecasting services

In market price forecasting the goal was to create a reliable tool that utilizes historical data from different bidding areas and gives Day Ahead market price forecasts to ESPs/Aggregators as accurately as possible, which will help them better plan their services and optimize their profits and at the same time assess the risks.
By using FLEXGRID ATP GUI and the respectively FST service offering #4 the ESP/Aggregator user will be able to have market price forecasts for the next day for bidding areas participating in the Nord Pool's Day Ahead market. In addition, it will be able to get market price forecasts, actual market prices and the Mean Absolute Error (MAE) of earlier dates

2.2 Real Business Applicability of FLEXGRID research

Table 2 Relation with the existing regulatory framework and real business needs of the involved market stakeholders

Use Case Scenario	Partner	Scope	Programming language
UCS 1.1	DTU	Assume a P-DLFM architecture. The FMO wants to clear an energy market, i.e., DLEM, with Offers and Requests from different ESPs, while ensuring that the resulting power flows are feasible for the network.	Python

UCS 1.2	DTU	Assume a R-DLFM architecture. The FMO wants to clear an active power reserve market, i.e., DLFM, with FlexOffers from the DSO and FlexRequests from different ESPs, while ensuring that the resulting power flows are feasible for the network.	Python
UCS 1.3	DTU	Assume a R-DLFM architecture. The FMO wants to clear a reactive power reserve market, i.e., DLFM, with FlexOffers from the DSO and FlexRequests from different ESPs, while ensuring that the resulting power flows are feasible for the network.	Python
UCS 2.1	UNIZG	Assume a R-DLFM architecture. The ESP wants to minimize its OPEX by optimally scheduling its FlexAssets to respond to the FlexRequests without paying stiff penalties in the balancing market.	Python
UCS 2.2	UNIZG	Assume a R-DLFM architecture. The ESP wants to minimize its CAPEX by optimally investing in new FlexAssets in the future.	Python
UCS 2.3	ICCS	Assume a R-DLFM architecture. The ESP wants to maximize its stacked revenues by co-optimizing its participation in various markets (including DLFM or not) instead of simply participating in each one of them individually in a sequential manner.	Python
UCS 4.1	UCY	Assume a R-DLFM architecture. The aggregator wants to maximize its profits by optimally responding to FlexRequests. This translates to maximization of its revenues and minimization of the associated payments to the end users.	Python
UCS 4.2	ICCS	Assume a novel B2C flexibility market which uses a FlexRequest as input. The aggregator user wants to determine better ways (via retail pricing schemes) to operate a novel B2C flexibility market, in which the end energy prosumers compete with each other. It also wants to evaluate the impact that new FlexContracts (with its end users) would have on several KPIs such as: aggregator's revenues, aggregated end users' welfare, quantity of flexibility offered to the system, individual end user's welfare.	Python
UCS 4.3	ICCS	Assume a R-DLFM architecture. The aggregator wants to determine/create a FlexOffer that best represents the current status of its portfolio and submits it to the FLEXGRID ATP. This FlexOffer may be used either in the: i) TSO's reserve market (cf. "no-DLFM" architecture), or ii) proposed DLFM market operated by the FMO to solve DN-level problems.	Python

UCS 4.4 - Price	UCY	The ESP/aggregator wants to forecast the market prices (only applicable auction-based markets) in a day-ahead and intra-day context. This service is offered on top of all the other FST and AFAT services described above.	Python

3 API and DB Integration

3.1 Introduction

APIs are programming interfaces that allow applications to exchange data and functionality in an easy and secure way, thus simplifying software development and innovation.
In the case of the project, the function of the implementation of these APIs will be to allow interaction and data exchange throughout the software platform and between the different modules/algorithms developed in the different work packages. They will also have the function of acting as connectors between the different actors involved in the management and transmission of electricity ensuring the optimization of this process.
With the possible development of many more advanced energy meta-services in the future, the APIs developed in the project will be rich and flexible enough to allow the integration of future platforms and avoid vendor lock-in.

An API is not a database. It is an access point to an application that can access a database. Therefore, since the project is going to develop APIs, it is absolutely necessary to have a database. A database is a system that collects and stores structured information, or data, allowing the access and manipulation of this data and thus facilitating data management.

In addition, having a database will provide numerous advantages when it comes to carrying out the project's objectives. Among these advantages we find; i) the independence between programs and data, that is to say, to separate the metadata of the applications that use data allowing the transfer of the data without influencing the programs that are processing the information, ii) the minimum redundancy of the data, allowing this redundancy when it is beneficial, since the redundancies of data are desirable in some cases and increase the performance of the database, iii)the improvement in the interchange of data, each group or person has specialized views of the data.

3.2 FLEXGRID S/W architecture

The ATP is an ICT platform through which an energy stakeholder can optimally design a marketplace according to its needs and automatically operate it in B2B or B2C mode.
To achieve this ATP is responsible for supporting the optimal and automated planning and operation of markets as required by modern stakeholders to interact with each other to deliver competitive ES through advanced flexibility trading.

The ATP will be responsible for the dynamic and conscious management of the power grid through the control of the flexibility assets, aiming to match FlexDemand with FlexSupply, thus clearing the ad-hoc flexibility market, and to define the FlexPrice based on the FlexAssets, as it will be the platform where the FlexAssets trading between FlexSuppliers and FlexBuyers will take place. In full operation of the market, the FLEXGRID ATP will be independent of any market participant or at least no market party or network owner will be a principal owner of the FLEXGRID market.

The ATP will integrate all the algorithms and graphical interfaces developed in the different UCS, carrying out a first analysis of the data model information.

The ATP will also be in charge of collecting the historical data obtained in each of the different periods of the project, of acting as a link between the end users and the algorithms developed in the UCS and of allowing the visualization of the results obtained by the users.

In addition, the ATP will perform functions such the ones described on the different UCS (section 2.1) to allow the aggregator to maximize its final benefits without neglecting the needs of the rest of the energy market agents.

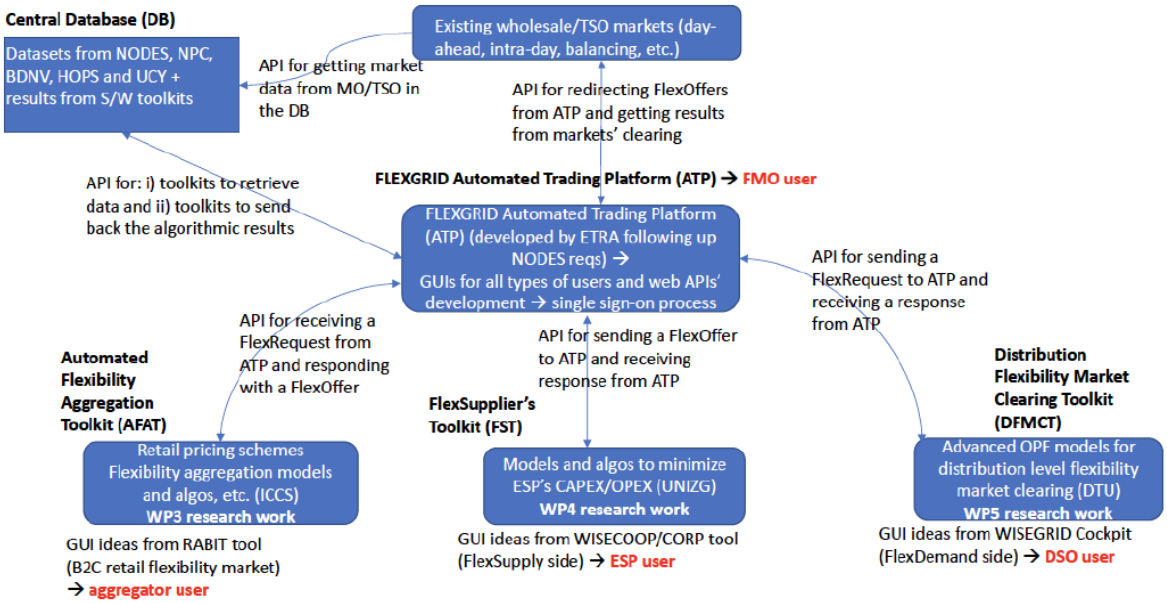


Figure 2 ATP application schema

The approached follow for the design of the ATP and the APIs allows the users to interact between them by exchanging web links and thus be able to facilitate new business cases. Both D7.3 and D8.3 will include more details of the interaction between users tested on the pilots and the novelties of the ATP for new business models.

3.3 Use Cases Scenarios integration

For the integration of the different use case scenario on the platform the methodology defined in section 1.3 was followed by the objective to obtain an API to make possible the interaction between the different modules and algorithms with the ATP interface as shown on Figure 2.

3.3.1 Creation of the API

Although each UCS has its specific characteristics in terms of system integration and creation of the API, they also share some common information and operation. For the API, they were created using the OpenAPI 3.0 specification. The editor.swagger.io online editing tool was used for editing the initial version of the API specification.

The API consists of two components:

1. The interface to the algorithm execution module: This component was created using the swagger-codegen tool, which allows automatic generation of a skeleton of the application, using the OpenAPI definition file as input. Two adapters were added, one for downloading the needed data from the local database, and one for invoking the algorithm itself.

2. The interface to the local database: This module was developed using the python eve tool, which allows the automatic creation of a mongo dB database, the API endpoints, and the OpenAPI definition, based on configuration files developed in python. Furthermore, the oauthlib library was used to provide an authentication service for the services of the use cases.

3.3.2 Integration of the algorithm on the server

The optimization algorithm itself was developed using python (see Table **2**), in a separate repository, that was added to the main API repository as a git submodule. The option of servers to be used for the integration were Azure or partners own server. This decision makes the process of integration different but with the same result, the creation of a swagger to make possible the iteration between the ATP and algorithms.

3.3.3 Use Case Scenarios specification

Following specific characteristics of each UCS integration is defined

UCS 1.1, 1.2 and 1.3 follow the same process in order to create and integrate the algorithms on the ATP interface:

Creation of the API

The API was created using the Azure Webapp services and it consists of two main components:

1. The interface to the algorithm execution module: This component was created using Azure CLI, Python 3.8 and python's Flask package.
2. The interface to the local database: This module was not developed.

Integration of the algorithm on the server

For the integration and iterative process was followed as the optimization algorithm itself was developed using python, in a separate repository, that was added to the main API repository. Some adaptations were needed to make the exchange of the information possible allowing the ATP user to run the specific algorithms of the UCS 1.1, 1.2 and 1.3. To see the OpenAPI document it should be include on the swagger explore bar on the following link

https://api.demo.etra-id.com/ the raw yaml definition located at https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS1.X.yaml.

UCS 2.1: The methodology that was used for the development of use case scenario 2.1 is similar to the one described in UCS 4.2 (section 2.1), so it is not repeated here for the purpose of brevity. The difference here is the definition of the OpenAPI document, and the implementation of the algorithm. In addition, since this algorithm is executed quickly, there is no background job, but the results are returned in the response body of the API call. The OpenAPI document can be seen at swagger format by including on the swagger's explore bar on the following link https://api.demo.etra-id.com/ the raw yaml definition located at https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS2.1.yaml

UCS 2.2: The methodology that was used for the development of use case scenario 2.2 is similar to the one described in UCS 4.2 and UCS 2.1, so it is not repeated here for the purpose of brevity. The difference here is the definition of the OpenAPI document, and the implementation of the algorithm. In addition, since this algorithm is executed quickly, there is no background job, but the results are returned in the response body of the API call. The OpenAPI document can be seen at swagger format by including on the swagger's explore bar on the following link https://api.demo.etra-id.com/ the raw yaml definition located at https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS2.2.yaml

UCS 2.3: The methodology that was used for the development of use case scenario 2.3 is similar to the one described on section 3.3.1 and 3.3.2. The difference here is the definition of the OpenAPI document, and the implementation of the algorithm. In addition, since this algorithm is executed quickly, there is no background job, but the results are returned in the response body of the API call. The OpenAPI document for this use case scenario is at https://stacked-revenues-api.flexgrid-project.eu/swagger/ (Raw yaml definition at https://stacked-revenues-api.flexgrid-project.eu/swagger/stacked-revenues.yml)

UCS 4.1: To create the API for UCS4.1, the following steps were followed:

- Definition of all services related to the UCS and documented in a text file
- Input/Output and Type of operation defined for all services
- Modification of python code to create the services (use of python library Flask)
- Upload services to UCY server
- Test Execution of services

The algorithm is integrated within the main service. The results are returned after the call of the service. As the service runs on demand, the results are not stored and needed parameters are included in local storage. No interaction is required with the central DB. The OpenAPI document can be seen at swagger format by including on the swagger's explore bar on the

following link https://api.demo.etra-id.com/ the raw yaml definition located at https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS4.1.yaml

UCS 4.2: The methodology that was used for the development of use case scenario 4.2 is the one described in section 3.3.1 and 3.3.2 and UCS 2.3. The difference here is that since the execution of the algorithms may take up to several minutes, the execution was invoked as a background task using the celery tool. When a new simulation is submitted for execution, the API returns a JOB_ID, which is a unique identifier for the specific algorithm execution. When the execution of the JOB is complete, the API posts back to a call back URL, to notify the client that the results are available. The client may use the JOB_ID to query the API server regarding the status of the execution, as well to get the algorithm results. The OpenAPI definition for the local database is available at https://db.flexgrid-project.eu/swagger/ (Raw JSON specification at https://db.flexgrid-project.eu/api-docs). The OpenAPI definition of the API endpoint is the at https://pricing-api.flexgrid-project.eu/swagger/ (Raw yaml definition at https://pricing-api.flexgrid-project.eu/swagger/pricing.yml

UCS 4.3: The methodology that was used for the development of use case scenario 4.3 is similar to the one described in section 3.3.1 and 3.3.2 and UCS 2.3. The difference here is the definition of the OpenAPI document, and the implementation of the algorithm. In addition, since this algorithm is executed quickly, there is not background job, but the results are returned in the response body of the API call. https://flex-offers-api.flexgrid-project.eu/swagger/ (Raw yaml definition at https://flex-offers-api.flexgrid-project.eu/swagger/flex_offers.yml)

UCS 4.4: To create the API for UCS4.4 – Market Price Forecasting, the following steps were followed:

- Definition of all services related to the Market Price Forecasting and documented in an MD file
- Inputs/Outputs and Type of operation defined for all services
- Modification of python code to create the services and write to the database server of UCY
- The API was developed using PHP code and each user will be provided with a private key to acquire the datasets with /GET request
- The user will provide the date and location for the Market price data (actual and forecasts) and the API will return the requested datasets. If the data for the specific dates are not available, the API will return an empty JSON.
- Data availability will be for 1 year period
- Upload services to UCY server
- Test Execution of services → TBD

The algorithm is developed as a service on the webserver that pushes the data sets to the database server. A separate API code was implemented in order to pull the datasets from the

database server and generate the JSON file. The results are returned after the call of the service. The service is asynchronous therefore, the availability of the data sets is dependable, but the API execution is not affected by the algorithm execution. No interaction is required with the central DB. OpenAPI document it should be include on the swagger explore bar on the following link https://api.demo.etra-id.com/ the raw yaml definition located at**Error! Hyperlink reference not valid.** https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS4.4_PricesForecasting.yaml

3.4 Central data base integration

The Central Database is a common local storage where algorithms in Flexgrid share information. Besides, it is used by ATP GUI to store results of executions when an algorithm is launched from it. The database is located at ETRA premisses, but it is accessible to partners through identified access, so data is protected.

During task T6.2, a survey about needs and requirements was distributed among Flexgrid partners, in order to determinate what kind of database was better. This survey included some question about the use of their data in the project, such as: Frequency of storage of new data, volume of data, etc. After the revision of partners' answers, we chose MongoDB as database engine, as it suited perfectly well in Flexgrid needs.

MongoDB is a database engine specially created for resiliency, scalability, privacy and data securing. It is a non-SQL database, that is, it does not follow the traditional schema of tables and fixed-formatted registers, but it organizes the information in lists of elements called collections. Each element in a collection can have its own structure, being formatted as a JSON. This flexibility allows to deal with very complex problems, as Flexgrid is, since data does need to follow a fixed structure. For instance, each algorithm generates results in a format that is not the same as other algorithms results.

About the interaction between the algorithms and the Central Database, developers have included in their code a library in python (pymongo[1]), which allows that communication. Here an example of code using this library, where the latest stored FlexOffer is loaded:

```
from pymongo import MongoClient

# Connection to MongoDB
client = MongoClient("centralDatabase URL")
db=client.admin

# Get the collection where flex offers are stored
flexOffers = db['FlexOffer']

# Find latest flex offer inserted
offer = flexOffers.find({}).sort("creationDate", pymongo.DESCENDING)[0]
```

[1] https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb

```
# Now variable offer contains the flex offer loaded and it can be used in
# the algorithm as usual.
```

For the use of collections in the project, partners filled a document indicating the list of data needed, so for each set of information to be used, a collection was prepared in the database. Some of these collections are Consumption, CurtaibleLoad, FlexRequest, FlexOffer, PriceBalancing, etc.

4 ATP Graphical User Interface

4.1 Introduction

The ATP is the main interface to the electricity market interaction with the algorithms developed on FLEXGRID.
First of all, it would be possible to log in the application with the specific user. Each user is able to manage different configuration tabs.

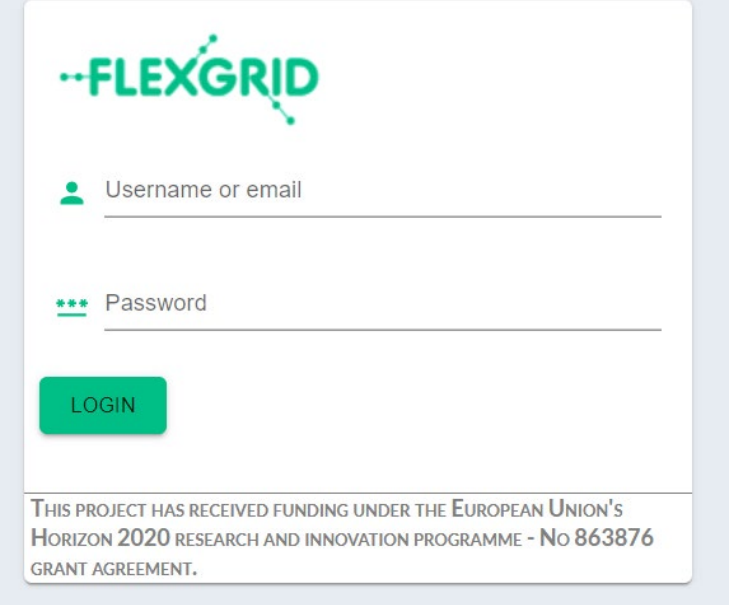


Figure 3 Login interface for ATP

After the log in, each user is able to see the specific UCS assigned to their role.

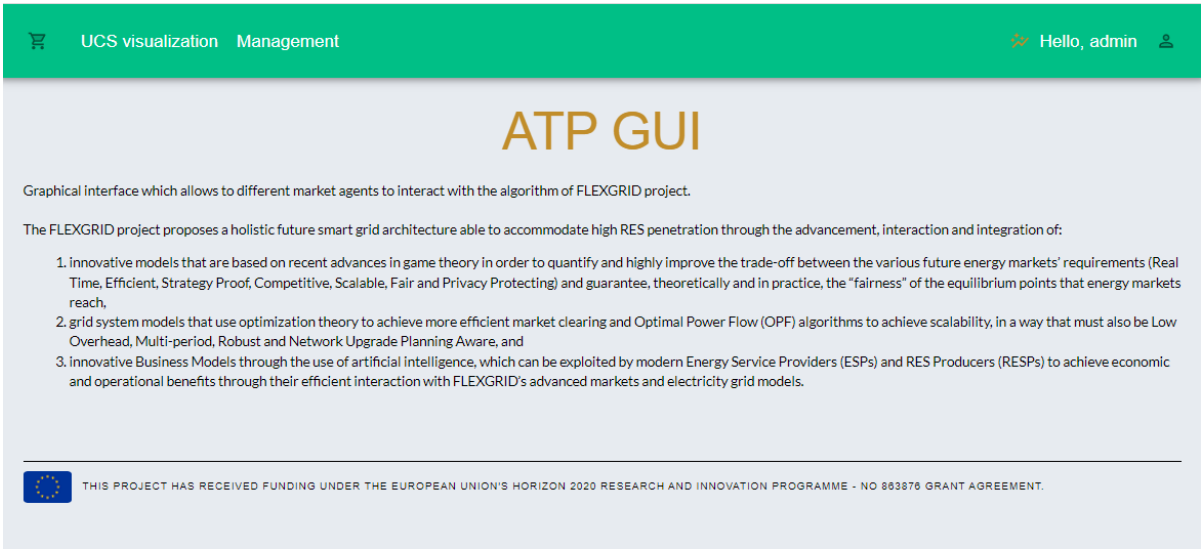


Figure 4 ATP frontend

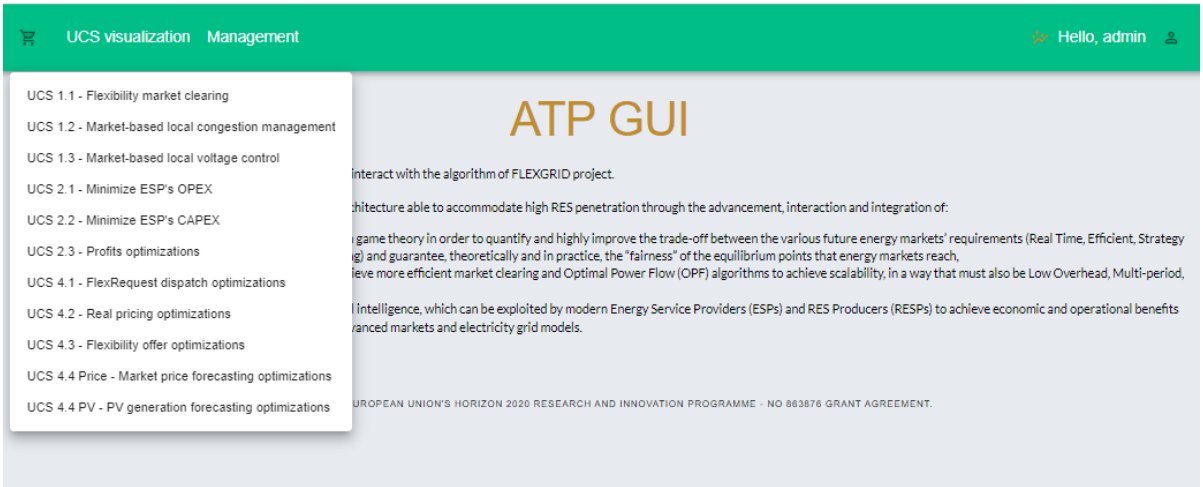


Figure 5 ATP frontend - Use case scenarios

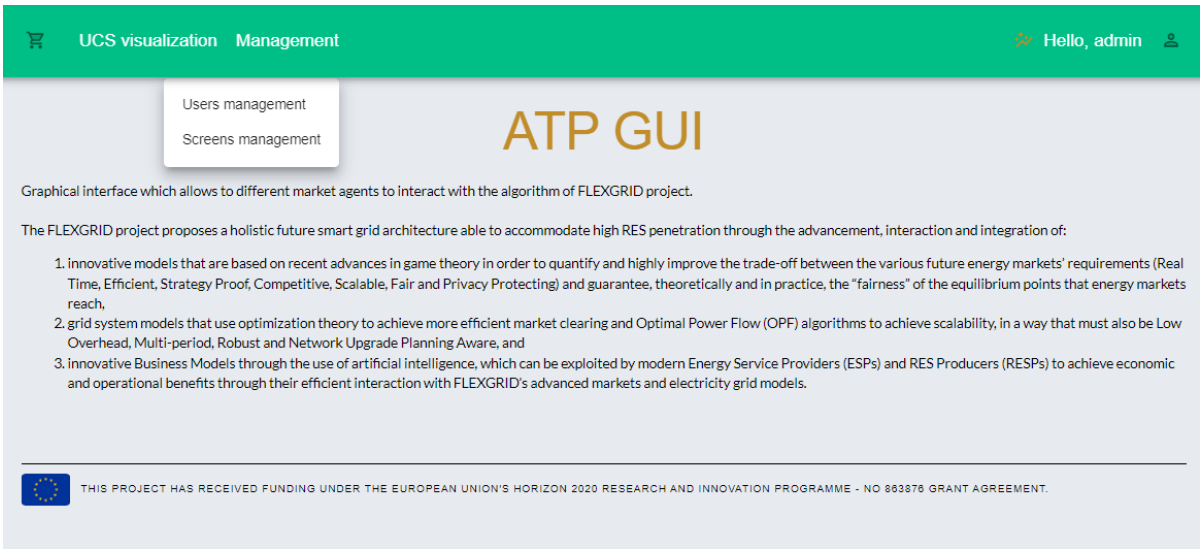


Figure 6 ATP frontend - Management tools

The owner of the whole platform (i.e administrative user) is able to manage the user permissions and the tabs available for each one in an easy way by clicking on "User management". It is also possible to add new users, edit or delete.

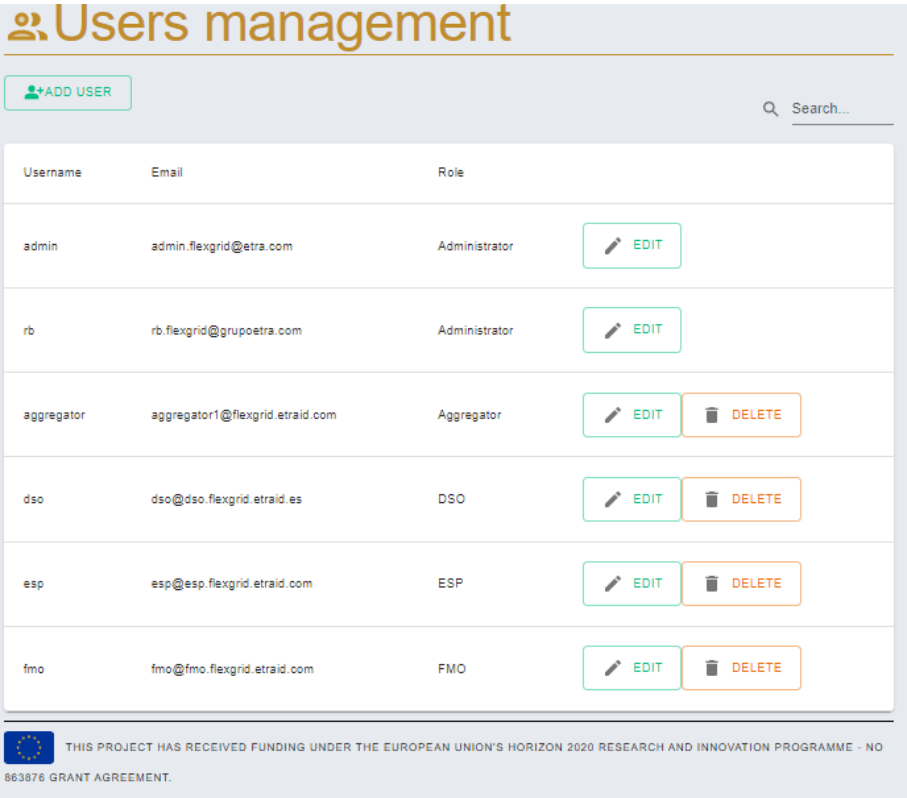


Figure 7 ATP users management and configuration

Additionally, on “Screen management” the administrator of the platform can give access to the “historical”, “configuration” and “results” tabs per UCS.

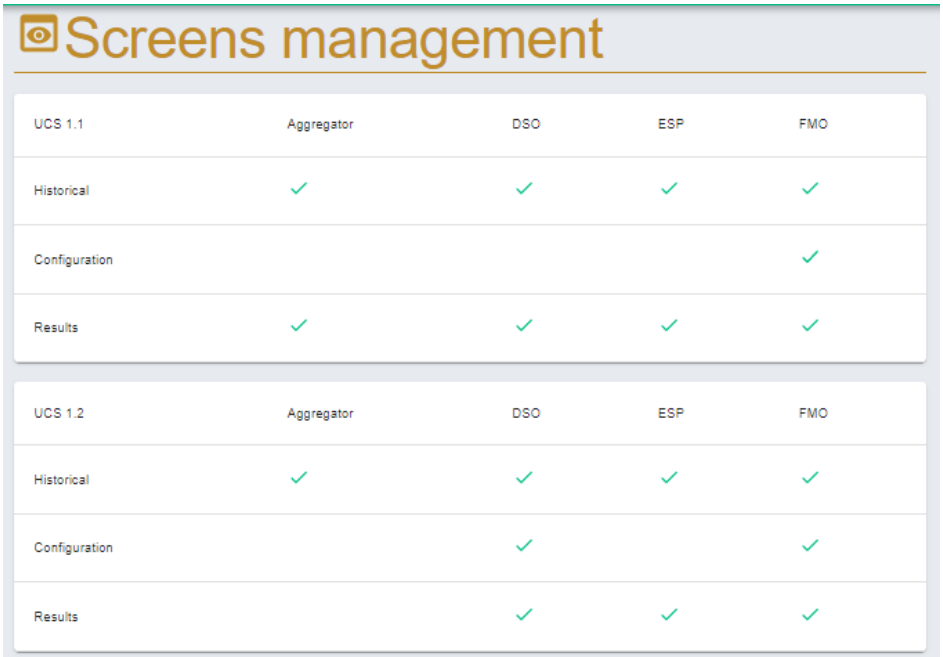


Figure 8 ATP screens premises configuration for users

4.2 Functionalities general overview

Once the user logs in the ATP (as described in section 4.1) they can interact with the different views of each UCS. By clicking on each UCS and depending on the access permissions/rights

for the user they will be able to navigate through the three main categories that were defined in D6.2:

- Configuration: It is the link between the application user and the algorithms. At the configuration view it is possible to indicate all the inputs needed to launch the UCS specific algorithms and optimization considering the market, the grid, the FlexAssests, the end users and more.
- Results: The "Results" view allows the user to see the results of the operations made according to his role registered on the ATP (e.g., revenues from the different markets available and the energy use in the market, relevant output from the algorithm calculations, etc…)
- Historical: For the market actors (DSO, TSO, FMO, aggregator…) allowed to use the ATP functionalities in this view it is possible to see all the historical data available for previous operations with a series of useful information to better understand each operation carried out in a different period.

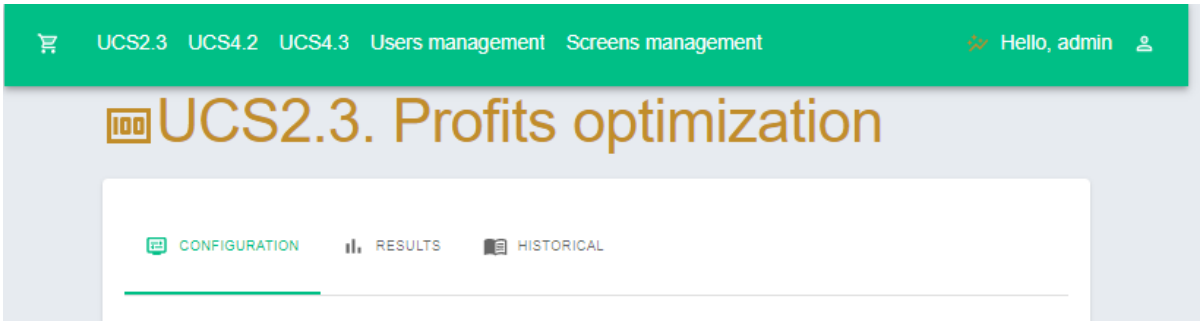


Figure 9 UCS 2.3 example for configuration, results, and historical tabs of ATP

The available functionalities for the ATP modules and UCS are the same as reported in D6.2 where the GUIs and functionalities were defined on the first stages of the project. The following table brings together all the functions that the different actors have to work with the FLEXGRID ATP, and how the different users' roles are linked with the functionalities' premises given by the administrator of the platform (ETRA) by using the screens management tab.

Table 3 GUIs functionalities

Module	UCS	Functionality	User			
			Aggr	DSO	ESP	FMO
FMCT	UCS1.1	Flexibility market clearing historical view	✓	✓	✓	✓
		Flexibility market clearing configuration	✓ 0	✗	✗	✓
		Flexibility market clearing results	✓ 2	✓	✓	✓
	UCS1.2	Market-based local congestion management historical view	✓	✓	✓	✓
		Market-based local congestion management configuration	✗	✓	✗	✓
		Market-based local congestion management results	✗	✓	✓	✓
	UCS1.3	Market-based local voltage control historical view	✗	✓	✓	✓
		Market-based local voltage control configuration	✗	✓	✓	✓
		Market-based local voltage control results	✗	✓	✓	✓
FST	UCS2.1	OPEX optimizations historical view (with price)	✗	✗	✓	✗
		OPEX optimizations historical view (without price)	✗	✓	✗	✓
		OPEX optimization configuration	✗	✗	✓	✗
		OPEX optimization results (with price)	✗	✗	✓	✗
		OPEX optimization results (without price)	✗	✓	✗	✓
	UCS2.2	CAPEX optimizations historical view	✗	✗	✓	✗
		CAPEX optimization configuration	✗	✗	✓	✗
		CAPEX optimization results	✗	✗	✓	✗
	UCS2.3	Profits optimizations historical view (with price)	✗	✗	✓	✗
		Profits optimizations historical view (without price)	✗	✓	✗	✓
		Profits optimizations configuration	✗	✗	✓	✗
		Profits optimization results (with price)	✗	✗	✓	✗
		Profits optimization results (without price)	✗	✓	✗	✓
AFAT	UCS4.1	FlexRequest dispatch optimizations historical view	✓	✓	✗	✓
		FlexRequest dispatch optimization configuration	✓	✗	✗	✗
		FlexRequest dispatch optimization results	✓	✓	✗	✓
	UCS4.2	Real pricing optimization historical view	✓	✗	✗	✗
		Real pricing optimization configuration	✓	✗	✗	✗
		Real pricing optimization results	✓	✗	✗	✗
	UCS4.3	Flexibility offer optimizations historical view (with revenues)	✓	✗	✗	✗
		Flexibility offer optimizations historical view (without revenues)	✗	✓	✗	✓
		Flexibility offer optimization configuration	✓	✗	✗	✗
		Flexibility offer optimization results (with revenues)	✓	✗	✗	✗
		Flexibility offer optimization results (without revenues)	✗	✓	✗	✓
	UCS4.4 (price)	Market price forecasting historical view	✓	✗	✓	✗
		Market price forecasting configuration	✓	✗	✓	✗
		Market price forecasting results	✓	✗	✓	✗

5 Results validation

The following sections show how the results of the integrated algorithms are represented on the ATP and how the user willing to use the ATP can interact with the configuration and historical results. This section the general overview of the GUIs for the application developed on the contents of the project.
All the results and simulations here represented where validate with the corresponding results for the algorithms developed on the relevant WP (WP3, WP4 and WP5). It is essential to indicate that all results are based on the current available data and the assumption of having access to proper data for further developments and accurate results.

5.1 FST GUIs and Results

5.1.1 UCS 2.1 - Minimize ESP's OPEX

After the ESP user has successfully logged in the FLEXGRID ATP, then s/he can select UCS 2.1 service ("Minimize ESP's OPEX") and the following web page will appear:

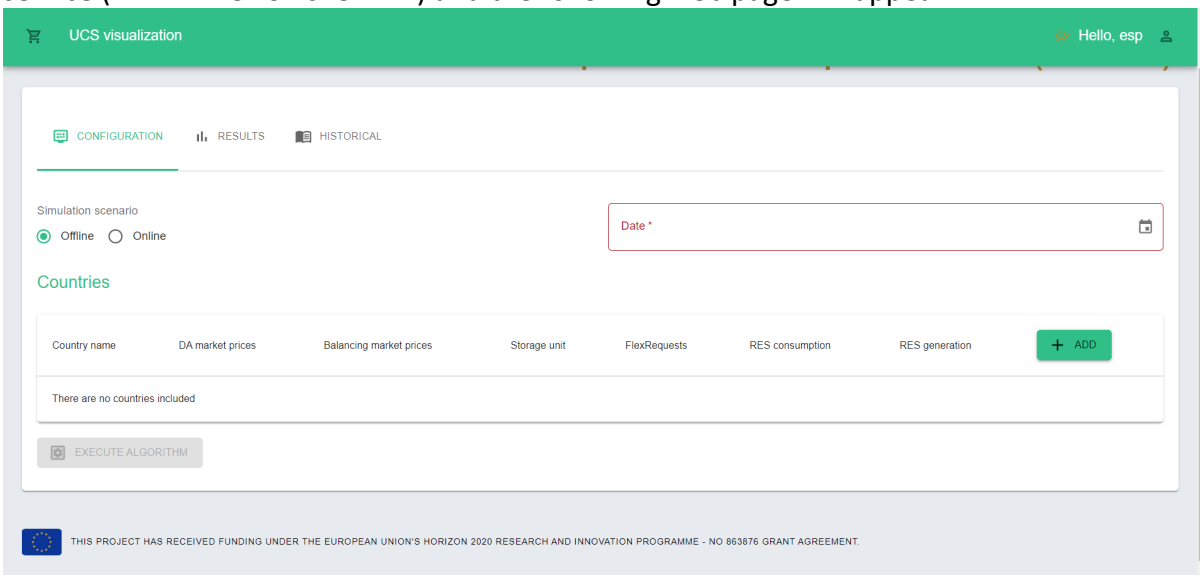


Figure 10 The ESP user selects UCS 2.1 service and fills in the input parameters via "Configuration" tab

Now, the ESP user can select the "Configuration" tab to fill in the input parameters. Firstly, the user is able to choose between the offline and online simulation scenario. Then s/he sets the date and is able to add all other required input data, as shown in the figure above, by pressing the "ADD" button.

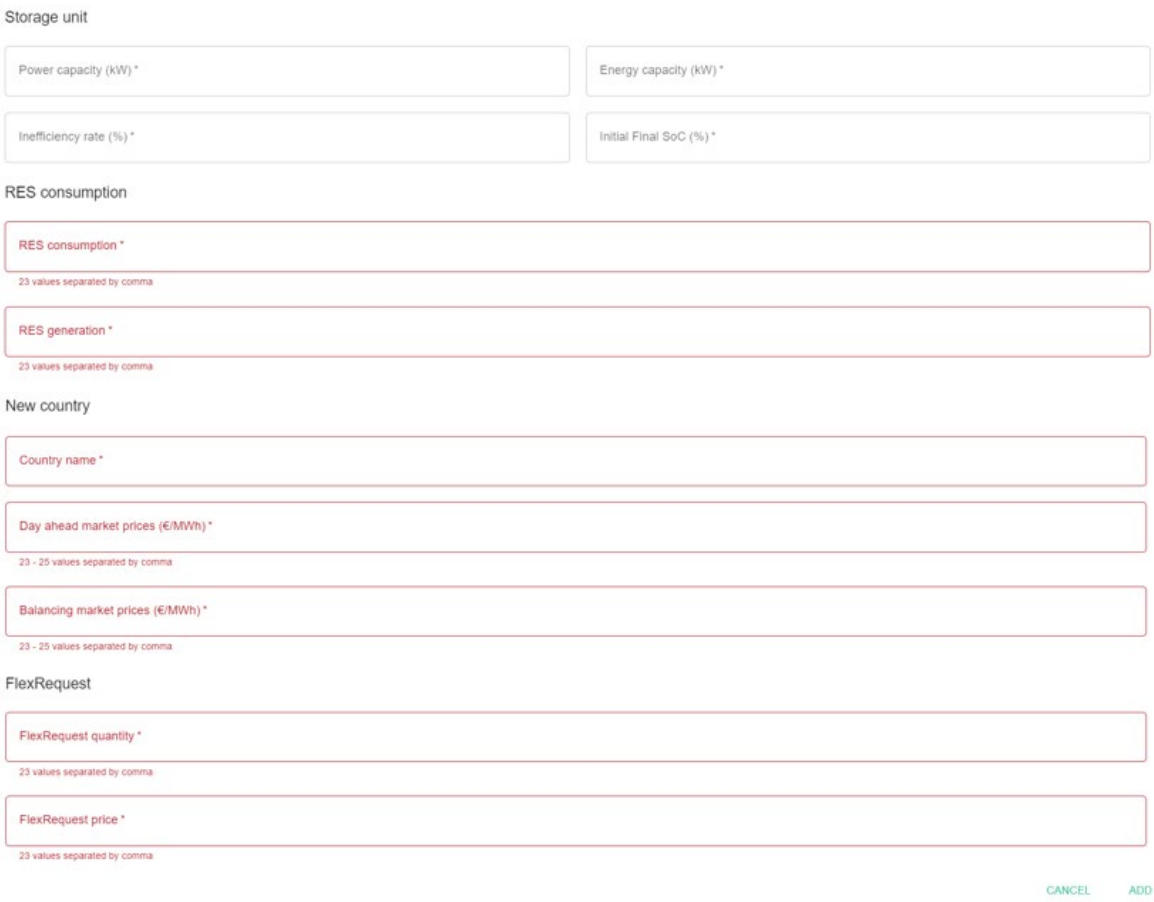


Figure 11 The user sees this window after pressing the “ADD” button on the right corner of the screen

Required inputs include country name, day-ahead prices, balancing market prices, FlexRequest information, Storage unit information (including Power capacity, Energy capacity, Inefficiency rate and Initial/Final SoC), and, finally, consumption and production data. For instance, a storage unit has been defined which has 100 KW power capacity, 400 KWh energy capacity, inefficiency rate 95%, initial/final State of Charge (SoC) 50%.

Then, the ESP user selects the “Optimize” button and waits a few seconds until the results are fetched back from the FST server. “Results” tab offers graphical and numerical representation of the results.

The following two figures show how optimization results are visualized. Figure 12 presents battery storage unit activity on various markets, whereas Figure 13 presents how battery storage unit state of energy is changing throughout the observed day.

Figure 12 The ESP user can visualize battery storage unit participation in various markets

In Figure 12, the ESP user may clearly observe in what manner is the observed battery storage unit participating in various markets. For each of the observed markets are two subgraphs dedicated, one for the battery storage unit discharging activities in the respective market, and one for the battery storage unit discharging activities.

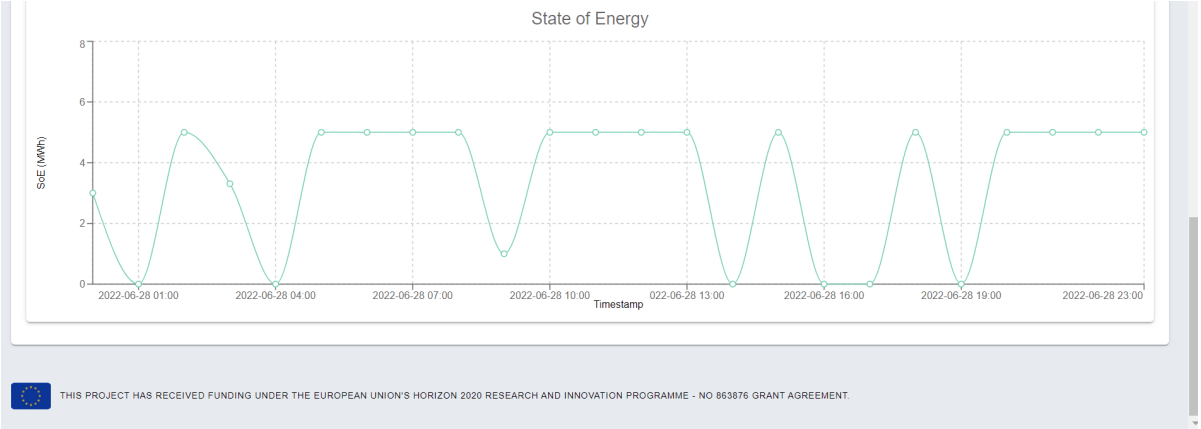


Figure 13 The ESP user can visualize battery storage unit state of energy throughout the day

Figure 13 nicely shows how easy it is for the ESP user to observe state of energy and, in fact, changes of the state of energy, for the observed battery storage unit. The state of energy is shown for each of the total number of the observed hours.

Finally, the ESP user can see all the historical results from all the past simulation runs via the "Historical" tab as shown in the figure below. Thus, s/he is able to compare them to get better overview about optimal strategies for different circumstances. Moreover, the FMO and DSO users are able to visualize all the bidding history of the ESP and thus be able to analyze each ESP's business behavior or even cooperate with the ESP towards organizing local flexibility markets and thus realize win-win business cases.

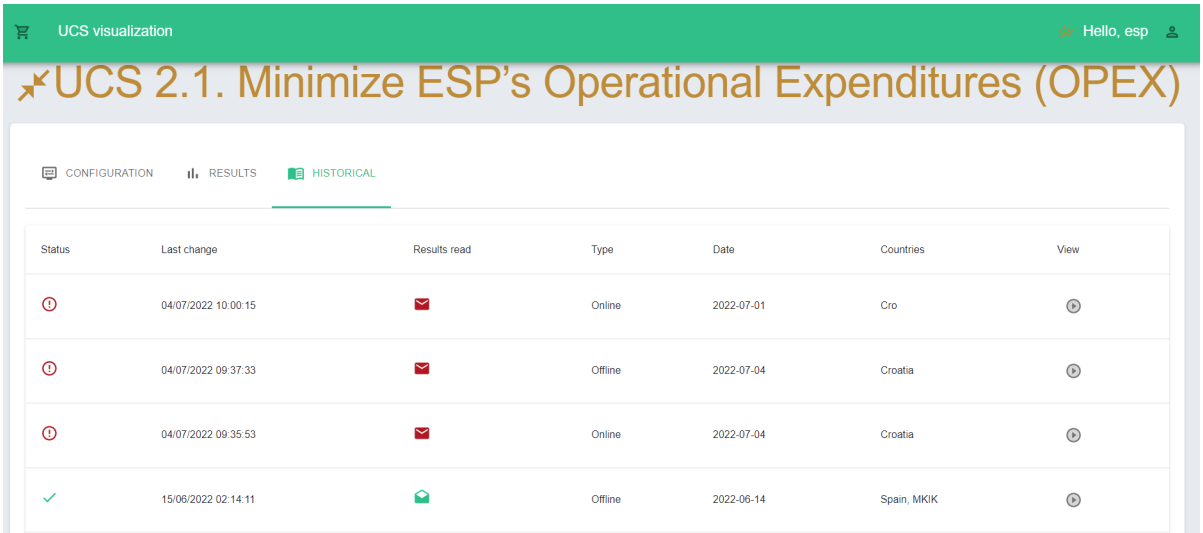


Figure 14 The ESP user can visualize and compare all past results via "Historical" tab

5.1.2 UCS 2.2 - Minimize ESP's CAPEX

After the ESP user has successfully logged in the FLEXGRID ATP, then s/he can select UCS 2.2 service ("Minimize ESP's CAPEX") and the following web page will appear:

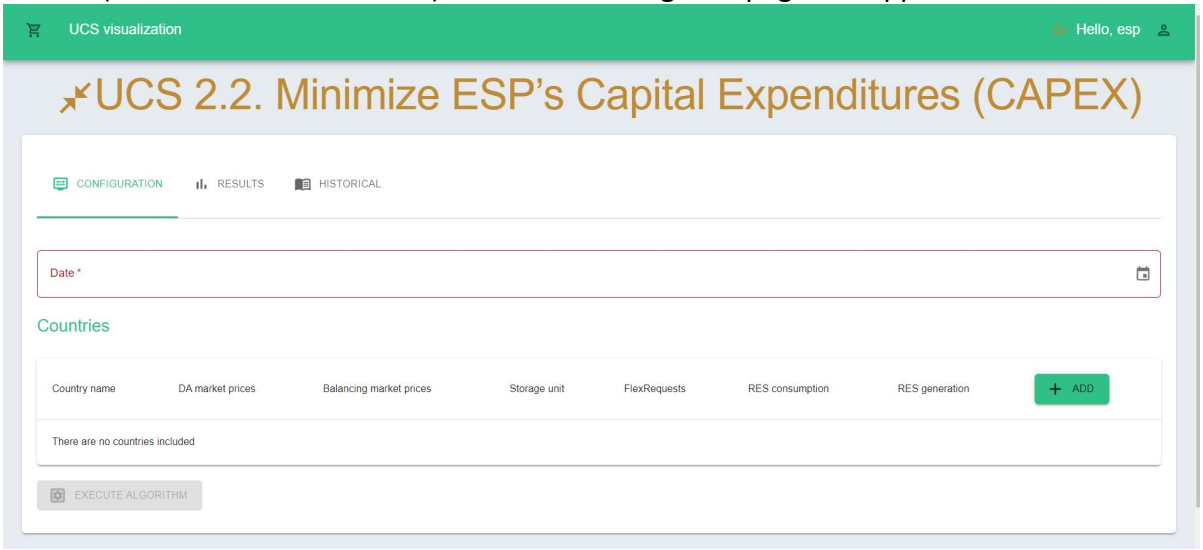


Figure 15 The ESP user selects UCS 2.2 service and fills in the input parameters via "Configuration" tab

Now, the ESP user can select the "Configuration" tab to fill in the input parameters. Firstly, the user is able to choose a the date. Then s/he is able to add all other required data, as shown in the Figure 16 and Figure 17, by pressing the "ADD" button.

Figure 16 The user sees this window after pressing the “ADD” button

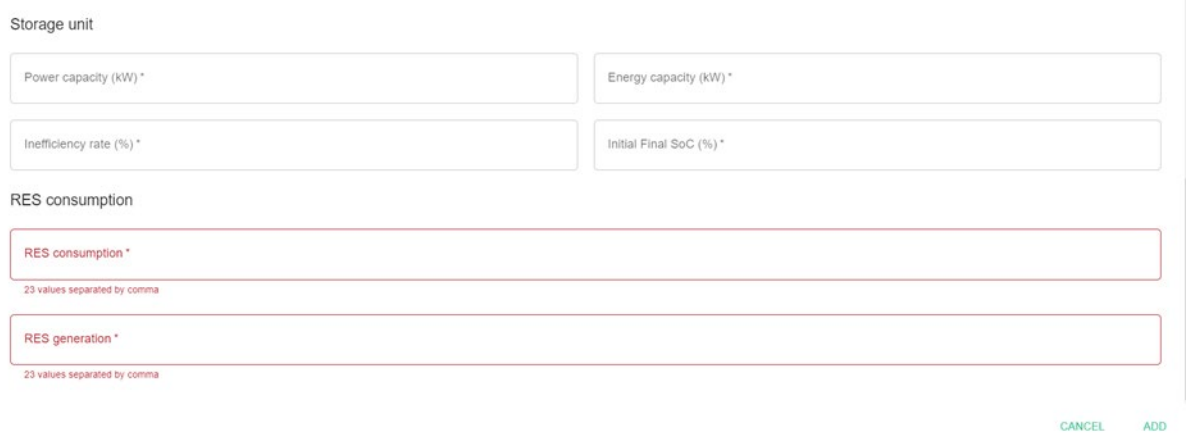


Figure 17 The user sees this window after pressing the “ADD” button

Required inputs include country name, day-ahead prices, balancing market prices, FlexRequest information, Storage unit information (including Power capacity, Energy capacity, Inefficiency rate and Initial/Final SoC), and, finally, consumption and production data. For instance, a storage unit has been defined which has 100 KW power capacity, 400 KWh energy capacity, inefficiency rate 95%, initial/final State of Charge (SoC) 50%.

Then, the ESP user selects the “Optimize” button and waits a few seconds until the results are fetched back from the FST server. “Results” tab offers graphical and numerical representation of the results.

Following figure shows how optimization results are presented in a visual manner. Figure 18 presents how battery storage unit state of energy is changing throughout the observed day.

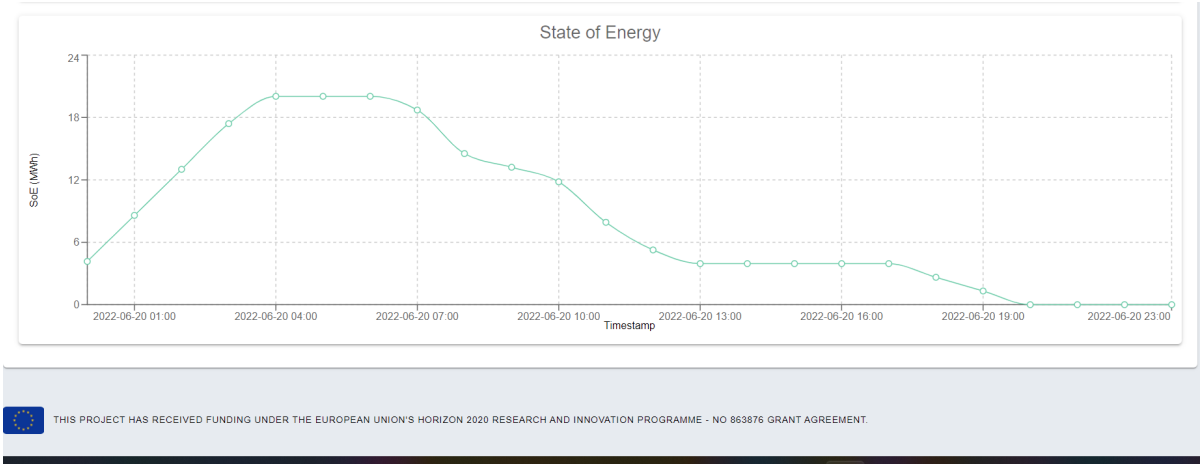


Figure 18 The ESP user can visualize battery storage unit state of energy throughout the day

In Figure 18, the ESP user may clearly observe changes of the state of energy for the observed battery storage unit. The state of energy is shown for each of the total number of the observed hours.

Finally, the ESP user can see all the historical results from all the past simulation runs via the "Historical" tab as shown in the figure below. Thus, s/he is able to compare them to get a better overview about optimal strategies for different circumstances.

Figure 19 The ESP user can visualize and compare all past results via "Historical" tab

5.1.3 UCS 2.3 - Stacked revenues maximization

After the ESP user has successfully logged in the FLEXGRID ATP, then s/he can select UCS 2.3 service ("Stacked revenues maximization") and the following web page will appear:

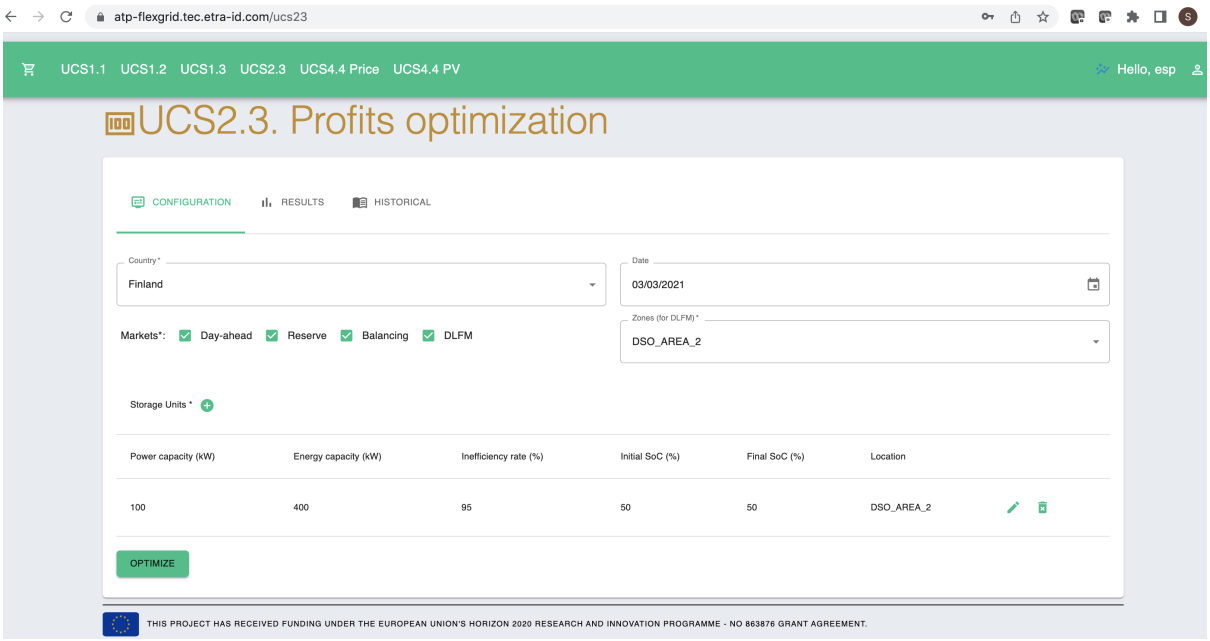


Figure 20 The ESP user selects UCS 2.3 service and fills in the input parameters via “Configuration” tab

Now, the ESP user can select the “Configuration” tab to fill in the input parameters. For example, s/he sets the country, date, specific markets to participate, the DSO area and the technical specifications of a storage unit. For instance, a storage unit has been defined which has 100 KW power capacity, 400 kWh energy capacity, inefficiency rate 95%, initial/final State of Charge (SoC) 50% and is located in DSO area 2.

Then, the ESP user selects the “Optimize” button and waits a few seconds until the results are fetched back from the FST server. As shown in the following two figures, the ESP user can see: i) which are the expected offers for participating in all 4 markets, and ii) which are the expected revenues from participating in each one of the 4 markets. For example, the balancing market (BM) offer down for 00:00 hourly timeslot is 23 kW and the BM offer up is 0 KW. On the other hand, for 18:00 hourly timeslot, the BM offer down is 0 kW and BM offer up is 200 kW. Regarding the expected revenues, these are 20.57 euros from Balancing Market (BM), 31.86 euros from Day-Ahead Market (DAM), 54.86 euros from Flexibility Market (FM) and 16.04 euros from Reserve Market (RM). So, the total expected revenues are 122.83 euros.

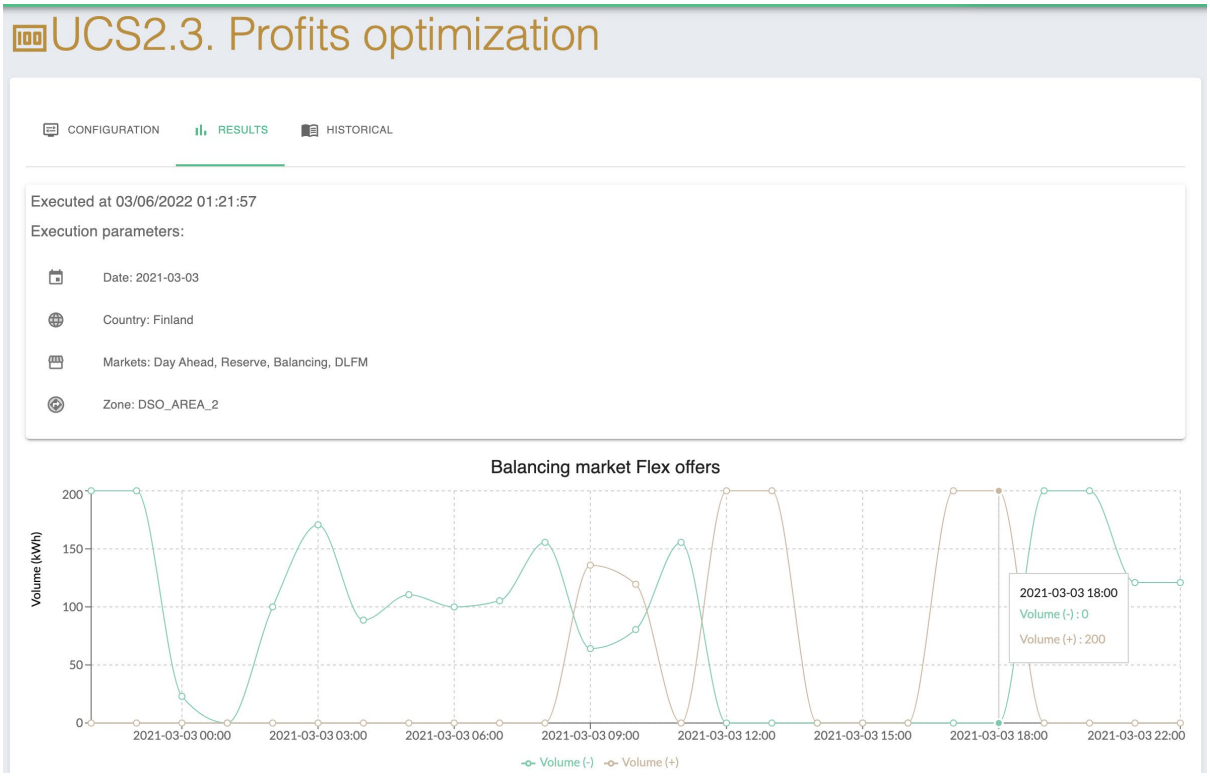


Figure 21: The ESP user visualizes the bid curves to all the markets via the "Results" tab

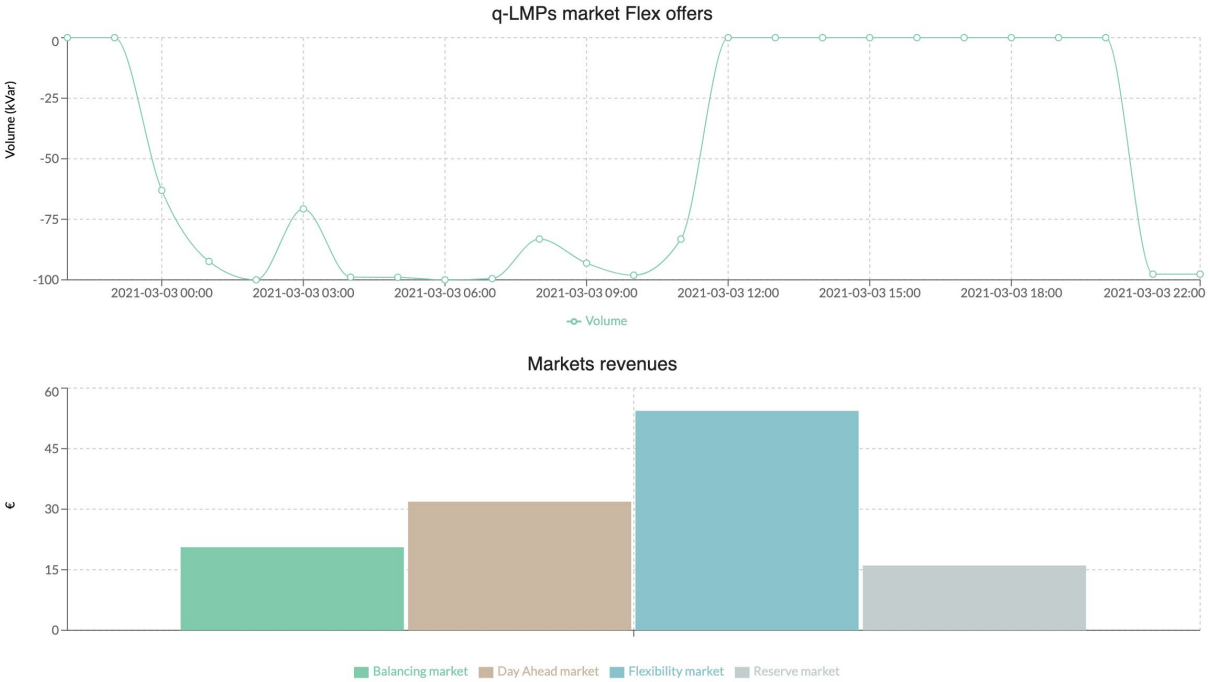


Figure 22: The ESP user visualizes the expected revenues for all markets via the "Results" tab (case 0)

Now, let's assume case 1 in which the same storage unit is selected, and it participates only in DAM and RM. In other words, this means that this FlexAsset can provide flexibility services only to the TSO. In Figure 23, in the upper part, one can see the FlexOffers for the RM (both for up and down directions) and the offers for the DAM (i.e. positive values infer up direction and negative values infer down direction). In Figure 24, the expected revenues from DAM are 10.34 euros and 45.20 euros from RM (i.e. the algorithm prefers to participate in RM due to

higher prices). The total expected revenues are 55.54 euros, which are quite less from the previous case and this is rational because the ESP has now fewer degrees of freedom towards selecting its optimal bidding policy.

Figure 23: Screenshot from case 1 results (ESP participates only in RM and DAM)

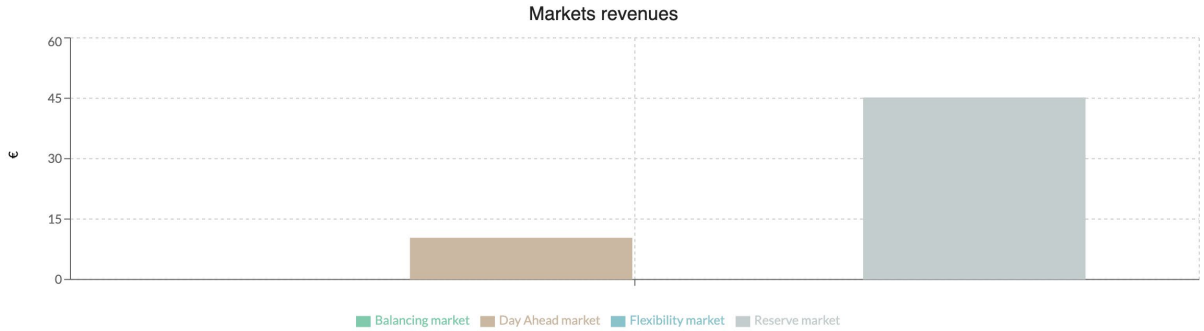


Figure 24: ESP's expected revenues for participating in DAM and RM only (services only to TSO)

Let us now assume case 2, in which the ESP participates in DAM, FM and BM providing thus flexibility services only to the local DSO. The expected revenues are shown in Figure 25 and are 111.03 euros in total. In Figure 26, we run a similar case, with the only difference that we now assume that the storage unit resides in DSO area 1, which does not encounter any congestion or voltage control problem. As a result, FM revenues are now equal to 0. Finally, if we assume that the storage unit resides in DSO area 3, the respective revenues are: 80.96 euros for BM, -20.78 euros for DAM and 15.32 euros for FM. This happens because the DSO area 3 is less congested and thus the FM prices are lower than in DSO area 2.

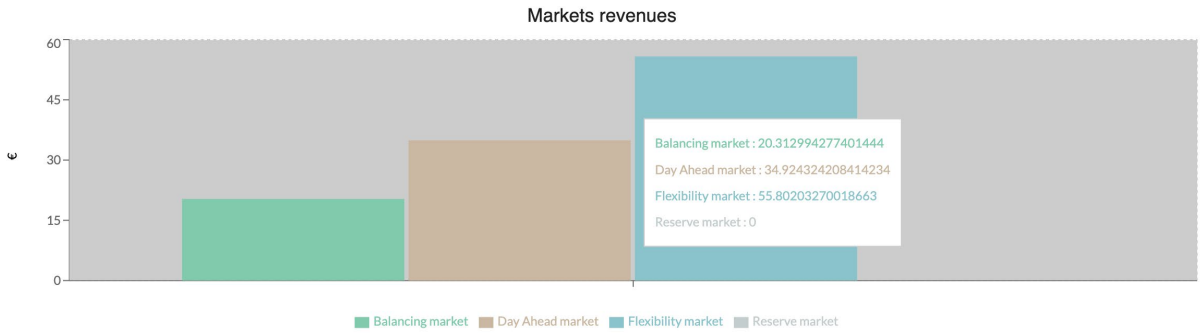


Figure 25: ESP's expected revenues for participating in DAM and FM and BM (services only to DSO)

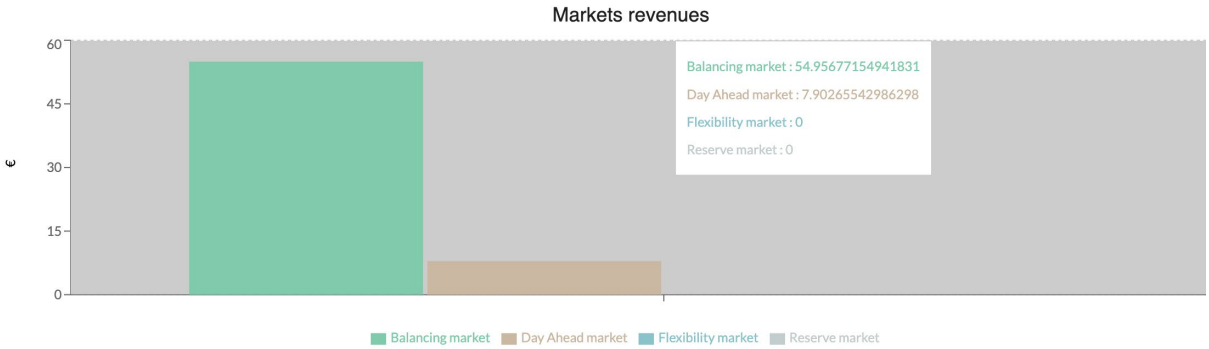


Figure 26: Similar to Case 2 but the DSO area 1 does not have a congestion/voltage control problem

We now run a scenario in which the storage capacity is doubled (i.e. 200 KW instead of 100 KW) and all the other input parameters are the same with the initial ones. As expected, we now see that the ESP's revenues are doubled (i.e. 245.69 euros in total). Of course, if we put one 100 KW storage unit in one DSO area and another 100 KW storage unit in another DSO area, the revenues will be less.

In another simulation, we just change the date (i.e. 03/03/2022 instead of 03/03/2021). As shown in Figure 27, the ESP's expected revenues have been considerably increased (i.e. 164.12 euros instead of 128.83 euros for date 03/03/2021). Given the fact that we used the same market prices for FM, the difference is mainly incurred by the increased energy/reserve prices that the EU has faced in early 2022.

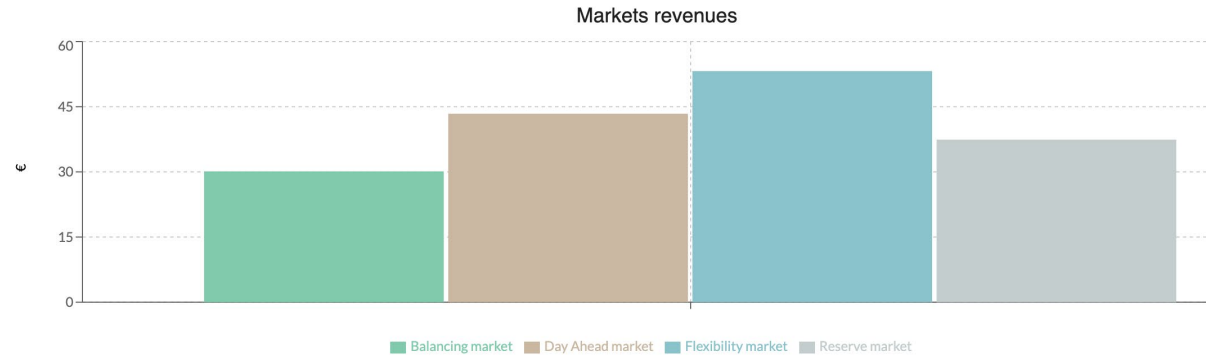


Figure 27: ESP's revenues for case 0 (03/03/2022 instead of 03/03/2021)

Finally, the ESP user can see all the historical results from all the past simulation runs via the "Historical" tab as shown in the figure below. Thus, s/he is able to compare them and hence decide about its optimal market participation policy and storage unit investments. Moreover, the FMO and DSO users are able to visualize all the bidding history of the ESP and thus be

able to analyze each ESP's business behavior or even cooperate with the ESP towards organizing local flexibility markets and thus realize win-win business cases.

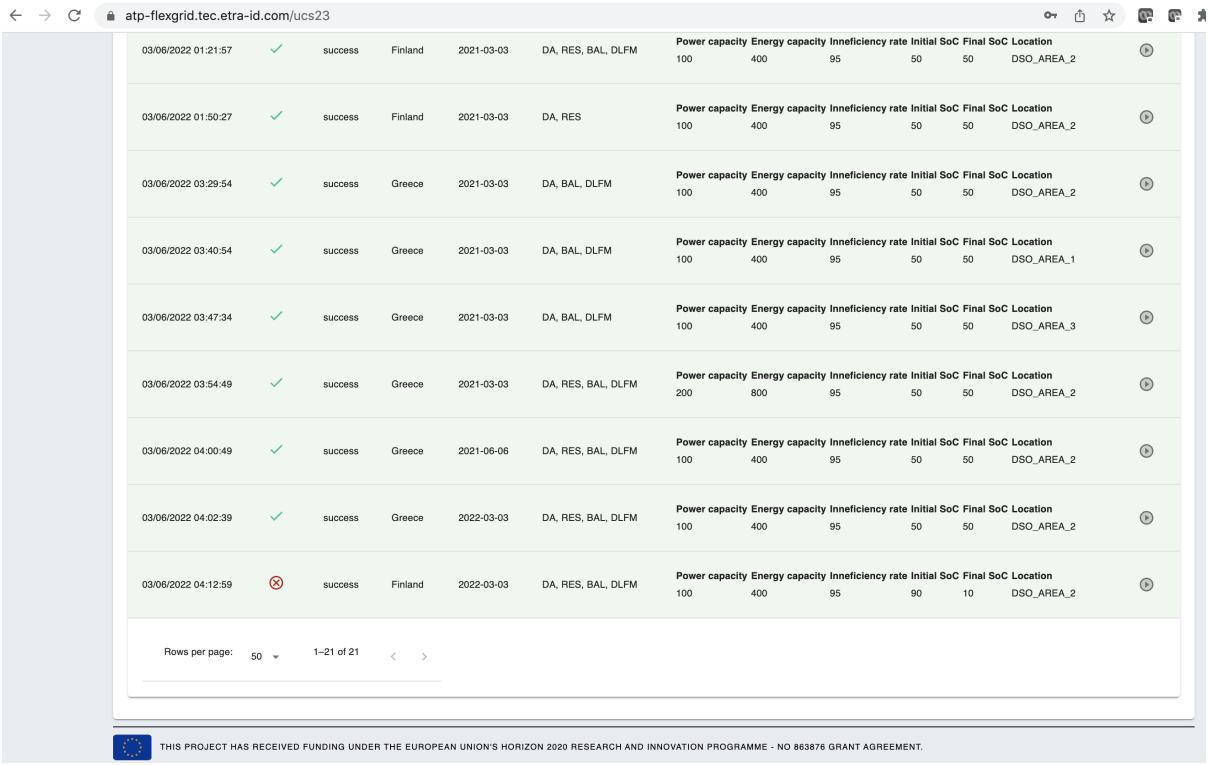


Figure 28: The ESP user can visualize and compare all past results via "Historical" tab

5.1.4 UCS 4.4a - Generation Forecasting Validation Results

When the Energy Service Provider (ESP) user has logged in to the FLEXGRID ATP successfully, then the user can select the UCS 4.4 service "PV generation forecasting" tab from the drop down menu at the upper left part of the screen (called "UCS Visualisation"), and the following screen (see Figure 29) will be appeared.

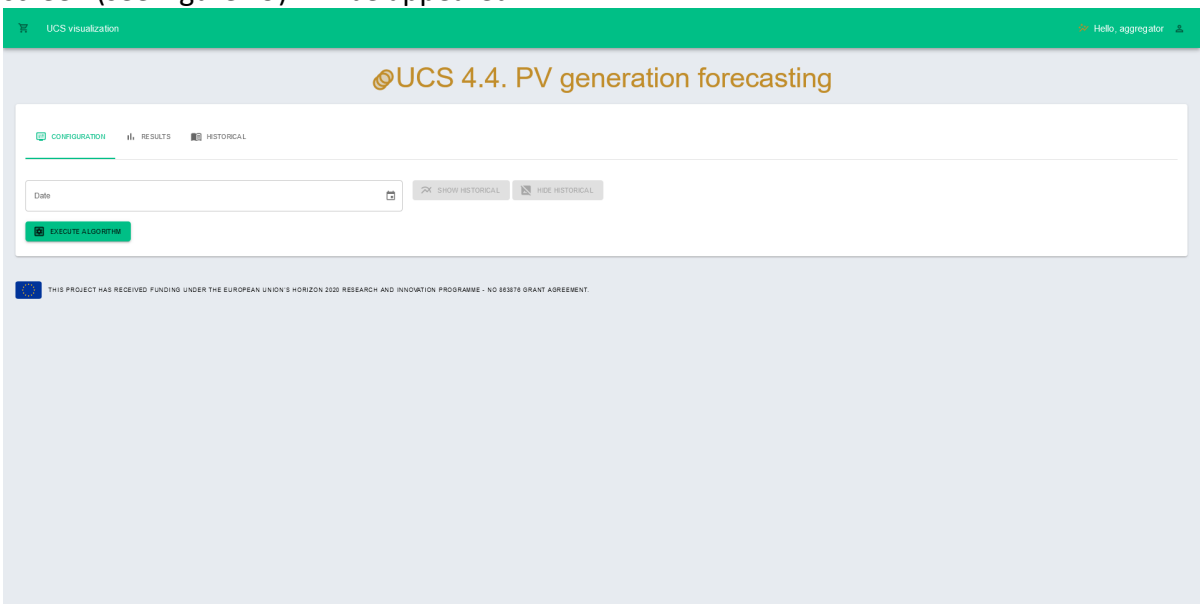


Figure 29: Front page of the web application for the UCS 4.4 service "PV generation forecasting".

To interact with the Graphical User Interface (GUI), the ESP user can select one from the three available tabs (i.e., Configuration, Results and Historical).
Through the “Configuration” tab, the ESP user can fill in the required input parameter (i.e., the “Date”). Once the user selects a specific date, then he/she has to press the “Execute Algorithm” button and the PV generation forecasts and actual data for that specific day are provided at the “Results” tab. For example, if the user sets the “Date” parameter to “29/06/2022” and then press the “Execute Algorithm” button, the actual and forecasted results will be available after a few seconds (time required for the results to be fetched from the FST server) at the “Results” tab. As it can be seen in Figure 21, from the “Results” tab the ESP user can see the details of the PV plant. These details include the:

- PV Plant’s name – “Name”,
- Filter used to acquire the data – “Filter”,
- Granularity of the data (if the granularity is set to “Yes” then the API returns hourly datasets) – “Granularity”,
- Day-ahead date (default value that can be overlooked) – “Date”,
- Date of the actual datasets against the forecasts (same date as the visualised plot) “Date actual” and
- Date of the calculated error (same as the date of the actual data sets) if available – “Date error”.

In addition, the visual comparison of the actual against the forecasted (predicted) power is also provided in the “Results” tab (see time plot graph in the figure below).

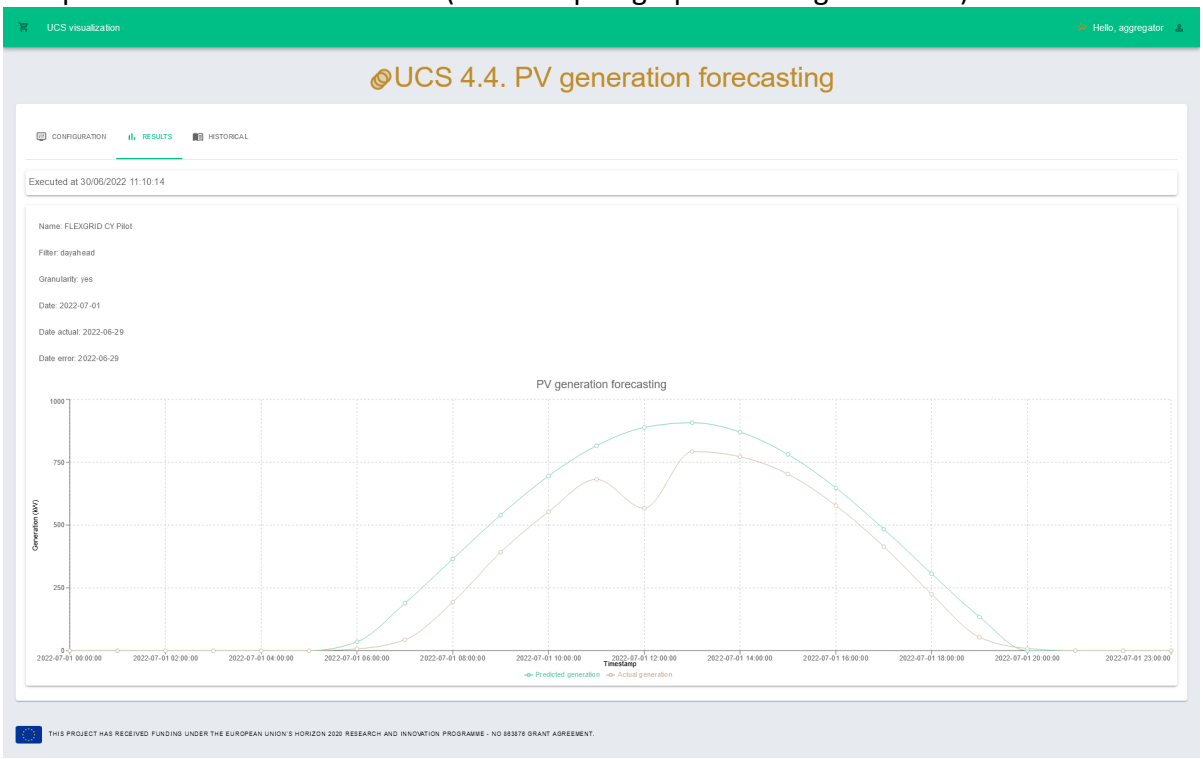


Figure 30: Actual vs the forecasted (predicted) PV generation - “Results” tab.

An additional feature of the “Configuration” tab is the “Show Historical” and “Hide Historical” buttons (see Figure 31). When the “Show Historical” button is selected, a visualisation of the actual against the predicted PV generation based on the historical data sets is demonstrated

in the same tab (“Configuration” tab), while the “Hide Historical” button hides the visualization of the “Show Historical” button.

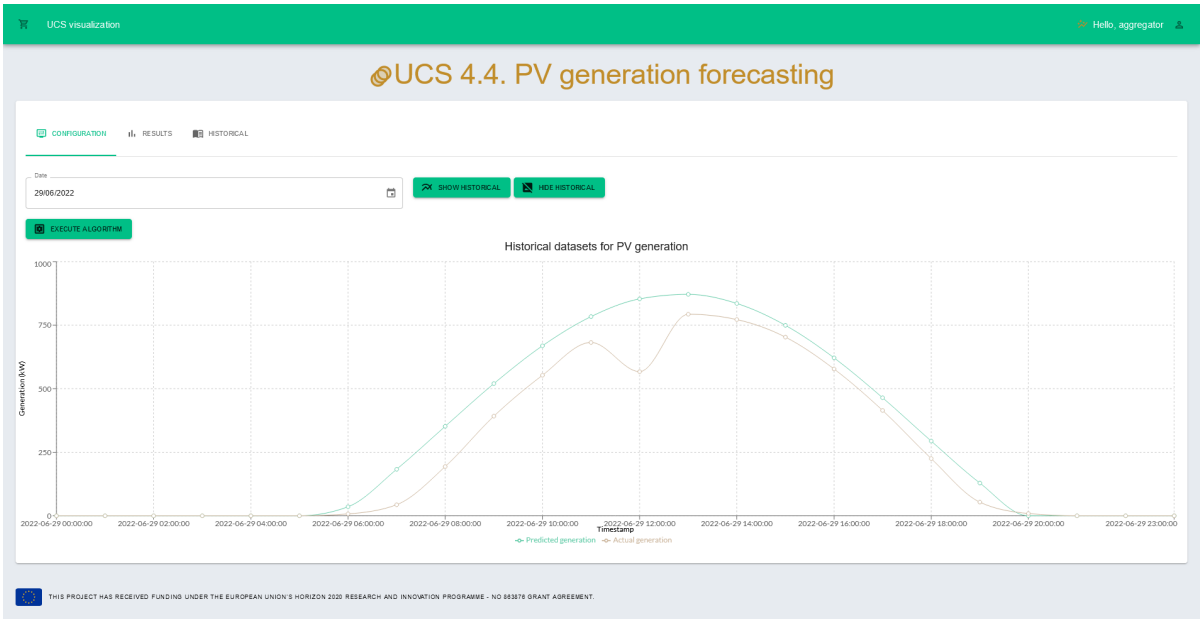


Figure 31: “Configuration” tab, “Show Historical” and “Hide Historical” buttons.

Finally, the ESP user can see all the historical results from past simulation runs via the “Historical” tab (see Figure 32). Thus, the user is able to compare the results and provide suggestions regarding the performance of the PV generation algorithm.

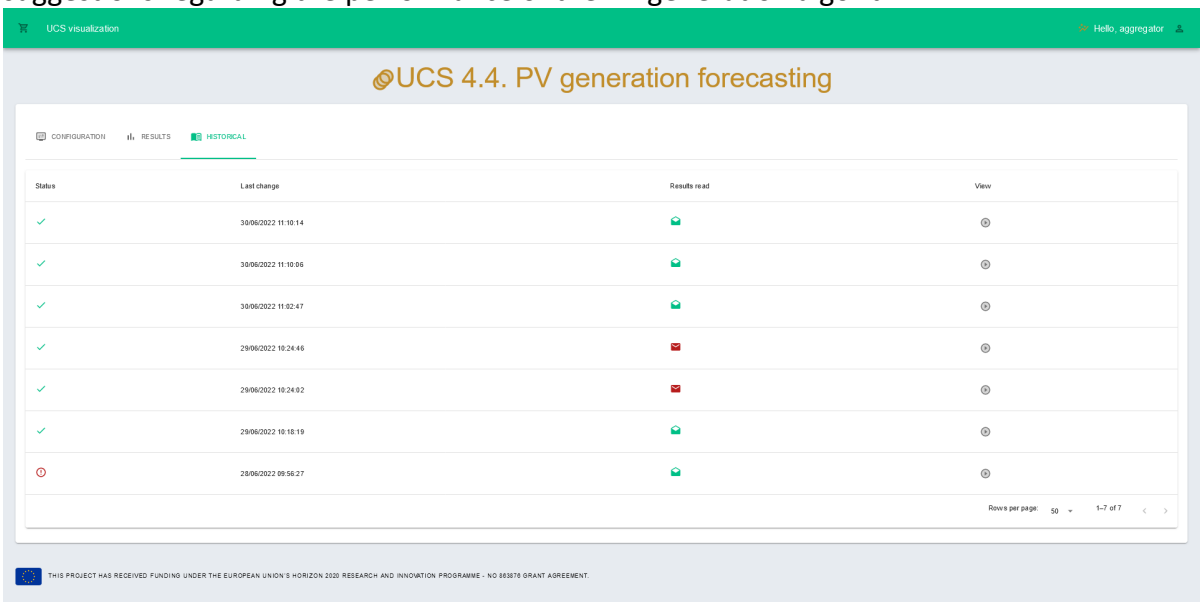


Figure 32: The ESP user can visualize and compare all past results via the “Historical” tab.

5.1.5 UCS 4.4b: Market Price Forecasting Service Validation Results

When the Energy Service Provider (ESP) connects to the FLEXGRID ATP, then one of the provided services is the Day Ahead Market Price Forecasting. When the user selects the UCS 4.4 service “Market price forecasting”, the following screen will be appeared (see Figure 33).

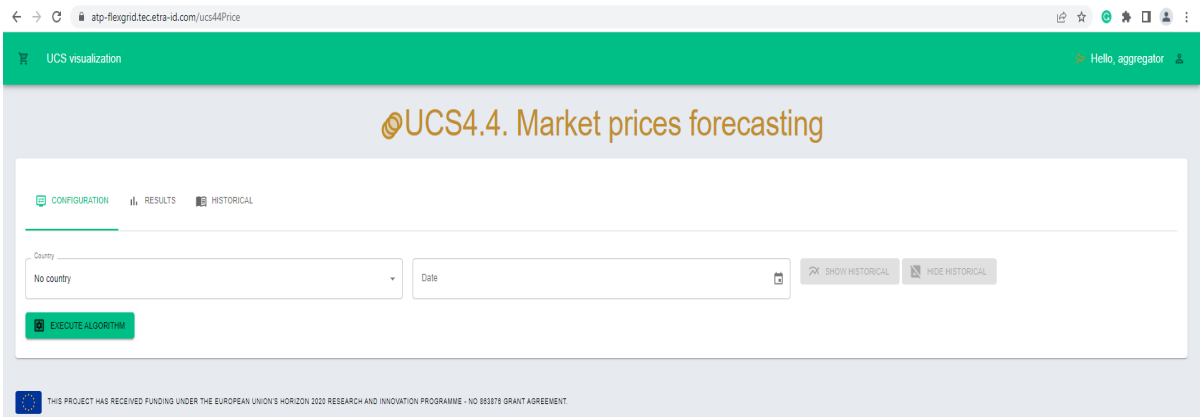


Figure 33: Front page of the platform for the UCS 4.4 service "Market prices forecasting".

Through the "Configuration" tab, the ESP user can fill in the required input parameters (i.e., "Country" and "Date"). Once the ESP user selects a specific date and country (from the list of countries that participate in the Nord Pool Day Ahead market), then he/she has to press the "Execute Algorithm" button and the Day Ahead forecasts and Actual Market prices for that specific date are provided at the "Results" tab (see Figure 34).

Figure 34: Day Ahead Forecasts and Actual Market Prices (for Austria in July 2022) - "Results" tab.

Finally, the ESP user can see all the historical results from all the past simulation runs via the "Historical" tab as shown in Figure 35. Therefore, the user can compare the results for different countries or dates.

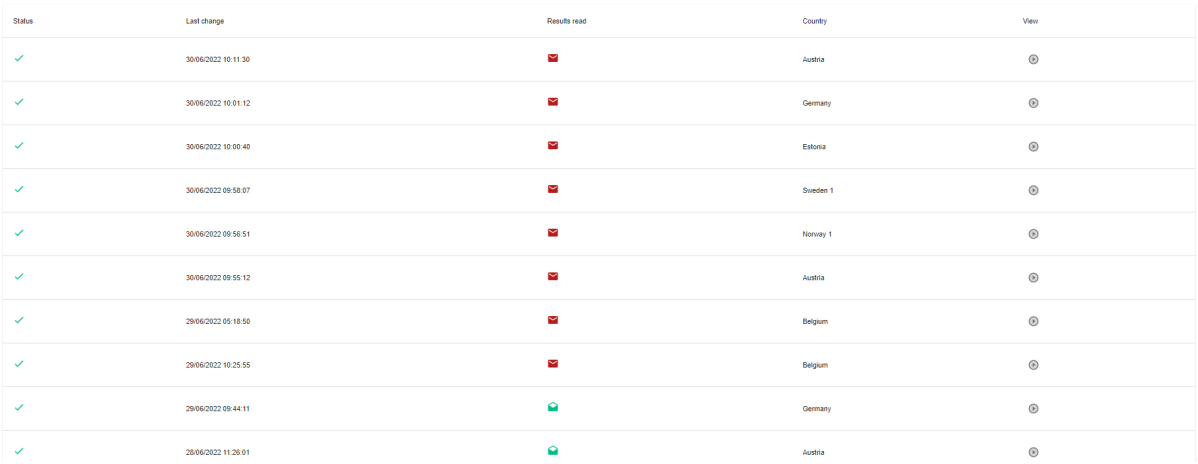


Figure 35: The ESP user can visualize and compare all past results via "Historical" tab.

5.2 AFAT GUIs and Results

5.2.1 UCS 4.1 – FlexRequest dispatch optimization

When the ESP user has been successfully logged in to the FLEXGRID ATP, then they can select from a drop-down menu at the upper left part of the screen, called "UCS Visualization," the UCS 4.1 service "FlexRequest dispatch optimizations" and following the link the specific page shown in Figure 36 will appear.

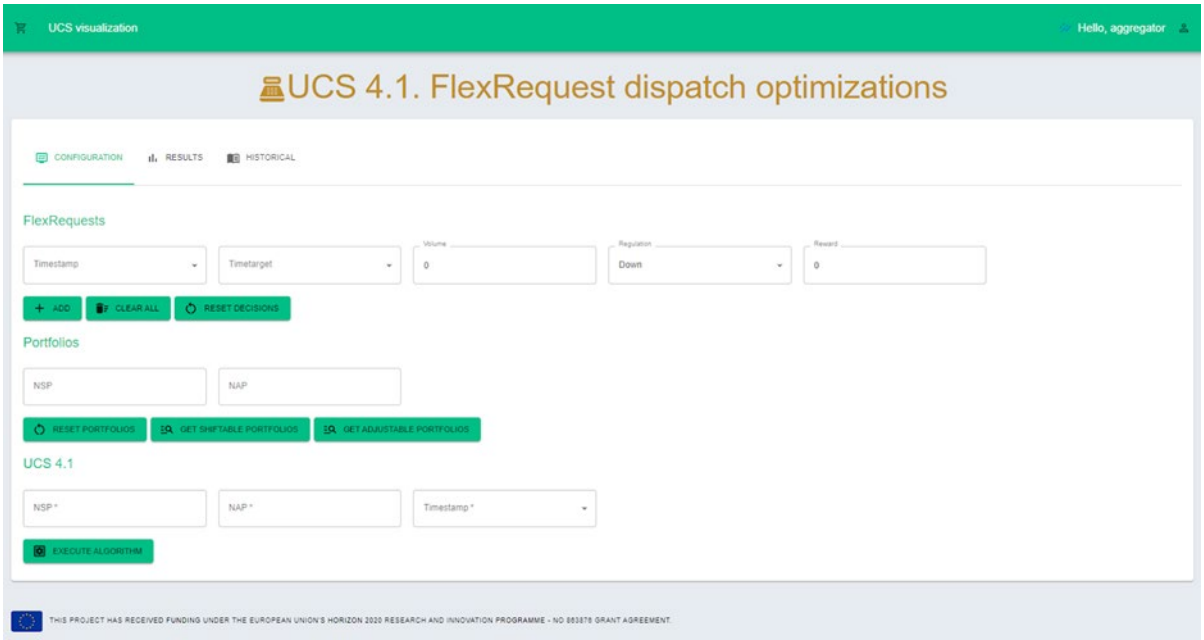


Figure 36 The ESP user selects UCS 4.1 service and fills in the mandatory input parameters via "Configuration" tab

To proceed with the operation of the GUI, the ESP user selects between three tabs. The first tab is the "Configuration" tab, which enables the ESP user to fill in the input parameters. Specifically, the input parameters that the ESP user can be filled in for the UCS 4.1 are:

- FlexRequest's specific parameters:
  - Timestamp: Time when the FlexRequest is published. Value in the interval [1-24].

- TimeTarget: Timeslot where deviation of energy is requested. Value in the interval [1-24].
- Volume: Amount of requested energy (flexibility) with a unit of kWh.
- Regulation: Direction of requested energy. Value can be Up or Down.
- Revenue: Compensation for the requested energy.

- Portfolios' specific parameters:
  - NSP: Number of Shiftable Portfolio. Value is a positive integer number from 0 to the number of available NSPs.
  - NAP: Number of Adjustable Portfolio. Value is a positive integer number from 0 to the number of available NAPs.

- UCS4.1 specific parameters for execution, while the previous parameters were used to initiate or reset the FlexRequests. The specific part of the screen is used to execute the UCS4.1 algorithm.
  - NSP: Number of Shiftable Portfolio. Value is a positive integer number from 0 to the number of available NSPs.
  - NAP: Number of Adjustable Portfolio. Value is a positive integer number from 0 to the number of available NAPs.
  - Timestamp: Current time. Value in the interval [1-24].

To initiate the procedure of UCS4.1 application, the FlexRequests portfolios must be cleared, therefore, the button "Clear All" should be selected and if the action is successful a message will be appeared on the screen informing the users that the FlexRequest were cleared, Figure 37 demonstrates the procedure. In addition, the "Reset Portfolios" (see figure below) should be selected to reset the output parameters of the UCS4.1 application.

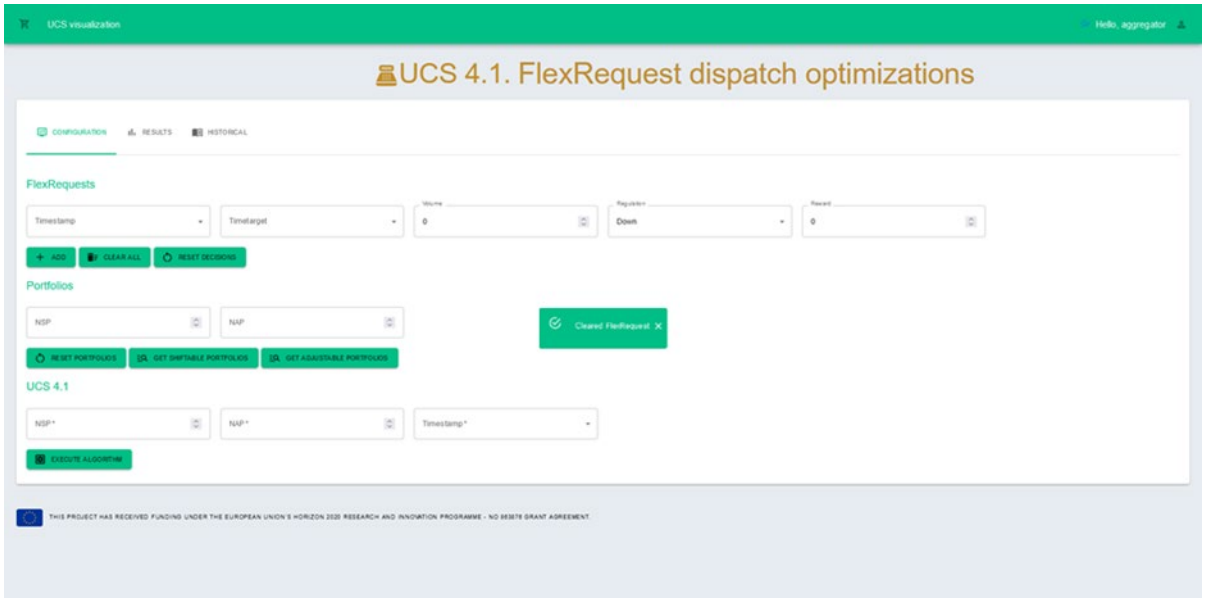


Figure 37 Clearing FlexRequest's portfolios to initiate the UCS 4.1 procedure

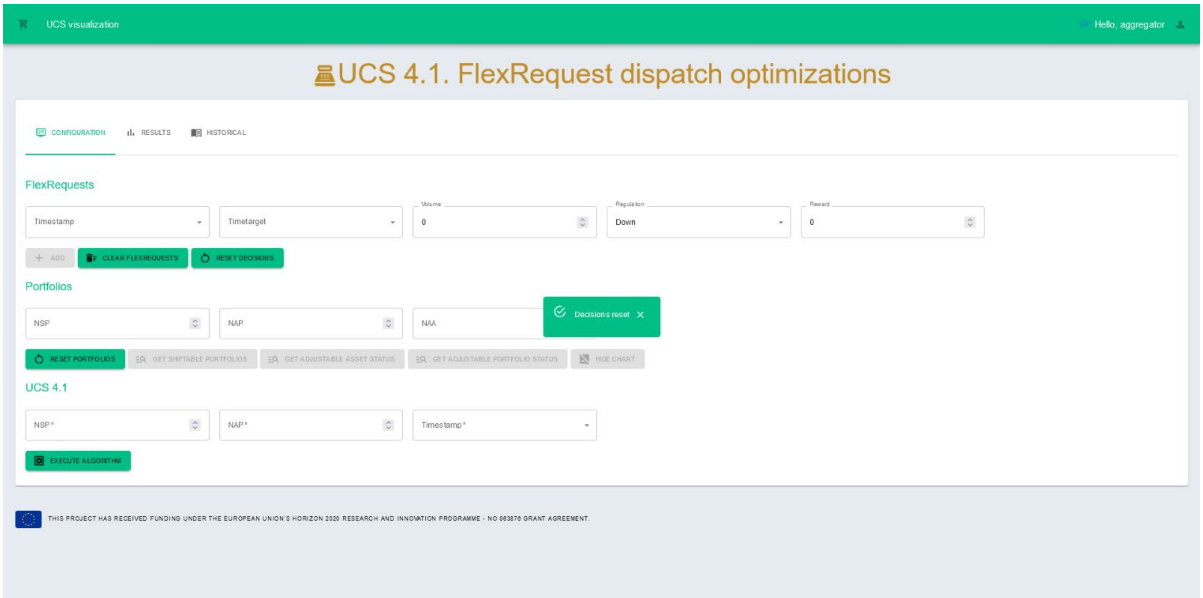


Figure 38 Resetting FlexRequest's outputs parameters to initiate the UCS 4.1 procedure.

FlexRequests-dispatch are issued at different timeslots and the ESP user needs to decide on the mandatory actions with a future-agnostic approach. Therefore, after the ESP user selects the “Clear All” and “Reset Portfolios'' buttons, they must fill in the mandatory fields and follow the appropriate procedure to acquire the desirable results. Specifically, to formulate a real-life scenario, a portfolio that consists of 10 end-users, 7 households and 3 small enterprises, where each end-user contributes to the flexibility portfolio with 2-3 shiftable assets and 1-2 adjustable assets was loaded to the FlexRequest-dispatch application. All assets are considered to have solely consumption patterns and all end-users are assumed to be a subset of the aggregator’s portfolio suitably located within the grid for responding to and serving the of received FlexRequests and the time horizon is a single day divided into 24 hourly timeslots/MTUs.

Specifically, the ESP user should firstly add a FlexRequest signal in order to indicate: i) time publishing of the FlexRequest, ii) the TimeTarget of the FlexRequest, iii) the amount of requested energy, iv) the direction of the regulation (that eventually will affect the direction of the deviating energy) and v) the compensation for the requested energy, therefore, the scenario will be comprised of a FlexRequest that is published at 04:00 AM and should be executed at 15:00, the volume of the request will be at 1 kWh with an Upward regulation and a Reward/Revenue at 35. Thus, the following parameters were be set (and serve as a real-life scenario):

- Timestamp = 4
- TimeTarget = 15
- Volume = 1
- Regulation = Up
- Revenue = 35

To add FlexRequest – Dispatch the “Add” button should be selected and eventually the user is informed that the FlexRequest is added (see Figure 40).

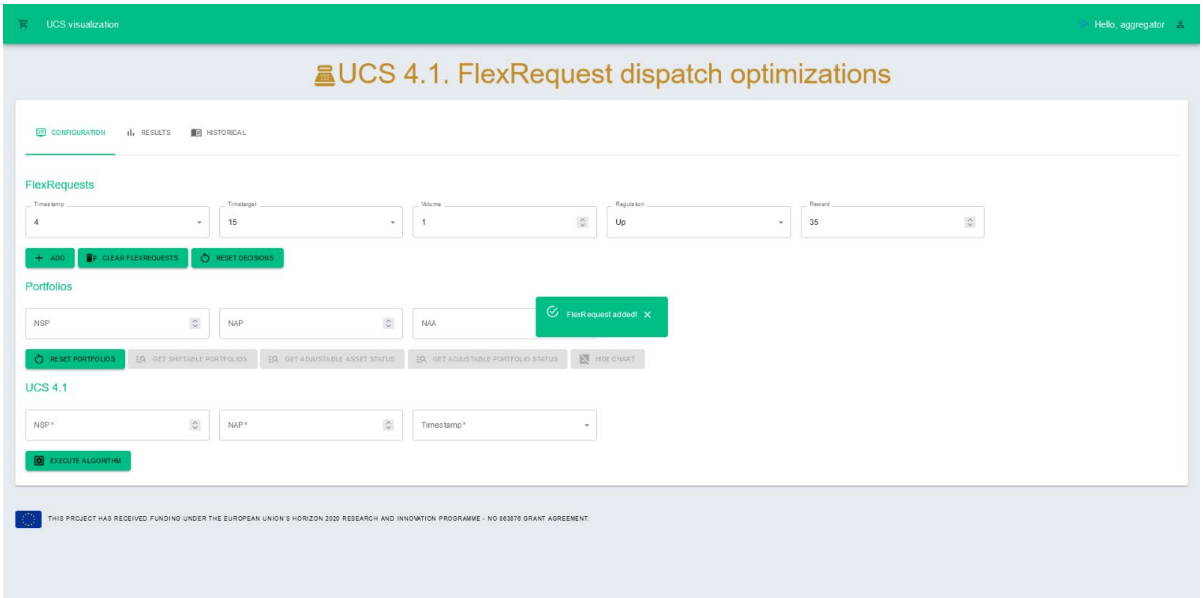


Figure 39 Adding a FlexRequest-dispatch signal

The next step of the execution of UCS4.1 is to load the NAP and NSP portfolios, therefore, the appropriate number for each portfolio is filled in and then the buttons “Get Shiftable Portfolios”, “Get Adjustable Portfolios” and “Get Adjustable Asset” must be selected. When the “Get Shiftable Portfolio” button is selected the plot of Scheduled Start Times vs Deviated Start Times that indicates the difference between the scheduled times (and deviations accordingly) and the Cost of deviation for each shiftable asset are demonstrated. At this scenario there are no cost of deviations for each shiftable asset and no deviation between scheduled times (see Figure 40).

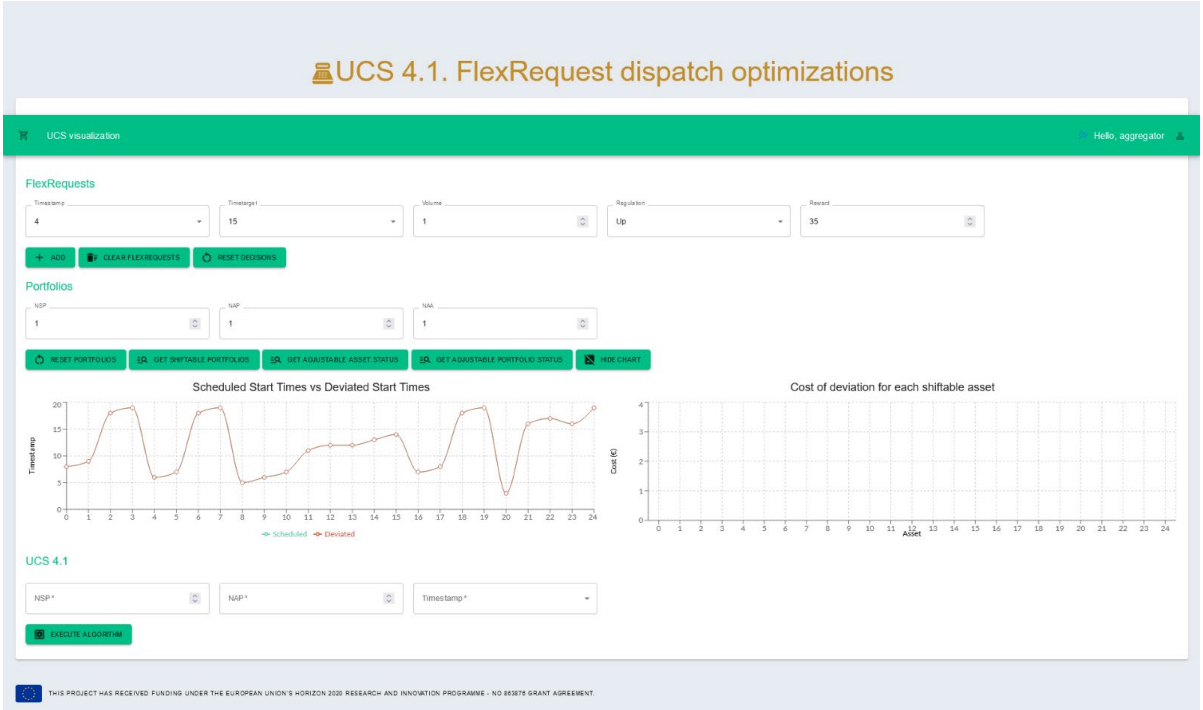


Figure 40 Loading the shiftable portfolios data on the platform

In addition, when the adjustable asset status is loaded, the plot of day ahead operation is demonstrated, which specifies the scheduled vs the deviated day ahead operations of the flexibility (see Figure 41).

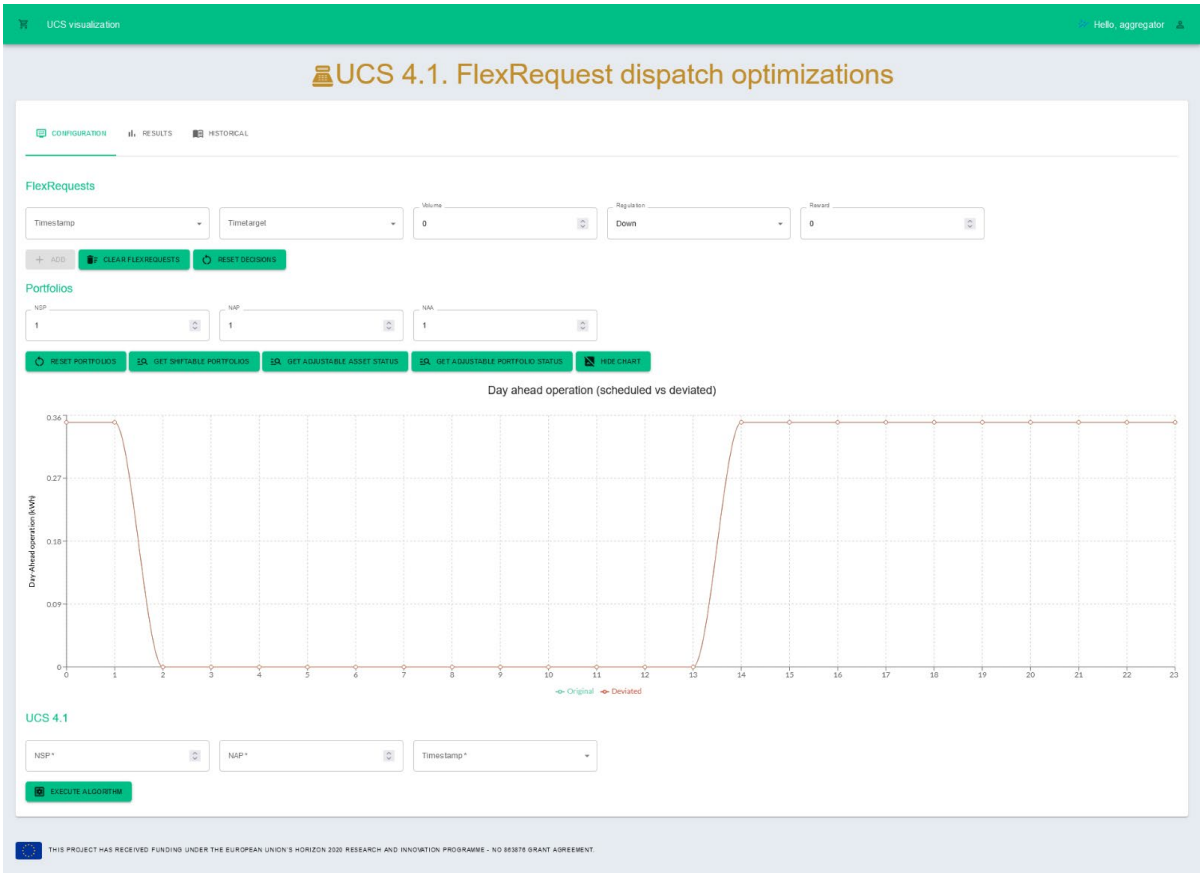


Figure 41 Loading the adjustable asset status data on the platform

Moreover, when the adjustable portfolio status is loaded, the plots of Scheduled Day-Ahead operation and Potential flexibility Up vs Potential flexibility Down are demonstrated that indicate the potential rewards (in Euro) per hour of the scheduled operation and the potential upward and downward directions of the flexibility, respectively (see Figure 43).

Figure 42 Loading the adjustable portfolio status data on the platform

The execution of the algorithm is performed by selecting the appropriate NAP and NSP portfolios, the appropriate execution Timestamp and by selecting the "Execute Algorithm" button. As can be seen in Figure 43, the ESP user can acquire the details of the FlexRequest submitted that includes:

- Current Reward: Compensation for the requested energy.
- Previous reward: Reward of accepted FlexRequests for previous runs of the service.
- Total Reward: CurrentReward + PreviousReward
- Total Cost SA: Cost of deviating operation of Shiftable Assets to comply with accepted FlexRequests.
- Total Cost AA: Cost of deviating operation of Adjustable Assets to comply with accepted FlexRequests.

In addition, the visual representation of the deviations of i) shiftable assets, ii) adjustable assets, iii) of the requested FlexRequests and iv) total deviations of assets. The specific scenario that was investigated demonstrated negative values of the energy deviation for the AA (negative values of deviating energy correspond to upwards regulation) and positive values of the energy deviation for the SA (positive values of deviated energy correspond to downwards regulation) for the Timestamp 11. However, the total deviating energy and the deviating energy of FlexRequest equals to 0 since the addition of the energy deviation of SA and AA equals to 0. On the other hand, for the Timestamp 15, the total deviating energy, and the energy deviation of the Flexrequest equals to -2 since the deviating energy of SA and AA are -0.5 and -2.0 respectively. The negative value of the total deviating energy indicates upwards regulation and eventually decrease of consumption/increase of generation.

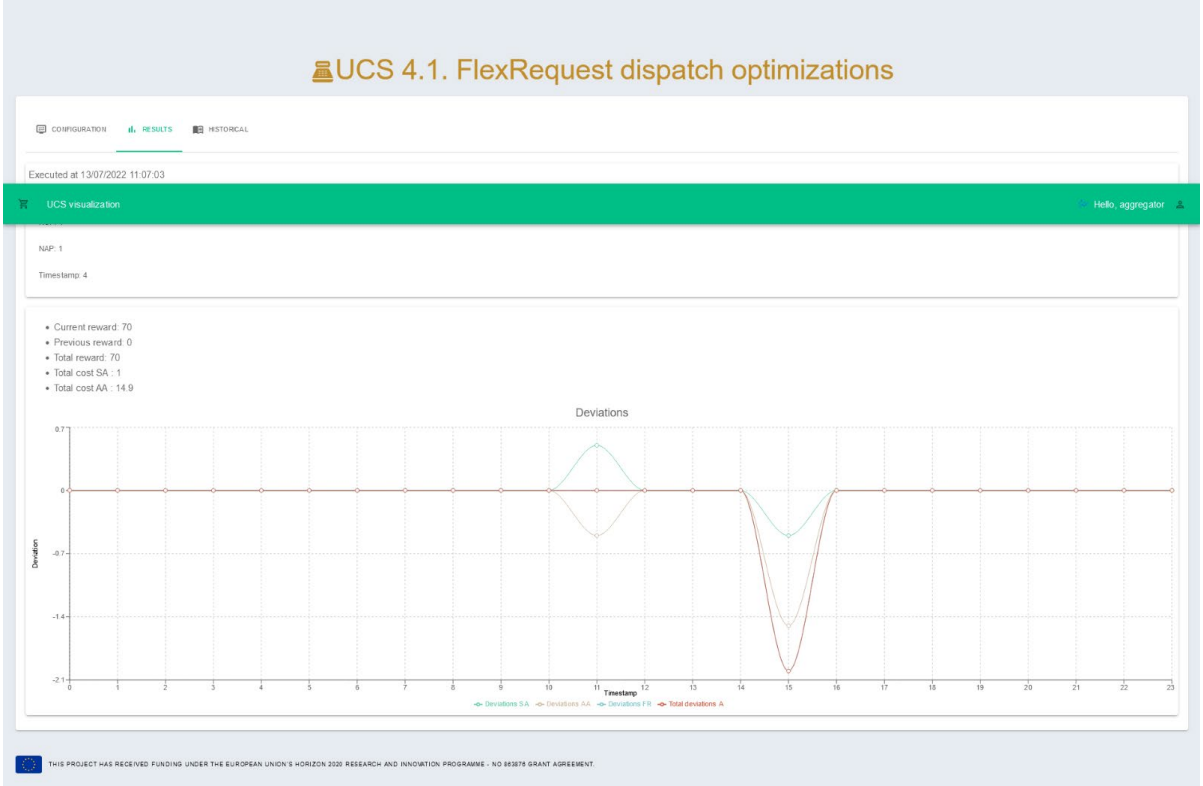


Figure 43 The ESP user visualizes the UCS4.1 FlexRequest's deviations.

An additional scenario was investigated that included the following parameters:

- Timestamp = 2
- TimeSlot = 6
- Volume = 2
- Regulation = Up
- Reward = 25

Figure 44 demonstrates the loading of shiftable portfolios, shiftable asset status and shiftable portfolio status. As can be seen in Figure 44a (shiftable assets) the from the 10th to 13th asset, therefore, there will be a cost for the deviation for each shiftable asset. While Figure 44b and 44c demonstrates the loading of shiftable asset status and shiftable portfolio status, respectively.

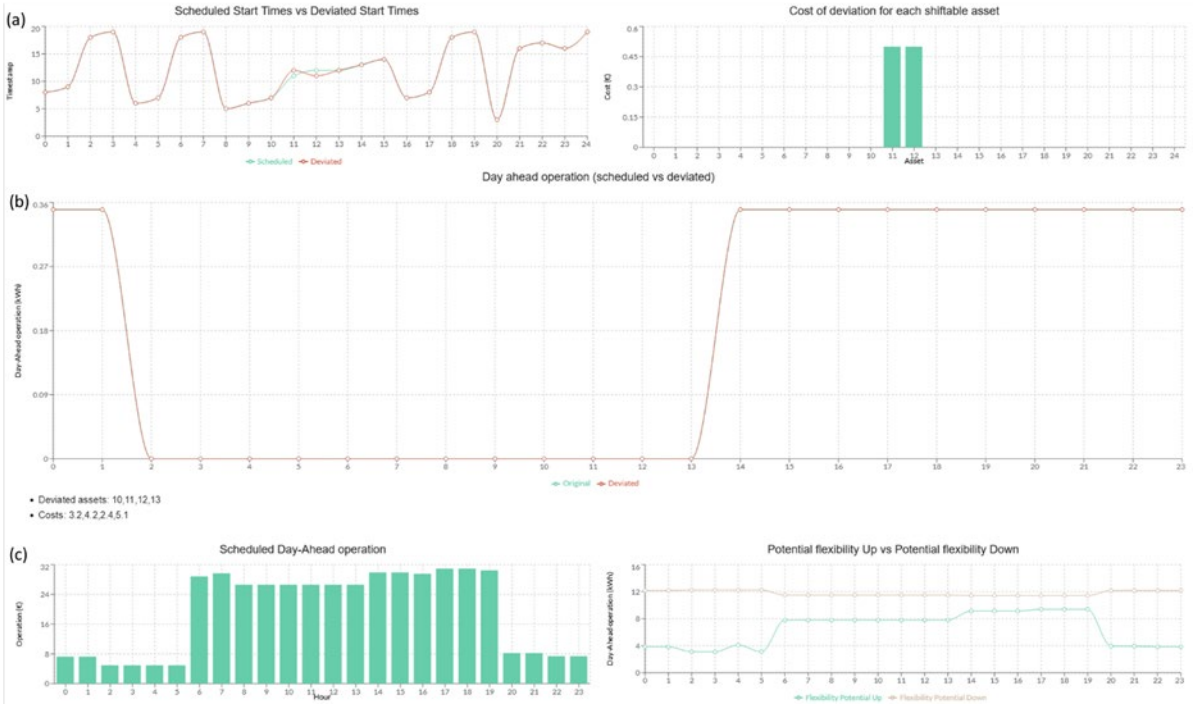


Figure 44 Loading: (a) the shiftable portfolio, (b) the adjustable asset status and (c) the adjustable portfolio status data on the platform

The specific scenario demonstrated a total energy deviation and FlexRequest deviation of -2 since the deviating energy of SA and AA are -0 and -1.5 respectively. The negative value of the total deviating energy indicates upwards regulation and eventually decrease of consumption/increase of generation.

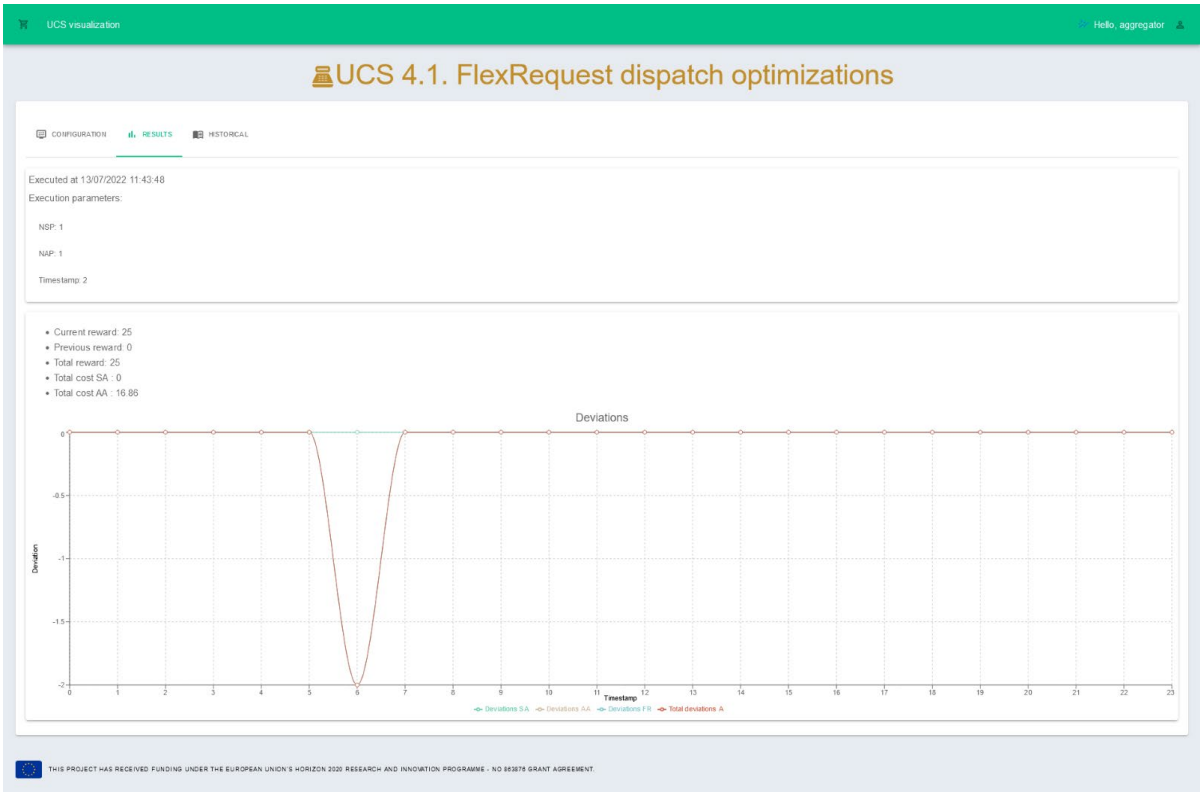


Figure 45 The ESP user visualizes the UCS4.1 FlexRequest's deviations

Finally, the ESP user can see all the historical results from all the past simulation runs via the "Historical" tab as shown in the figure below. Thus, they can compare them and decide/ provide suggestions regarding the UCS4.1 business case.

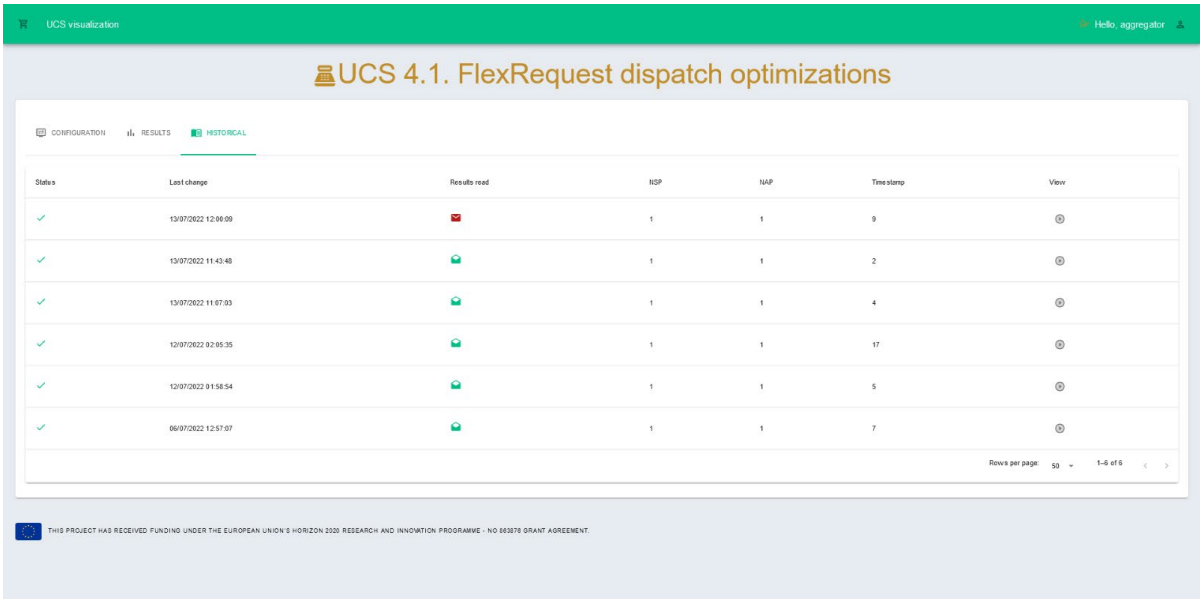


Figure 46 The ESP user can visualize and compare all past results via "Historical" tab

5.2.2 UCS 4.2 - Manage a B2C flexibility market

After the aggregator user has successfully logged in the FLEXGRID ATP, then s/he can select UCS 4.2 service ("Retail pricing optimization" or else "Manage a B2C flexibility market") and the following web page will appear:

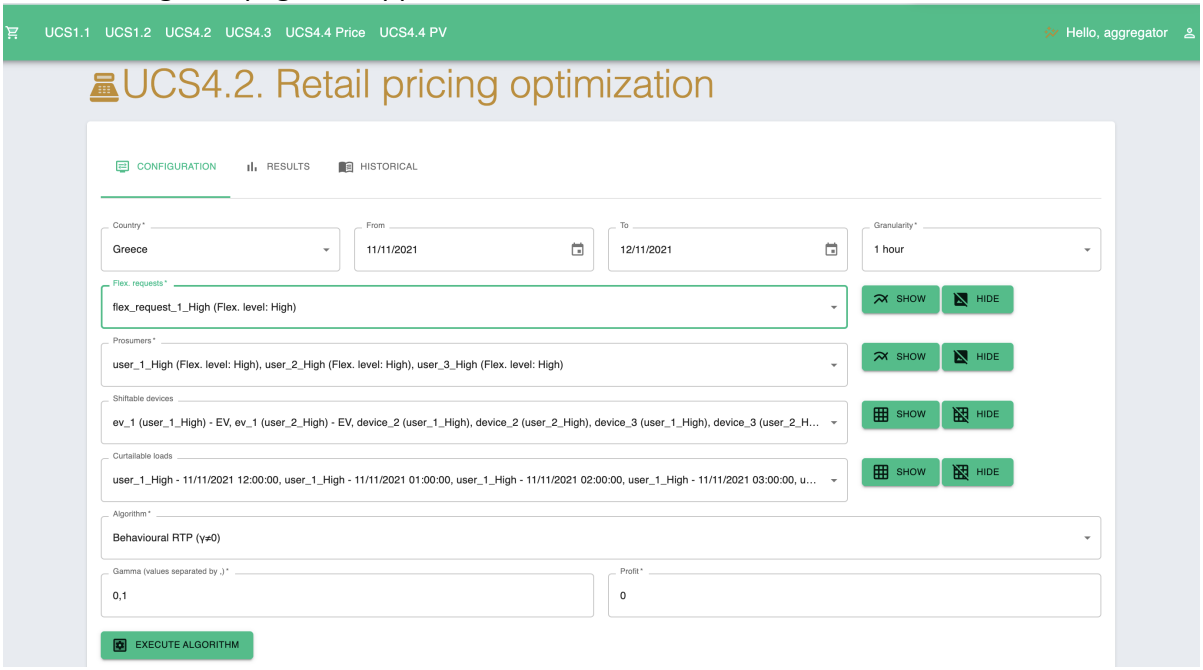


Figure 47: The aggregator user selects UCS 4.2 service and fills in the input parameters via "Configuration" tab

Now, the aggregator user can select the "Configuration" tab to fill in the input parameters. For example, s/he sets the country, date, time granularity, the type of FlexRequest (e.g. low,

medium, high), the set of end prosumers together with their flexibility offer profile (e.g. low, medium, high), the shiftable devices per end prosumer, the curtailable loads per end prosumer, the type of behavioral real-time pricing schemes (cf. “algorithm” field and the “gamma” parameter where various B-RTP schemes can be compared) and the aggregator’s business profit (%). For example, in the figure above, the aggregator runs the following “what-if” simulation scenario to figure out whether a new type of FlexContract will be more beneficial for both its end users (i.e. prosumers) and its own business profits. More specifically, in this simulation scenario, the aggregator’s business portfolio is located in Greece, one day with 24 hourly timeslots is simulated and flexibility demand is assumed to be high or else the FlexBuyer is willing to pay more for procuring a flexibility service via FLEXGRID ATP (cf. “FlexRequest_High”). Moreover, three end prosumers are selected with high flexibility. This means that they are willing to provide their flexibility with a relatively low price per unit (i.e. euros/kWh). In the fields “shiftable devices” and “curtailable loads”, the aggregator user can select (from a short-list) the exact subset of devices that is taken into account. In the “algorithm” drop-down menu, the aggregator user can select the type of behavioral pricing schemes that will be compared. In this scenario, the traditional Real-Time Pricing (RTP with γ=0) is compared with the proposed Behavioral Real Time Pricing (B-RTP with γ=1). Finally, the profit margin for the aggregator business is set to 0, which means that all monetary gains from flexibility revenues are entirely dispersed to the end prosumers involved, while the aggregator does not gain any respective extra profit.
After the mentioned steps the aggregator user is ready to select the “Execute Algorithm” button and waits a few seconds until the results are fetched back from the AFAT server. As shown in the following figure, the aggregator user can visualize the ratio between the Aggregated Users’ Welfare (AUW) with B-RTP (γ=1) and RTP (γ=0). Thus, we can see that the AUW in B-RTP (γ=1) is slightly increased by 2.15%.

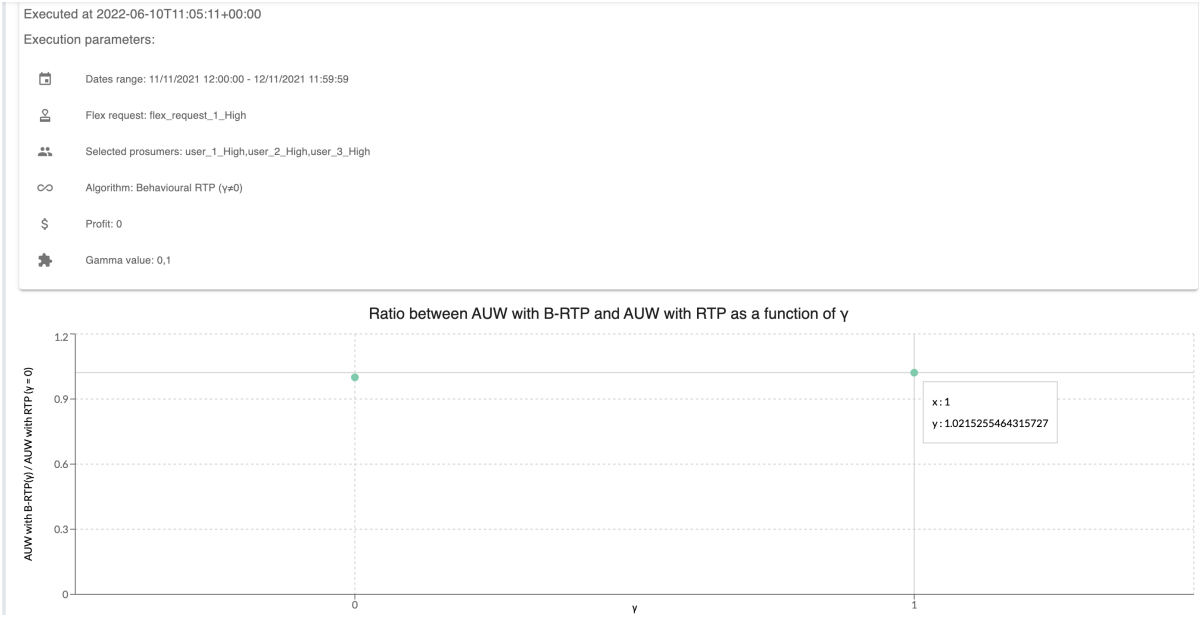


Figure 48: The aggregator user visualizes the Aggregated Users’ Welfare (AUW) difference via the “Results” tab (case 0)

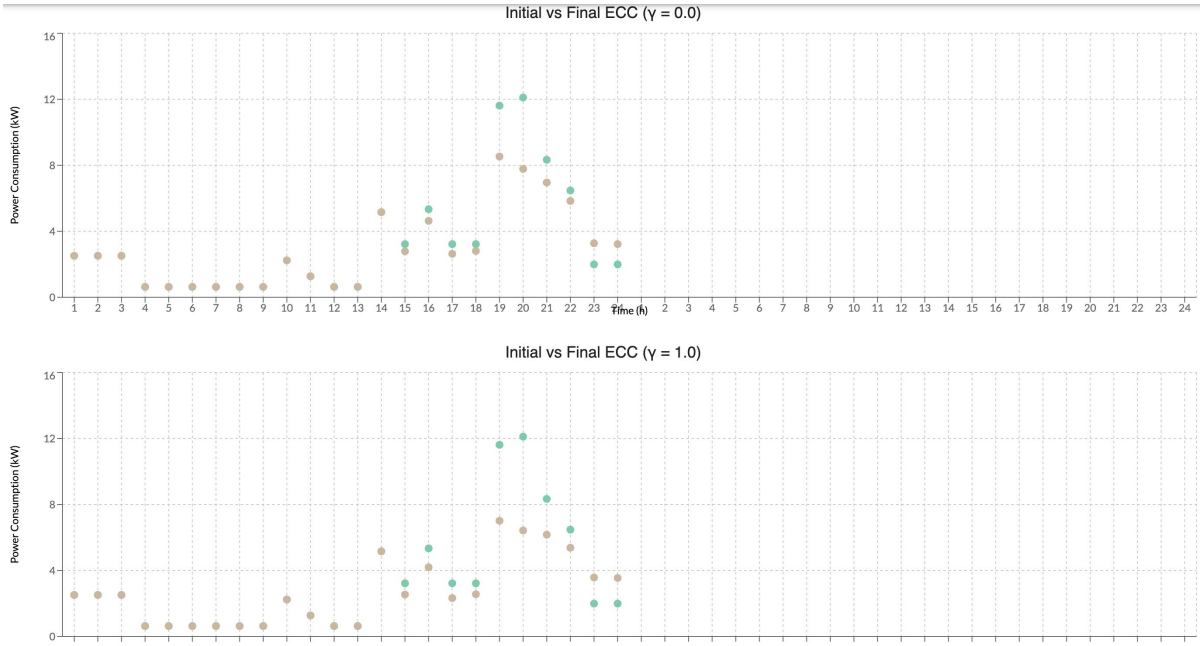


Figure 49: The aggregator user visualizes the initial vs. final aggregated Energy Consumption Curve (ECC) via the “Results” tab (case 0)

Furthermore, the aggregator user visualizes the initial vs. final Energy Consumption Curves (ECC). As shown in the figure above, when γ=0, the reduction of the final ECC is smaller than when γ=1. For both ‘γ’ values, one may observe that the energy consumption is slightly reduced in timeslots 15:00-18:00, while a larger reduction takes place during 19:00-21:00. On the other hand, during 23:00-00:00, a “rebound” effect is observed due to the late operation of shiftable devices.

Next, the aggregator user can also view the total flexibility quantity delivered and total flexibility revenues. In particular, when γ=0, the quantity is 12.91 kW, while when γ=1, the quantity is 19.24 kW. This 49% increase is explained by the fact that with B-RTP the highly flexible end users are incentivized to appropriately curtail/shift their loads. Regarding the flexibility revenues, these are increased from 5.38 euros to 7.25 euros, which is a ~35% increase. Given the fact that profit parameter is 0, all these monetary gains are exploited by the flexible end users, who will see a respective discount in their electricity bills.

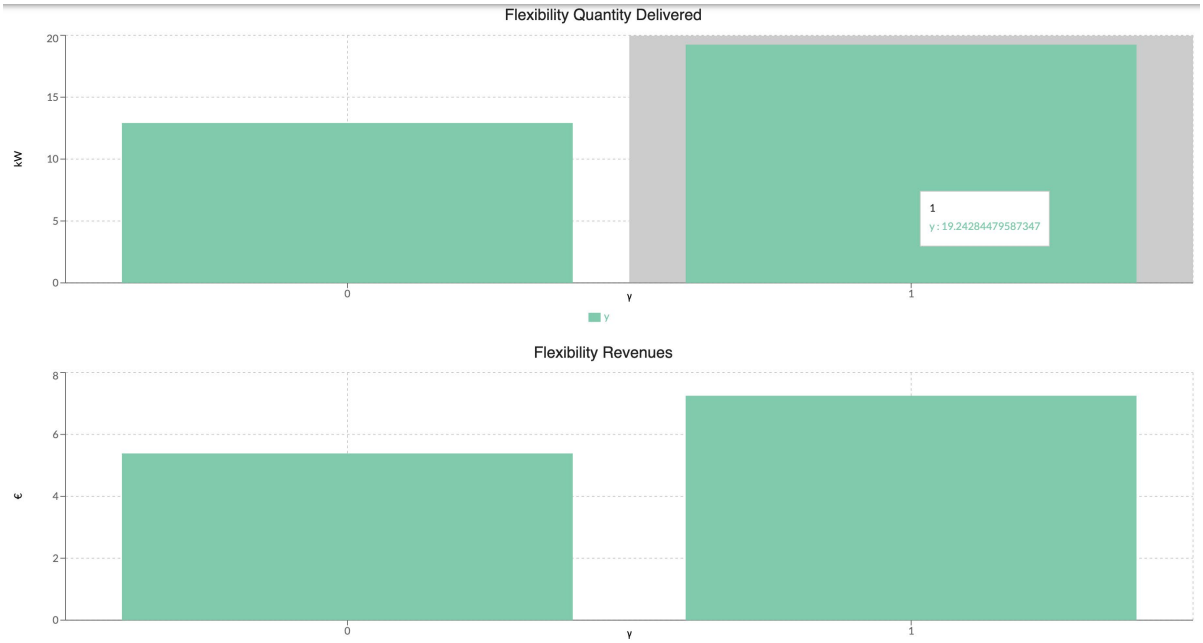


Figure 50: The aggregator user visualizes the total flexibility quantity delivered and the total flexibility revenues via the "Results" tab (case 0)

Finally, based on the figure below, the aggregator user can understand how the aggregated users' welfare (AUW) is divided among the involved end users. For example, the first graph in the figure below depicts that all three end users have the same user's welfare (i.e. the respective ratio equals to 1). However, in the second graph, one may observe that user 1 is more satisfied with B-RTP (γ=1) by ~14%, while user 2 and user 3 are less satisfied by ~1% and ~4% respectively. This means that these end users have provided their flexibility but in turn they lose a small part of their convenience/comfort levels (or else their individual flexibility revenues are relatively small compared to their comfort loss). These graphs are quite important for aggregator's business, but it can easily understand each individual end user's expected behavior and thus be able to adapt its pricing policy or else recommend more targeted and personalized FlexContracts.

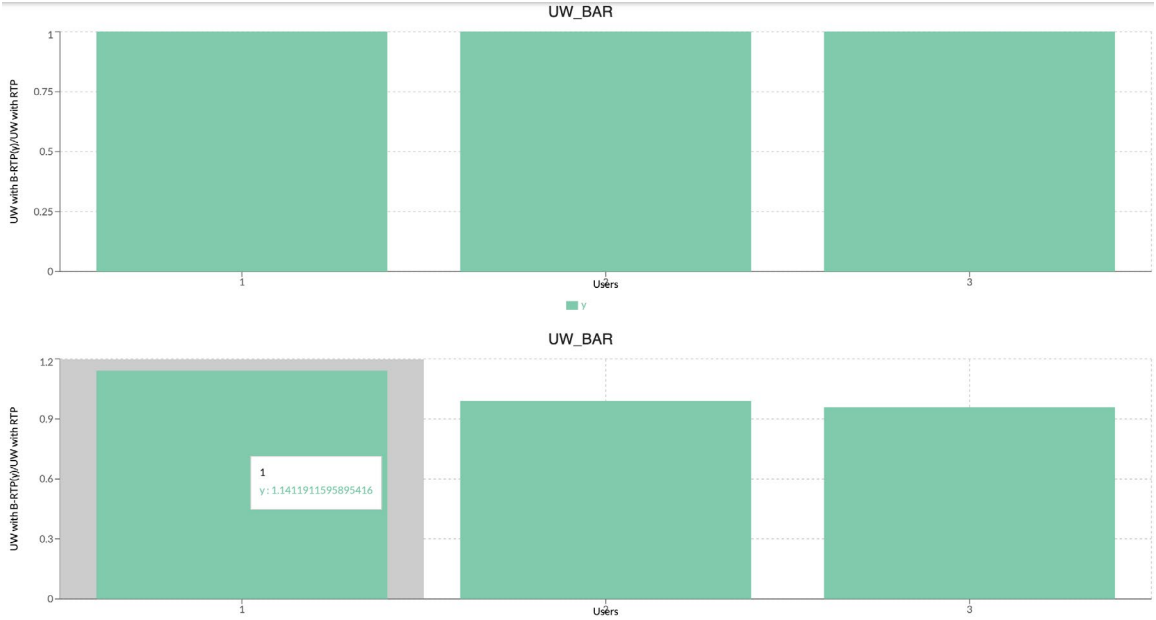
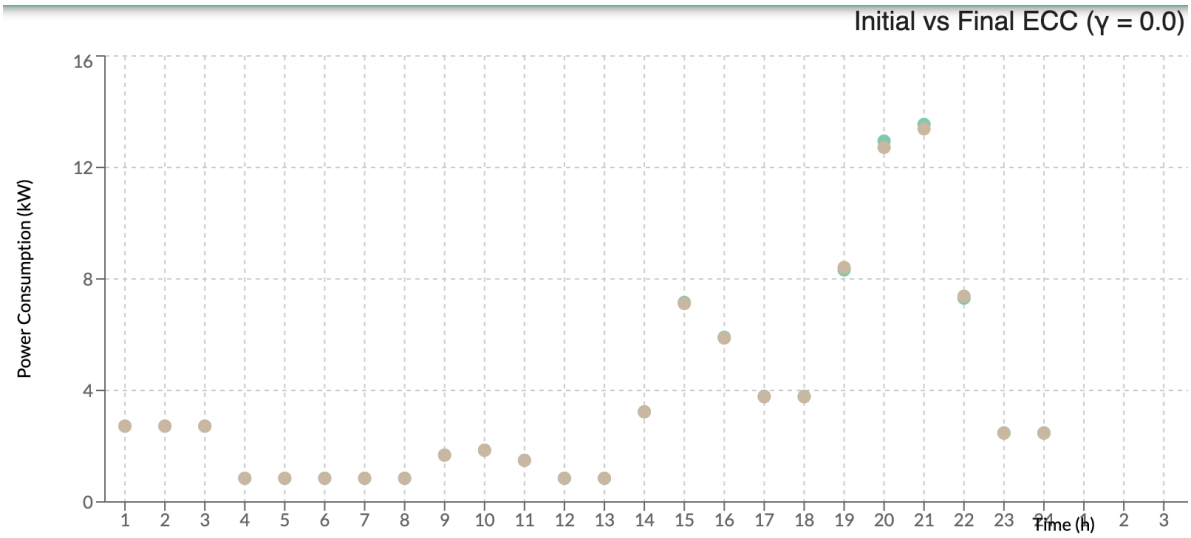


Figure 51: The aggregator user visualizes the welfare per individual end user (UW) via the “Results” tab (case 0)



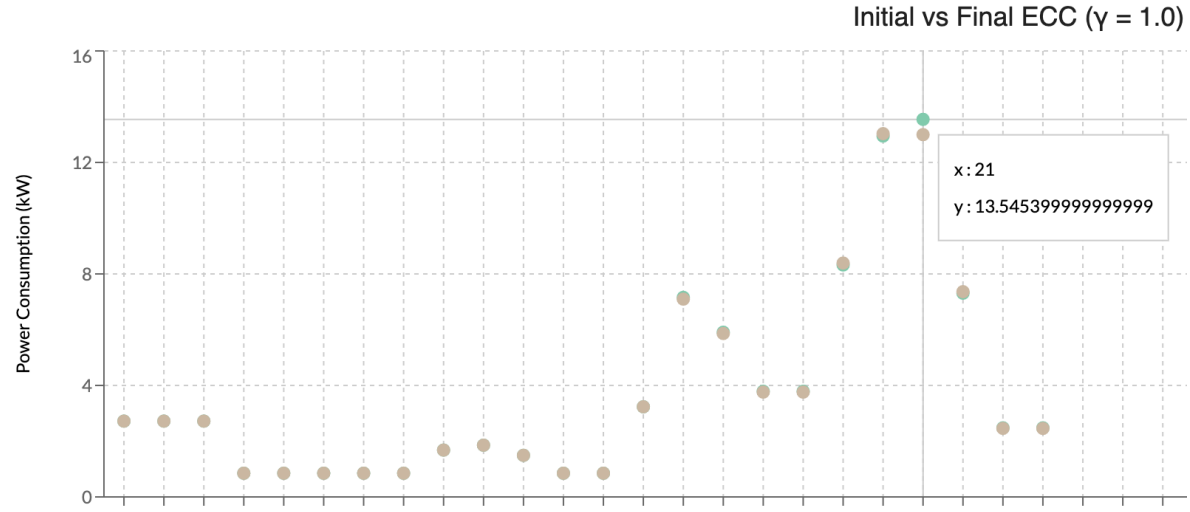


Figure 52: Initial vs. final ECCs for case 1

Now, let’s assume case 1 in which all input parameters remain the same except for the type of FlexRequest and the type of end prosumers, whose values are now set to ‘low’. As shown in the figure above, the energy reduction is much less compared to case 0, which is rational due to the fact that the end prosumers offer much less flexibility and the FlexBuyer is willing to pay much less for each delivered flexibility unit. The figure below is similarly explained. One may observe that the revenues are just 0.36 KW and 0.55 KW when γ=0) and γ=1 respectively. Flexibility revenues are just 0.09 euros and 0.12 euros respectively.

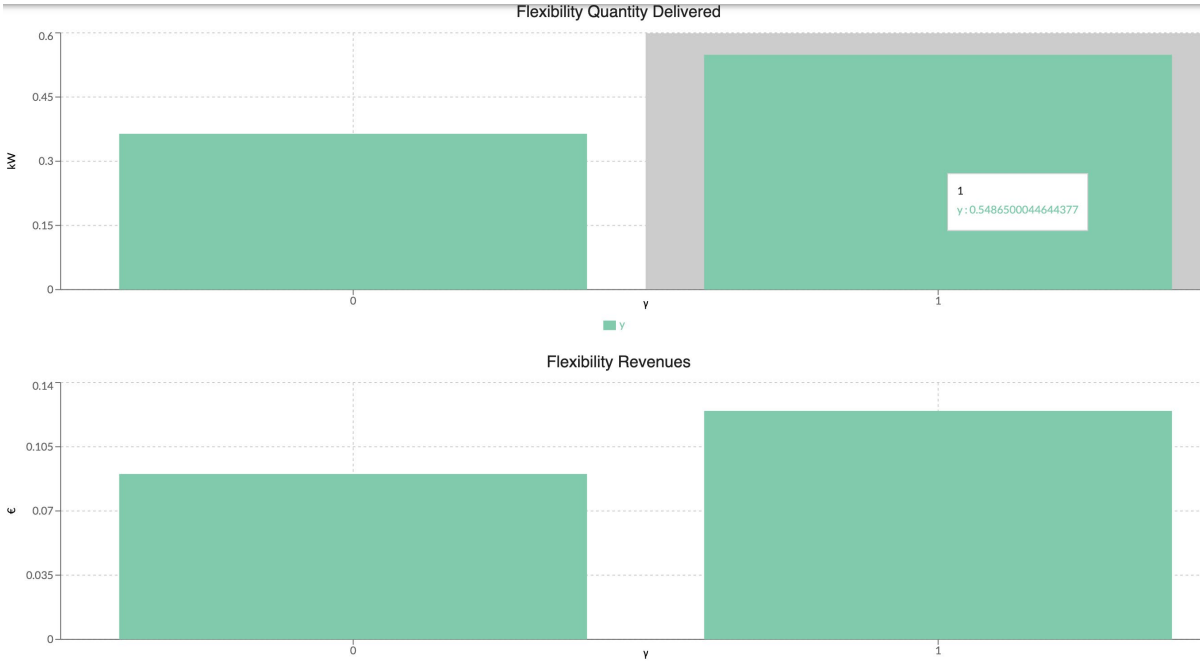


Figure 53: Total flexibility quantity delivered and total flexibility revenues for case 1

18/05/2022 02:28:41	✓	success	11/11/2021 12:00:00	11/11/2021 11:59:59	flex_request_1_Medium	user_1_Medium, user_2_Medium, user_3_Medium	Behavioural RTP (γ≠0)	0	01
07/06/2022 06:52:39	✓	error	01/06/2022 12:00:00	07/06/2022 11:59:59	flex_request_1_Low_Large_Up	user_2_Medium, user_10_Low	Fixed pricing	1	00.50.7
10/06/2022 11:05:56	✓	success	11/11/2021 12:00:00	11/11/2021 11:59:59	flex_request_1_Low	user_1_Low, user_3_Low, user_5_Low	Behavioural RTP (γ≠0)	0	00.54
10/06/2022 12:08:57	✓	success	11/11/2021 12:00:00	12/11/2021 11:59:59	flex_request_1_Medium	user_1_Low, user_2_Low, user_3_Low	Behavioural RTP (γ≠0)	0	0.51
10/06/2022 12:18:40	✓	error	11/12/2021 12:00:00	12/11/2021 11:59:59	flex_request_1_Medium	user_1_Medium, user_2_Medium, user_3_Medium	Behavioural RTP (γ≠0)	0	00.5
10/06/2022 12:25:59	⊗	error	11/12/2021 12:00:00	12/11/2021 11:59:59	flex_request_1_High	user_1_Medium, user_2_Medium, user_3_Medium	Behavioural RTP (γ≠0)	0	01
10/06/2022 12:29:10	✓	success	11/11/2021 12:00:00	12/11/2021 11:59:59	flex_request_1_High	user_1_Medium, user_2_Medium, user_3_Medium	Behavioural RTP (γ≠0)	0	01
10/06/2022 02:05:17	✓	success	11/11/2021 12:00:00	12/11/2021 11:59:59	flex_request_1_High	user_1_High, user_2_High, user_3_High	Behavioural RTP (γ≠0)	0	01
10/06/2022 05:21:19	✓	success	11/11/2021 12:00:00	12/11/2021 11:59:59	flex_request_1_Low	user_2_Low, user_3_Low, user_4_Low	Behavioural RTP (γ≠0)	0	01
10/06/2022 05:39:24	⊗	error	11/12/2021 12:00:00	12/11/2021 11:59:59	flex_request_1_High	user_2_Low, user_2_Medium, user_2_High	Behavioural RTP (γ≠0)	0	012

Figure 54: The aggregator user can visualize and compare all past results via “Historical” tab

Finally, the aggregator user can see all the historical results from all the past simulation runs via the “Historical” tab as shown in the figure below. Thus, s/he is able to compare them and hence decide about its optimal behavioral pricing policy (i.e. specific type of new FlexContract to recommend), which may be adapted in a personalized manner per individual end user based on the latter’s needs and desires.

5.2.3 UCS 4.3 - Create an aggregated FlexOffer

After the aggregator user has successfully logged in the FLEXGRID ATP, then s/he can select UCS 4.3 service (“Create an aggregated FlexOffer” or else “Flexibility offer optimizations” and the following web page will appear:

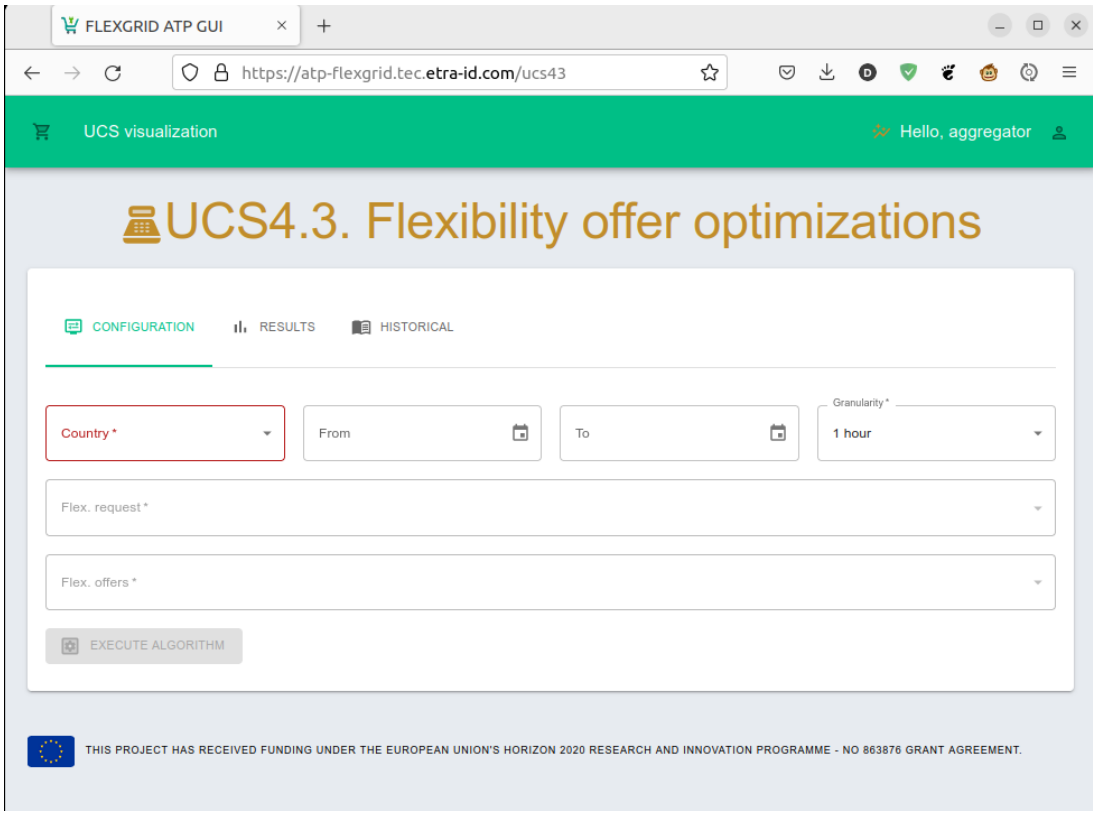


Figure 55: The Flexibility offer optimizations screen

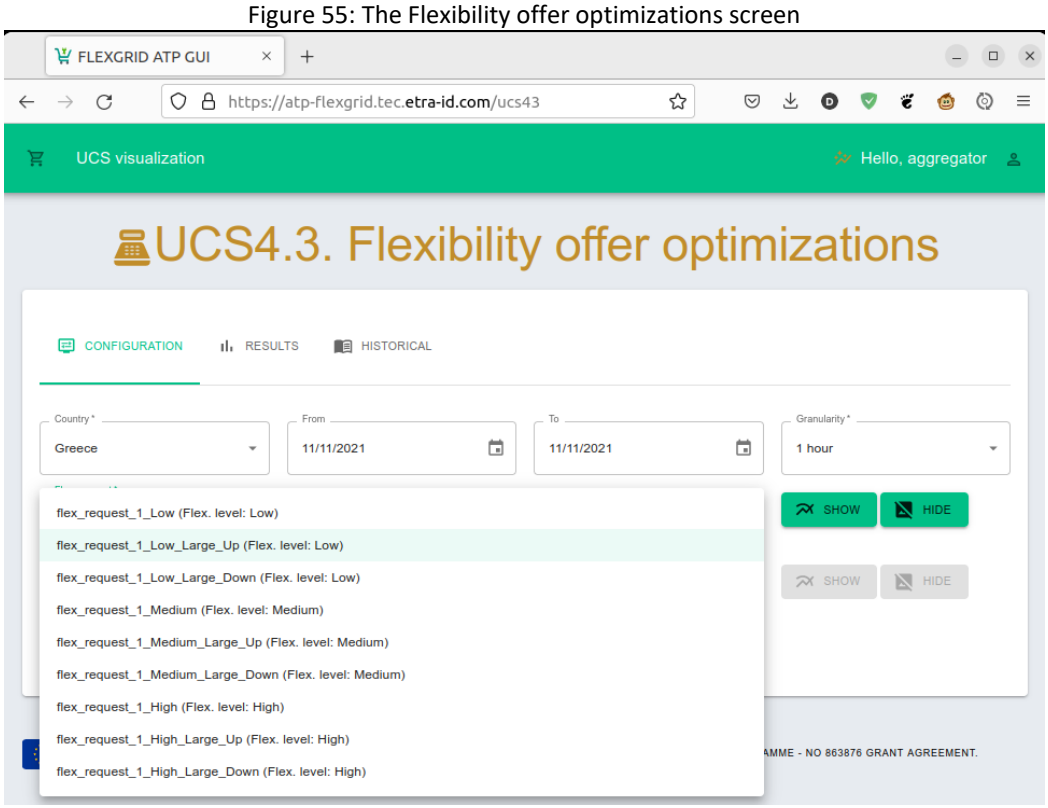


Figure 56: Selecting the FlexRequest to be used for the evaluation of the aggregate FlexOffer

Now, the aggregator user can select the “Configuration” tab to fill in the input parameters. For example, s/he sets the country, from/to dates, and time granularity. A FlexRequest may

also be selected, which represents the demand bid that will be used for calculating the revenues for the aggregated FlexOffer.

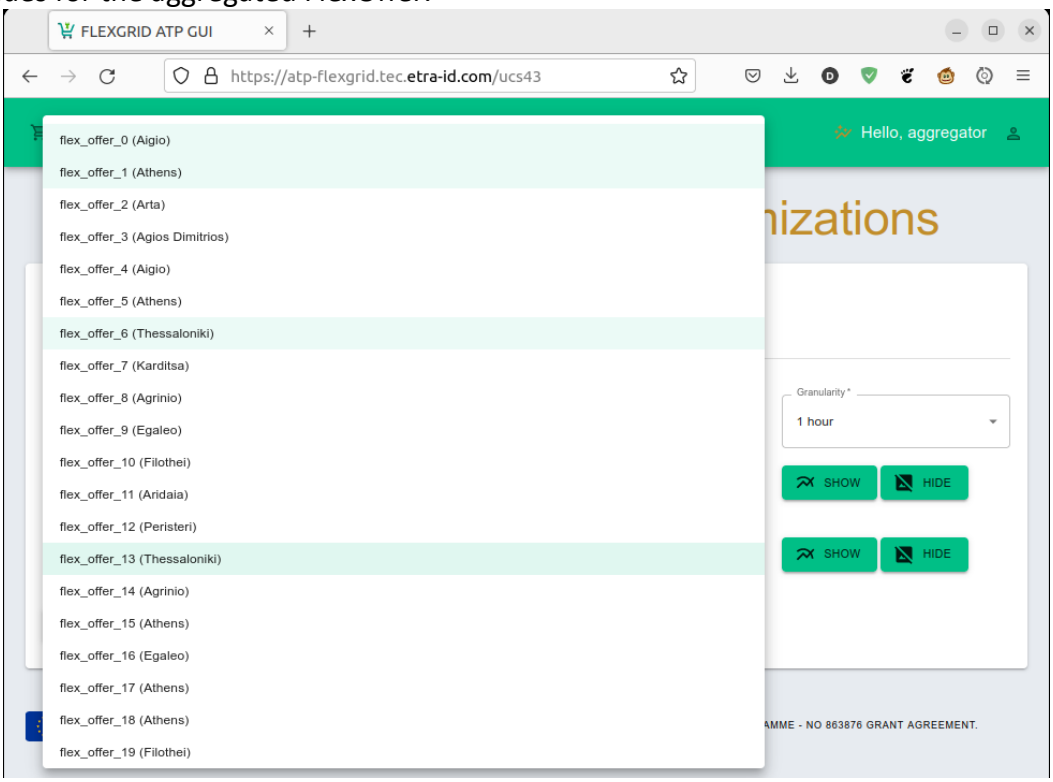


Figure 57: Selecting the individual FlexOffers that will be used for producing the aggregated FlexOffer

The final step is to select the individual FlexOffers that will be aggregated into a single FlexOffer. The figure above shows the aggregator user's interface for selecting the FlexOffers. The aggregator user is ready to select the "Execute Algorithm" button and waits a few seconds until the results are fetched back from the AFAT server. The following figure shows the output when the algorithm is still being executed:

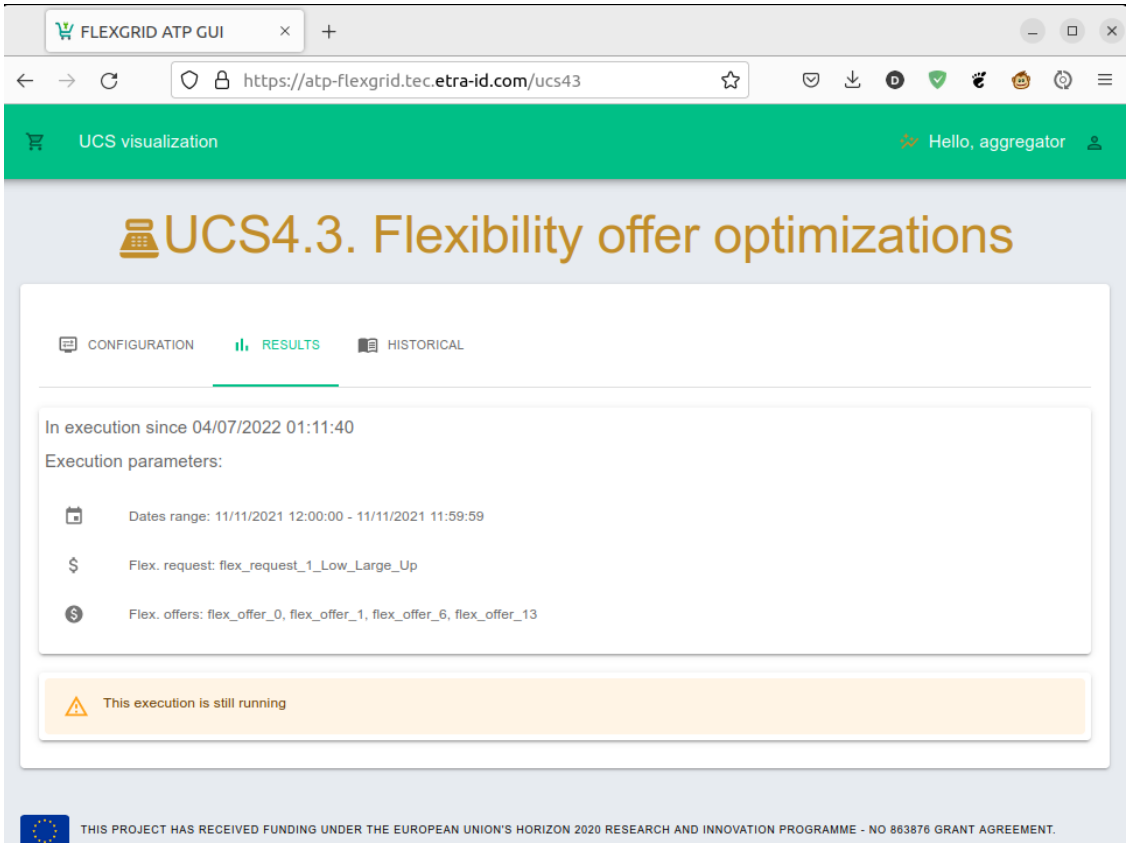


Figure 58: The output screen when the algorithm run is still in progress.

After the execution of the algorithm is completed, the following results appear:

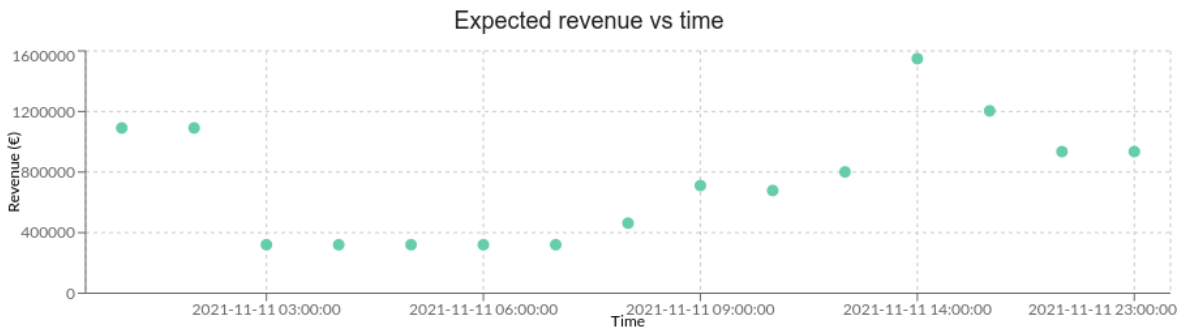


Figure 59: Expected revenue vs time

The first graph depicts the expected revenues for each of the timestamps that are considered in the execution of the algorithm. The expected revenue is calculated by first aggregating the individual FlexOffers to a single aggregated FlexOffer. Then, the resulting FlexOffer is matched with the FlexRequest, just as they would be cleared in a flexibility market. For each timestamp, a price and a quantity value are calculated. Thus, the aggregator may run different scenarios with different FlexOffers and FlexRequests and see how the different FlexOffers should be clustered in order to produce the optimal expected revenues for the aggregator's business.

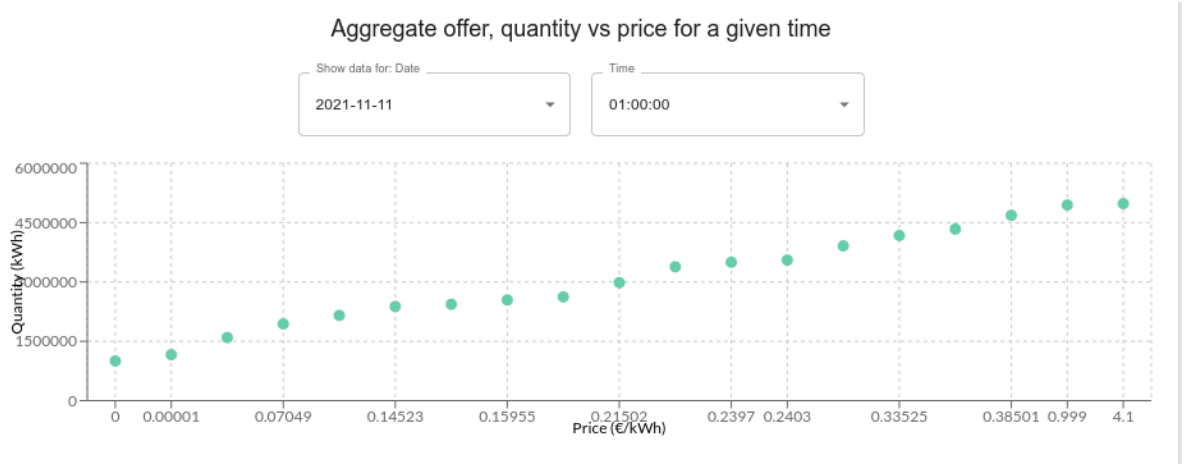


Figure 60: Aggregated FlexOffer, quantity vs. price for a given timeslot (e.g. 01:00 am)

In the figure above, the aggregated FlexOffer is depicted, for a given timestamp. The aggregator user may select from the dropdown menus, which timestamp s/he is interested in viewing, and the resulting piece-wise linear bid curve will appear. We can see that the FlexOffer for each timestamp will be increasing, meaning that as the price increases, the aggregator is willing to provide more flexibility to the flexibility market/system.

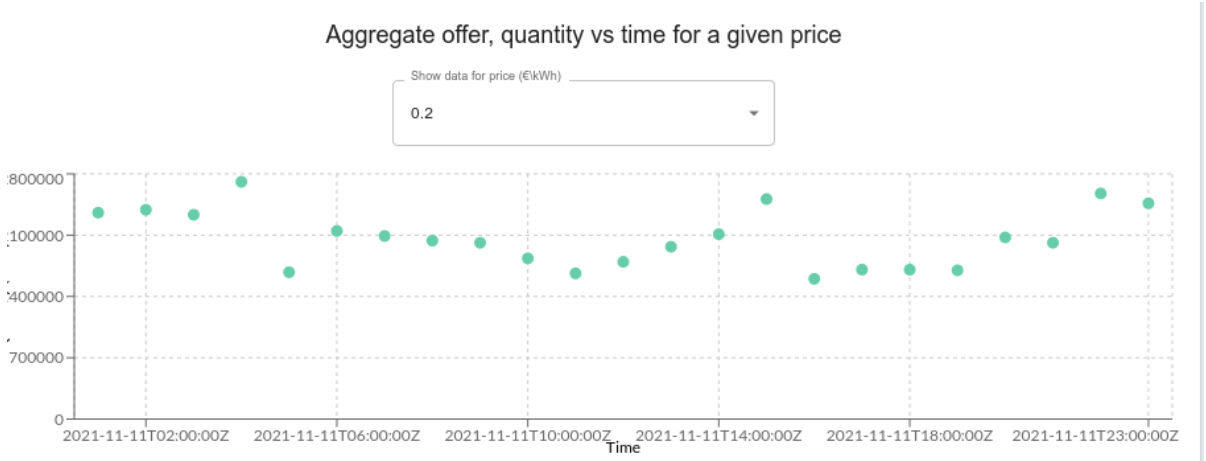


Figure 61: Aggregated FlexOffer, quantity vs. time for a given price (e.g. 0.20 euros/kWh)

The figure above depicts the aggregated FlexOffer, but this time the axes are the timestamp and the quantity. The aggregator user may select the price point s/he is interested in inspecting, and then the available quantity for each timestamp at that price point is depicted. We can see that during the day the available flexibility changes with time, depending on the offers that have been included in the aggregated FlexOffer.

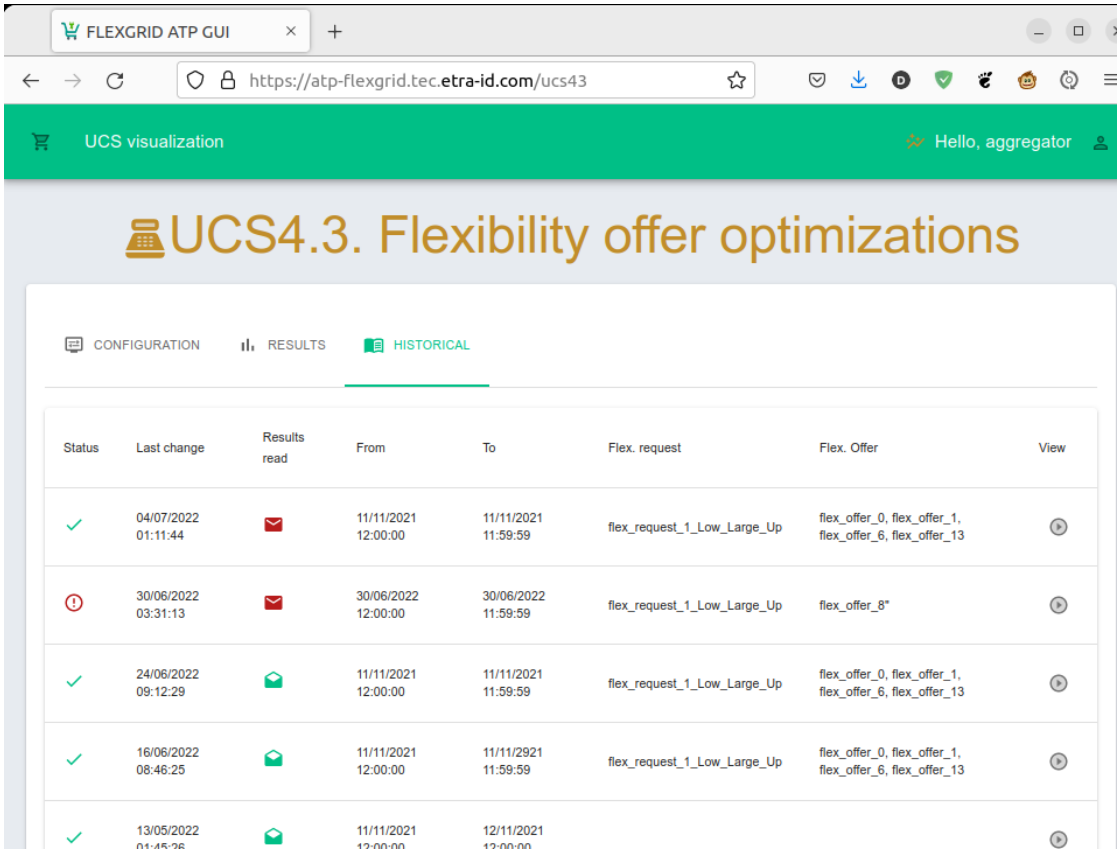


Figure 62: The aggregator user can visualize and compare all past results via “Historical” tab

Finally, by selecting “Historical” tab, the aggregator may view the previous algorithm executions, in order to be able to easily compare the results with different input parameters.

5.3 FMCT GUIs and Results

5.3.1 UCS 1.1 - DLFM clearing for the active power product

Regarding UCS 1.1: the FMO user wants to efficiently clear (a set of) FlexRequests and FlexOffers for energy that maximize social welfare while considering network constraints. For doing so the FMO user should enter the ATP with its credentials. They will be available to interact with UCS 1.1 to provide configuration inputs, see results and previous simulations (historical data).

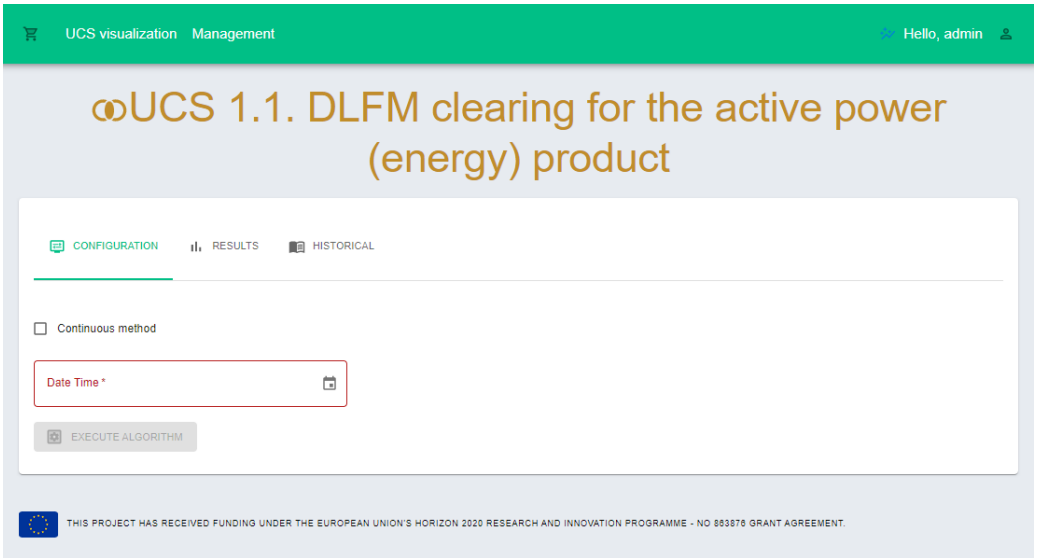


Figure 63 The FMO user selects UCS 1.1 service and fills in the input parameters via "Configuration" tab

The FMO user is able to clear the market under full consideration of network constraints, i.e., including line ratings, reactive power limits, and voltage bounds. Moreover, the active participation of the DSO and ESPs is considered with a continuous market setup.

As most of the needed information for clearing the market is related with the grid configuration, the ATP only allows the FMO user to select the date and to do the clearing and also the possibility to clear it based on continuous market clearing method. Internally the algorithm running includes the selected date and match the bids (locally included on the algorithm running locally). The results that the FMO user can see for the n time selected five different KPIs are:

- Social Welfare
- Procurement Cost
- Volume of Flexibility
- Energy-not served (ENS)
- The amount of curtailments

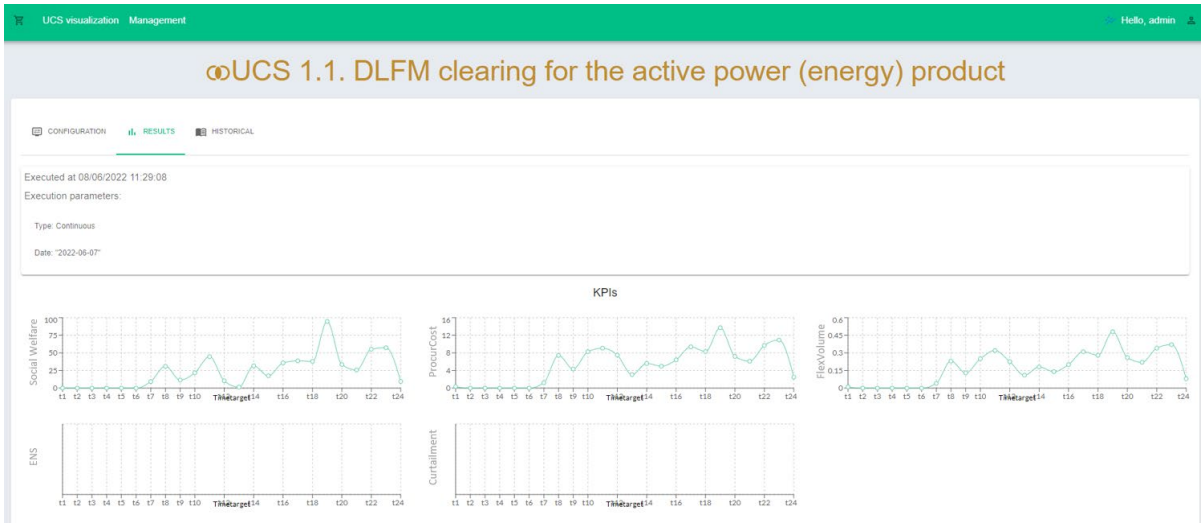


Figure 64 FMO KPI results after matching bids

Additionally to the KPIs, the main FMCT outcome is the price of energy per node and per time step and also the accepted bids per node including respective information such as the price, volume and direction (see figure below).
The FMO then informs the DSO of the accepted FlexRequests and the respective FSPs of their accepted FlexOffers. The price is set by the FlexOfffer in the pay-as-bid clearing.

Price DLFM

Node id	Type	Direction	Timetarget	Price (€)
n12	P	Down	t1	33
n1	P	Up	t10	298
n10	P	Down	t10	42
n11	P	Down	t10	22
n16	P	Down	t10	31
n18	P	Down	t10	39
n25	P	Down	t10	42
n30	P	Up	t10	277
n7	P	Down	t10	43
n1	P	Down	t11	41
n11	P	Up	t11	297
n14	P	Up	t11	261
n2	P	Up	t11	297
n29	P	Down	t11	41
n30	P	Up	t11	25

Figure 65 DLFM results to up and down energy

Accepted bids

Node id	Condition	Type	Direction	Volume	Price (€)	Timetarget	Timestamp	DispatchChange	Block	bid
n30	Unconditional	P	Up	0.02	30	t14			No	Offer
n21	Unconditional	P	Up	0.02	279	t14			No	Request
n33	Unconditional	P	Down	0.02	31	t14			No	Offer
n1	Unconditional	P	Down	0.02	38	t14			No	Request
n33	Unconditional	P	Down	0.05	37	t11			No	Offer
n1	Unconditional	P	Down	0.05	41	t11			No	Request
n6	Unconditional	P	Down	0.03	30	t11			No	Offer
n1	Unconditional	P	Down	0.03	41	t11			No	Request
n29	Unconditional	P	Up	0.02	28	t12			7896	Offer
n15	Unconditional	P	Up	0.02	270	t12			No	Request
n29	Unconditional	P	Down	0.02	28	t11			7896	Offer
n1	Unconditional	P	Down	0.02	41	t11			No	Request
n3	Unconditional	P	Up	0.04	31	t19			No	Offer
n9	Unconditional	P	Up	0.04	291	t19			No	Request
n31	Unconditional	P	Up	0.04	36	t22			No	Offer

Figure 66 DLFM accepted bids

5.3.2 UCS 1.2 and UCS 1.3 - DLFM clearing for the active and reactive power reserve

With the UCS 1.2 and UCS 1.3, the FMO user wants to efficiently clear (a set of) FlexRequests and FlexOffers for active and reactive power reserve that maximize social welfare, while considering network constraints. The task of the FMCT in UCS 1.2 and UCS 1.3 is to identify flexibility offers with feasible price that match the physical limits of the DN, and to maximizing social welfare within the given network constraints. The UCS 1.2 can be considered one special case of UCS 1.3 that is why the ATP provide the same results in terms of price and accepted bids despite of resulting from both UCS separately (for configuration, results and historical) also.

Similarly with UCS 1.1, most of the needed information for clearing the market is related to the grid configuration, so the ATP only requires the FMO user to select the date and to do the clearing based on the continuous market clearing method (Figure 67 and Figure 68).

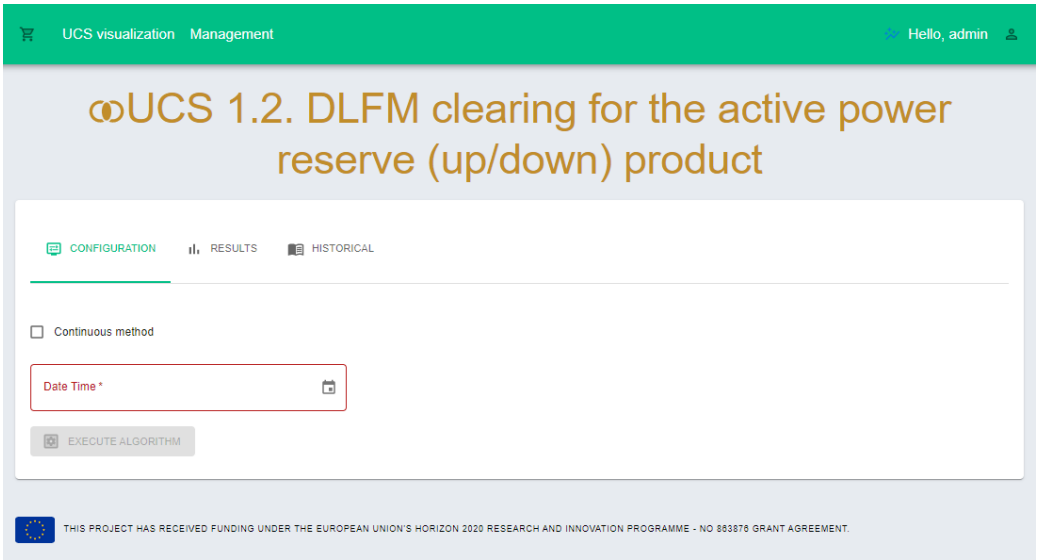


Figure 67 The FMO user selects UCS 1.2 service and fills in the input parameters via "Configuration" tab

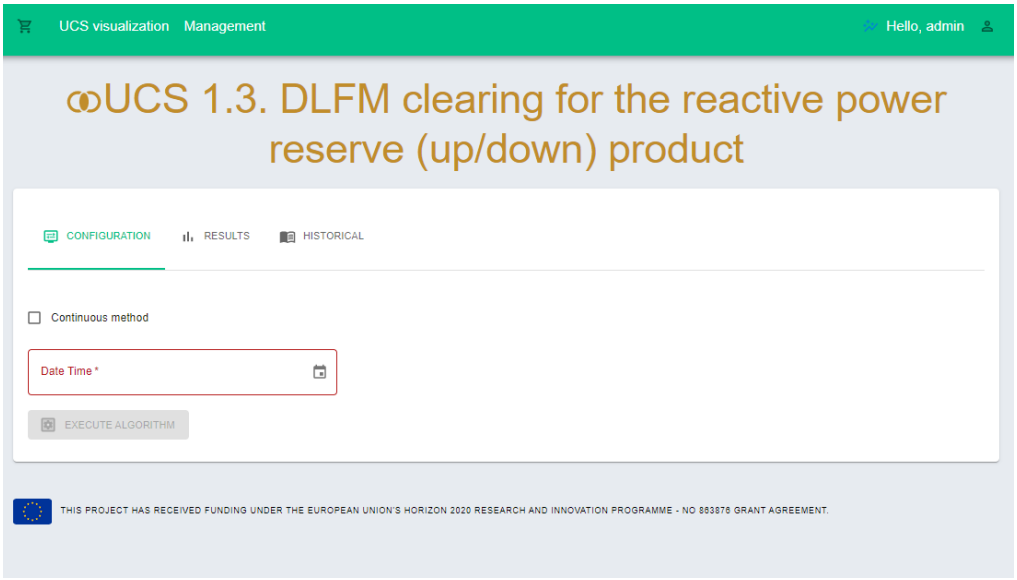


Figure 68 The FMO user selects UCS 1.3 service and fills in the input parameters via "Configuration" tab

Technically, the market clearing is an hourly clearing, so it is possible to only show one hour or 24 next hours as in UCS 1.1. To visualize the results shown on both UCS 1.2 and UCS 1.3, here, three-time steps are shown in the GUIs and the interaction with the ATP works properly.

Figure 69 shows the KPIs results for: Social Welfare, Procurement Cost, Volume of Flexibility, Energy-not served (ENS) and the amount of curtailment as in UCS 1.1 that gives the FMO the relevant information to work.
Additionally, it is possible to see on the result tab the DLFM price for each node and the accepted bids per node with the main relevant information for the FMO (see figures below).

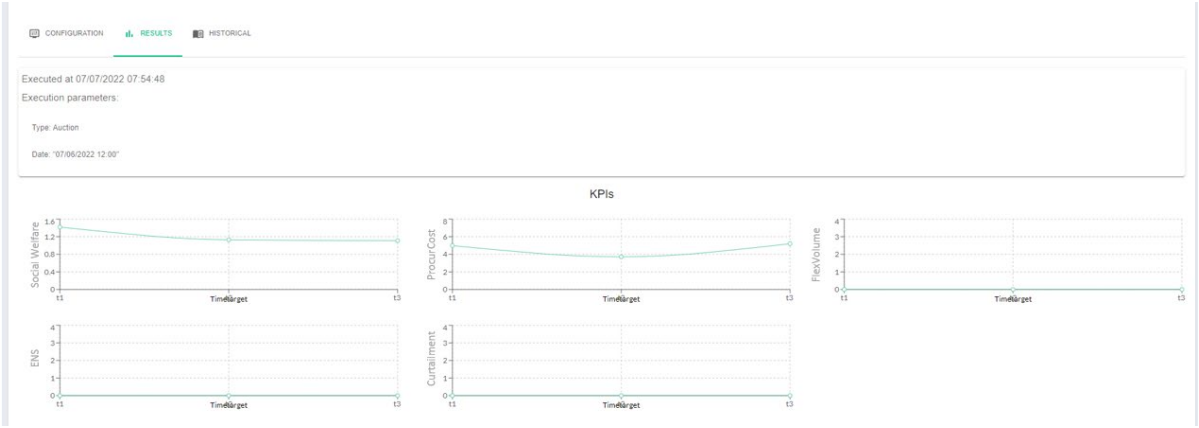


Figure 69 FMO KPI results after mating bids for the 3 hours selected

Price DLFM

Node id	Type	Direction	Timetarget	Price (€)
n10	P	Down	t1	32
n10	P	Up	t1	32
n12	Q	Down	t1	32
n13	P	Down	t1	36
n13	P	Up	t1	36
n15	P	Up	t1	32
n15	Q	Down	t1	32
n2	Q	Up	t1	39
n7	P	Down	t1	34
n7	P	Up	t1	34
n8	P	Up	t1	31
n10	P	Down	t2	32
n10	P	Up	t2	32
n11	P	Up	t2	30
n12	P	Down	t2	32
n12	Q	Down	t2	32

Figure 70 DLFM results to up and down energy (active or reactive)

Accepted bids

Node id	Condition	Type	Direction	Volume	Price (€)	Timetarget	Timestamp	DispatchChange	Block	bid
n10	Unconditional	P	Down	0.02	41	t1	2021-09-21 12:01:23	0.02	No	Request
n4	Unconditional	P	Down	0.01	44	t1	2021-09-21 12:05:23	0.01	No	Request
n15	Unconditional	P	Up	0.02	41	t1	2021-09-21 12:15:23	0.02	No	Request
n13	Unconditional	P	Up	0.03	42	t1	2021-09-21 12:21:23	0.03	No	Request
n5	Unconditional	P	Down	0.01	40	t1	2021-09-21 12:25:23	0.01	No	Request
n10	Unconditional	P	Up	0.03	37	t1	2021-09-21 12:34:23	0.03	No	Request
n13	Unconditional	Q	Down	0.04	39	t1	2021-09-21 13:13:23	0.04	No	Request
n4	Unconditional	P	Down	0.01	44	t2	2021-09-21 12:03:23	0.01	No	Request
n15	Unconditional	P	Up	0.02	41	t2	2021-09-21 12:04:23	0.02	No	Request
n13	Unconditional	P	Up	0.03	42	t2	2021-09-21 12:28:23	0.03	No	Request
n10	Unconditional	P	Down	0.02	41	t2	2021-09-21 12:40:23	0.02	No	Request
n5	Unconditional	P	Down	0.01	40	t2	2021-09-21 12:46:23	0.01	No	Request

Figure 71 DLFM accepted bids per node and type of bid

6 FLEXGRID ATP service installation

This section explains the basic steps that an S/W developer should follow in order to be able to download, install and configure a FLEXGRID service in its own system. Due to the modular-by-design FLEXGRID ATP architecture, each FLEXGRID ATP service can be offered as a stand-alone service or as a part of a bunch of services according to the end customer's business preferences. FLEXGRID ATP deployment is based on open-source S/W tools and thus a basic (DEMO) version of FLEXGRID services are publicly available in the project's GitHub area (https://github.com/FlexGrid). It should be noted that the final version of the FLEXGRID ATP (containing advanced functionalities tailored to specific customer segments) will be kept in closed access according to the FLEXGRID's exploitation plan. The ATP access and the general steps on how to work with the GitHub is here presented to better illustrate the main steps that can be extrapolated to the other services to run all the available algorithms of the ATP.

6.1 ATP access and APIs use

The Automated Trading Platform is available online following the open science approach and allowing during the whole life of the project and to be ready to test it after termination.

Link: https://atp-flexgrid.tec.etra-id.com/

Login: demo user will be available and share with strategical stakeholders to prove the ATP

User role	Username	email	Password
Aggregator	aggregator	aggregator1@flexgrid.etraid.com	aggregator
DSO	dso	dso@dso.flexgrid.etraid.es	dso
ESP	esp	esp@esp.flexgrid.etraid.com	esp
FMO	fmo	fmo@fmo.flexgrid.etraid.com	fmo

Registration: The registration process can be done via the owner of the ATP (i.e administrative user). The owner can add and remove any new users so every client for the ATP modules should contact the owner to ask them for premises. At this stage of the project and until its end the owner of the platform is ETRA, as developer, and after the lifetime of the project ETRA will have the administrative role until an interested stakeholder agrees on having the relevant license for the ATP.

All the APIs use swagger editor

6.2 Service installation

6.2.1 Step 1: Design of the API using swagger editor

As a first step, one should use the online tool at https://editor.swagger.io/ to create the API definition. As a starting point, one can use the swagger file that is provided by FLEXGRID project. Thus, the developer can copy and paste the source code into the swagger online tool and then adapt the API to meet the goals of the developer's endpoint.

Example: the source code for AFAT service 2 is available here: https://github.com/FlexGrid/AFAT-service-2-manage-b2c-flexibility-market
In order to make changes to this project, it is possible to edit the swagger definition file (.yml) available for the different services.

- UCS 1.1, 1.2, 1.3-Continuous and auction clearing: https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS1.X.yaml
- UCS 2.1 -Minimize ESP's OPEX: https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS2.1.yaml
- UCS 2.2 - Minimize ESP's CAPEX: https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS2.2.yaml
- UCS 2.3 – Stack revenues maximization: https://stacked-revenues-api.flexgrid-project.eu/swagger/stacked-revenues.yml
- UCS 4.1 – FelxRequest dispatch optimization: https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS4.1.yaml
- UCS 4.2 – Manage a B2C flexibility market: https://pricing-api.flexgrid-project.eu/swagger/pricing.yml
- UCS 4.3 – Create and aggregated FlexOffer: https://flex-offers-api.flexgrid-project.eu/swagger/flex_offers.yml
- UCS 4.4 – Market and Price forecasting: https://resources.demo.etra-id.com/SWAGGER/flexgrid/UCS4.4_PV_production.yaml

6.2.2 Step 2: Connect to FLEXGRID Central Database

The second step is for the developer's API endpoint to connect to the FLEXGRID Central Database authorization system. For this reason, the developer should contact the FLEXGRID ATP administrator and ask for client credentials for testing his/her API. Then, the developer will obtain: i) a client id, ii) a username, and iii) a password. After that, the developer will be able to obtain a token by posting a curl request and get a respective response, in which there is the token that the developer needs to test his/her API service. For more technical details about the connection to the FLEXGRID central database, please check https://github.com/FlexGrid/AFAT-service-2-manage-b2c-flexibility-market
You may also run the algorithm using local data (see below), so this step may be omitted.
If you want to try a local copy of the central database, you can use the repository at https://github.com/FlexGrid/central-db-api, and then set the .env file with the appropriate CENTRAL_DB_BASE_URL value, such as http://localhost:5000, and also set the credentials set in your local copy of the database.

6.2.3 Step 3: Deploy, test and run your server locally

The third step is to deploy, test and run the server locally (i.e. localhost). This server contains all the source code (i.e. written in python language) that needs to be executed in order for the FLEXGRID service to be delivered. Example: For FST service 3 The developer should visit https://editor.swagger.io/, and from the top menu select "Generate Server". A zip file will be downloaded by the browser. Then, this file should be unzipped and saved in a directory. Python3 and pip3 applications should be downloaded and installed in the local PC in order for the FLEXGRID UCS 2.3 algorithm to be executed properly.

6.2.4 Step 4: Deploy the FLEXGRID application on your server

Now that the server is up and running, the next step is to deploy the FLEXGRID application (e.g. UCS 4.2) on this server. This procedure is based on: https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uswgiand-nginx-on-ubuntu-18-04. It assumes that the operating system is Ubuntu 18.04, and outside facing web server is nginx. We use uWSGi as the application server for our application, which will only be accessible through nginx. Several technical steps should be followed based on the FLEXGRID developer's manual and can be summarized as follows: i) Install required packages, ii) Clone the project repository and create a virtual environment for python, iii) Activate venv, iv) Add files for usgi deployment, v) Test that the server can start with wsgi, vi) Deactivate the venv, vii) Create a uwsgi configuration file following the technical instructions, viii) Add system configuration to automatically run the service, ix) Create a nginx configuration and the relevant certificates with certbot, x) Validate that the FLEXGRID UCS 4.2 service is working properly.
Deployment has been tested with nginx with uwsgi, using systemd to start and enable the api service and the celery program for the background tasks.
Sample configuration files for these services may be found at the ./config/ subdirectory, but changes are needed to set your own url, file paths, and user names.

6.2.5 Step 5: Implement the algorithm

The algorithm that has been imported for the FLEXGRID UCS services can be found in the GitHub area. In order to integrate the algorithm, one can add the repository as a git submodule. Then, one can call the submodule code from the controller that was generated by codegen. It should be noted that the basic version of FLEXGRID algorithms is publicly available for further reuse, testing and exploitation by every interested party. In case an interested individual or legal entity wants to use the full version of FLEXGRID services, then this service should be purchased according to the FLEXGRID's exploitation plan (see D8.3).

6.2.6 Step 6: Using external data or data to further test and validate the algorithm operation

To test with external data, you may provide an external data source using your own REST API server. The implementation of the server used in the flexgrid deployment is available at https://github.com/FlexGrid/central-db-api. To configure the new data source, edit the .env file per the sample.
In any case, the central db adapted should follow the schema defined in https://db.flexgrid-project.eu/swagger/.

7 Conclusions

This Deliverable presents the work carried out until M33 of T6.2 “Design of APIs and S/W Development” and T6.3 “GUIs and integration activities”. It covers all the integration process and GUIs development during the second phase of the project and after the algorithms from WP3, WP4 and WP5 were finalized.

Via the ATP GUIs, a proof-of-concept solution is demonstrated through a “Minimum Value Product - MVP” approach that is integrated in the FLEXGRID ATP (i.e. mathematical models and algorithms). All services and features were developed within the Project`s WPs giving thus the opportunity to the various market actors to enhance their business cases as well as optimize their economical expenditures. From WP3, stable versions of the algorithms of UCS 4.1, UCS 4.2 and UCS 4.3 are fully integrated in ATP. Via these AFAT services, the aggregator is able to manage its portfolio by performing “what-if” simulations and testing under different business scenarios. Regarding WP4 algorithms of UCS 2.1, UCS 2.2 and UCS 2.3, these are now integrated within the ATP and ESP user can easily run “what-if simulation scenarios” in order to create optimal strategies improving thus its market position. As of WP5, through the ATP, the FMO user can run and manage algorithms for continuously clearing a distribution level energy (UCS 1.1), active power reserve (UCS 1.2) and reactive power reserve market (UCS 1.3).

With the modular-by-design approach adopted by FLEXGRID, it is possible to cover a broad range of services for various stakeholders allowing for the integration with current and available tools on the market and to optimize the management of the assets. Additionally, thanks to integration per use case scenario, the work performed on WP7 with the pilots is easy to manage and control making possible for one actor to take the best decision over their installation and available assets. Ultimately, the use of a central data base to manage results allows for future integration and interoperability with a high scalability potential, while ensuring privacy and data security.