REPORT ON DELIVERABLE 5.4.2

DAIAD@commons and DAIAD@utility software

PROJECT NUMBER: 619186
START DATE OF PROJECT: 01/03/2014
DURATION: 42 months

DAIAD is a research project funded by European Commission's 7th Framework Programme.

Dissemination Level	Public
Due Date of Deliverable	Month 40, 30/06/2017
Actual Submission Date	30/06/2017
Work Package	WP5 Big Water Data Analysis
Tasks	5.5 DAIAD@commons 5.6 DAIAD@utility
Type	Prototype
Approval Status	Submitted for approval
Version	1.0
Number of Pages	44
Filename	D5.4.1_DAIAD@commons_utility_software.pdf

Abstract

This report presents an overview of the Prototype Deliverable D5.4.2 "DAIAD@commons and DAIAD@utility software for trials", which includes all software developed in the context of WP5 packaged into two web applications supporting the corresponding major deployment scenarios of DAIAD. DAIAD@commons implements the '*bottom-up*' consumer-driven deployment and adoption of DAIAD, in which the only data source available is amphiro b1 data. In contrast, DAIAD@utility implements the '*top-down*' utility-driven deployment and adoption of DAIAD, in which the data source includes a SWM and (optionally) an amphiro b1. Both applications are implemented over the big data management engine developed in Task 5.1 (*D5.1.2 'Updated Big Water Data Management Engine'*), integrate the corresponding analytics functionality developed in Tasks 5.2, 5.3 (*D5.2.2 'Updated Consumption Analytics and Forecasting Engine'*), and apply the appropriate UI elements of T5.4 (*D5.3.2 'Knowledge Discovery Workbench'*).

History

version	date	reason	revised by
0.1	27/03/2017	First draft	Yannis Kouvaras
0.2	30/05/2017	Contributions in various sections	Yannis Kouvaras, Michalis Alexakis, Stelios Manousopoulos, Nikos Karagiannakis, Giorgos Hatzigeorgakidis, Spiros Athanasiou
0.5	02/07/2017	Updated figures and descriptions	Yannis Kouvaras, Stelios Manousopoulos
0.8	20/02/2017	Multiple revisions and updates	Yannis Kouvaras, Michalis Alexakis, Stelios Manousopoulos, Nikos Karagiannakis Spiros Athanasiou
1.0	30/06/2017	Final version	Spiros Athanasiou

Author list

organization	name	contact information
ATHENA RC	Spiros Athanasiou	spathan@imis.athena-innovation.gr
ATHENA RC	Yannis Kouvaras	jkouvar @imis.athena-innovation.gr
ATHENA RC	Stelios Manousopoulos	smanousop@imis.athena-innovation.gr
ATHENA RC	Michalis Alexakis	alexakis@imis.athena-innovation.gr
ATHENA RC	Giorgos Hatzigeorgakids	ghatzi@imis.athena-innovation.gr
ATHENA RC	Nikos Karagiannakis	nkara@imis.athena-innovation.gr
Waterwise	Aaron Burton	aaronburton@waterwise.org.uk
AMAEM	Ignacio Casals del Busto	ignacio.casals@aguasdealicante.es
AMAEM	Alejandro Garcia Monteagudo	alejandro.garcia@aguasdealicante.es

Executive Summary

This report presents an overview of the Prototype Deliverable D5.4.2 "DAIAD@commons and DAIAD@utility software", which includes all software developed in the context of WP5 packaged into two web applications supporting the corresponding major deployment scenarios of DAIAD. DAIAD@commons implements the '*bottom-up'* consumer-driven deployment and adoption of DAIAD, in which the only data source available is amphiro b1 data. In contrast, DAIAD@utility implements the '*top-down'* utility-driven deployment of DAIAD, in which the data sources include a SWM and (optionally) an amphiro b1.

The two applications share a common codebase, with the appropriate functionalities available to end-users defined during deployment. On a high-level, DAIAD@commons can be considered as a subset of DAIAD@utility, with the latter providing full access to SWM-based analytics, as well as dedicated analysis and forecasting services for water utilities.

DAIAD@commons and DAIAD@utility have been developed using as building blocks all software components and elements developed in the context of WP5. In particular:

- D5.1.2 Updated Big Water Data Management Engine. It handles the entire data lifecycle, providing scalable data management and querying services. An overview of the Data Engine is provided in the report for Prototype Deliverable D5.1.2.
- D5.2.2 Updated Consumption Analytics and Forecasting Engine. It provides the full suite of analytics and forecasting services, executed within the Big Water Data Management Engine. Essentially, it provides the 'business logic' of both applications, executing the appropriate analysis algorithms and returning the results to the UI elements. An overview of the Engine is provided in the Report for Prototype Deliverable D5.2.2.
- D5.3.2 Updated Knowledge Discovery Workbench. It provides all UI elements for presenting and invoking the analysis results of D5.2.2. In this manner, we hide the complexity of the underlying analytics and data engine, providing a coherent and simple to use interface for users. An overview of the Workbench is provided in the Report for Prototype Deliverable D5.3.2.

In Section 1, we provide an overview of the two applications, elaborate on their architecture, and present the major libraries and frameworks applied for their development.

In Section 2, we present a walkthrough of the various features offered by DAIAD@utility, which as discussed is a superset of DAIAD@commons (*bottom-up, only amphiro b1 data*). The walkthrough serves to demonstrate how all software components and elements developed in the context of WP5 are assembled to deliver the two web applications.

Abbreviations and Acronyms

API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
AOP	Aspect Oriented Programming
APK	Android application package
BT	Bluetooth
CI	Continuous Integration
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
DOM	Document Object Model
DTO	Data Transfer Object
JSON	JavaScript Object Notation
MR	MapReduce
MVC	Model View Controller
MVP	Minimum Viable Product
OGC	Open Geospatial Consortium
ORM	Object Relational Mapper
RERO	Release Early, Release Often
REST	Representational State Transfer
RF	Radio Frequency
RPC	Remote Procedure Call
SPA	Single Page Application
SWM	Smart Water Meter
UI	User Interface

Table of Contents

1. Implementation 8

1.1. Overview 8

1.1.1. DAIAD@utility Web Application 9

1.1.2. DAIAD@commons Web Application 11

1.2. Development 12

1.2.1. Development Environment 12

1.2.2. Tools 12

1.3. Architecture 13

1.3.1. Overview 13

1.3.2. Application Patterns and Design 15

1.3.3. DAIAD Services 16

1.4. Libraries and Frameworks 21

1.4.1. Spring 21

1.4.2. Apache Log4j2 23

1.4.3. Joda-Time 23

1.4.4. ICU - International Components for Unicode 23

1.4.5. JTS Topology Suite 23

1.4.6. Hibernate 23

1.4.7. Jadira Framework 24

1.4.8. Apache POI 24

1.4.9. GeoTools 24

1.4.10. Flyway 24

1.4.11. JSch – Java Secure Channel 24

1.4.12. React 25

1.4.13. Redux 25
1.4.14. React-Router-Redux 25
1.4.15. React-Bootstrap 25
1.4.16. Moment 25
1.4.17. ECharts 26
2. DAIAD@utility walkthrough 27
2.1. Layout and Menu 27
2.2. Dashboard 28
2.2.1. Counters 28
2.2.2. Charts 28
2.2.3. Maps 29
2.2.4. Tables 30
2.3. Analytics 30
2.3.1. Charts 30
2.3.2. Maps 31
2.3.3. Reports 32
2.3.4. Favorites 33
2.4. Forecasting 34
2.5. Consumers 34
2.6. Engagement 36
2.7. Trial Management 41
2.8. Support 42
2.9. Job Management 43
2.10. Logging 44

1. Implementation

1.1. Overview

DAIAD@commons and DAIAD@utility are the web applications that integrate all functionality developed in the context of WP5. On a high-level, DAIAD@commons is a subset of DAIAD@utility, with the latter providing full access to SWM-based analytics, as well as dedicated analysis and forecasting services for water utilities. Specifically:

- DAIAD@commons implements the 'bottom-up' consumer-driven deployment and adoption of DAIAD, in which the only data source available is amphiro b1 data.
- DAIAD@utility implements the 'top-down' utility driven deployment of DAIAD, in which the data sources include a SWM and (optionally) an amphiro b1.

The two applications share a common codebase, with the appropriate functionalities available to end-users defined during deployment, as well as the authorization and API call parameters (Figure 1).

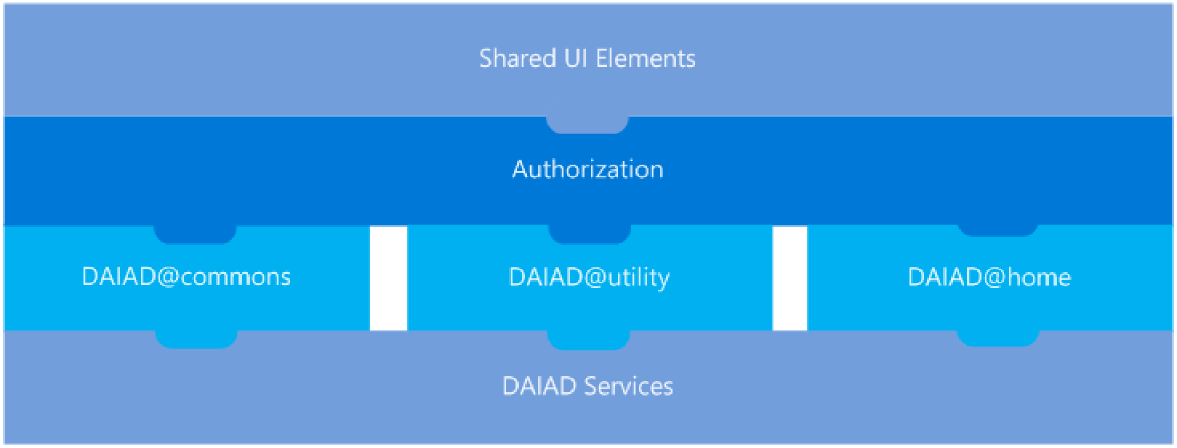


Figure 1: DAIAD web applications shared architecture

In addition, DAIAD@commons and DAIAD@utility have been developed using as building blocks all software components and elements developed in the context of WP5 (see Figure 2). In particular:

- D5.1.2 Updated Big Water Data Management Engine. It handles the entire data lifecycle, providing scalable data management and querying services. An overview of the Data Engine is provided in the report for Prototype Deliverable D5.1.2.
- D5.2.2 Updated Consumption Analytics and Forecasting Engine. It provides the full suite of analytics and forecasting services, executed within the Big Water Data Management Engine. Essentially, it provides the 'business logic' of both applications, executing the appropriate analysis algorithms and returning the results to the UI elements. An overview of the Engine is provided in the Report for Prototype Deliverable D5.2.2.

- D5.3.2 Updated Knowledge Discovery Workbench. It provides all UI elements for presenting and invoking the analysis results of D5.2.2. In this manner, we hide the complexity of the underlying analytics and data engine, providing a coherent and simple to use interface for users. An overview of the Workbench is provided in the Report for Prototype Deliverable D5.3.2.

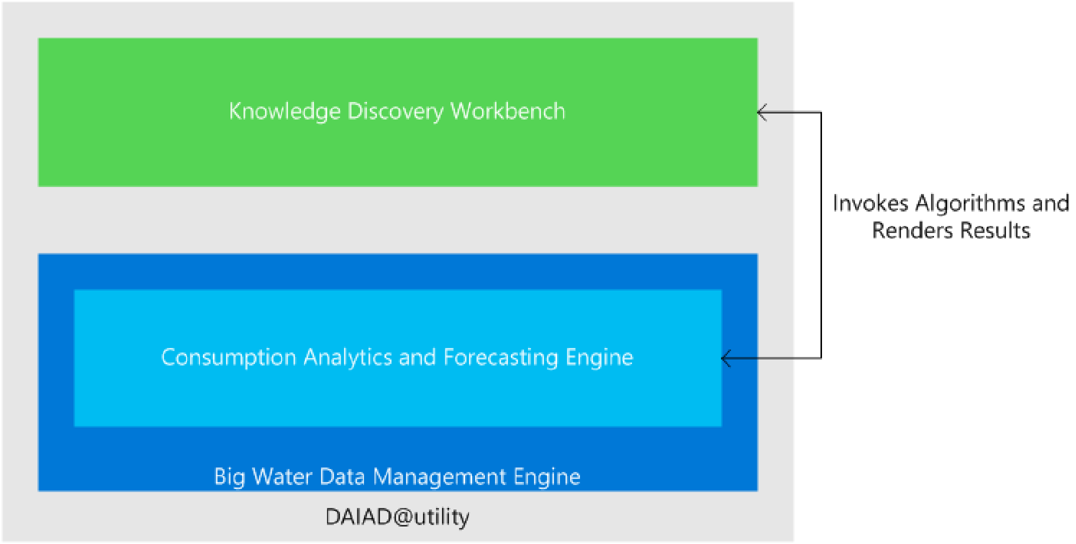

Knowledge Discovery Workbench
Invokes Algorithms and Renders Results
Consumption Analytics and Forecasting Engine
Big Water Data Management Engine
DAIAD@utility

Figure 2: Software components and elements developed in the context of WP5

1.1.1. DAIAD@utility Web Application

DAIAD@utility web application, referenced as the 'application' in this sub-section for brevity, provides the UI elements required for performing tasks related to a single utility including:

- Presentation of general information about recent user activity.
- Analysis of water consumption data from smart water meter and one or more amphiro b1 devices.
- Water consumption forecasting for a utility, a group of users or a single user.
- Browsing users and their data with the ability to visualize data from multiple users or groups of users
- Schedule, execute and monitor jobs.
- Performing administration tasks such as viewing application log files, managing trial users, uploading raw data manually or sending messages to multiple users

The application is designed as a Single Page Application that submits requests to DAIAD Services using the Action API described in section 1.3.3.1. The application architecture is based on React and Redux JavaScript libraries. User interface is composed by React components that utilize Redux unidirectional data flow for maintaining application state. The application architecture is displayed in Figure 3.

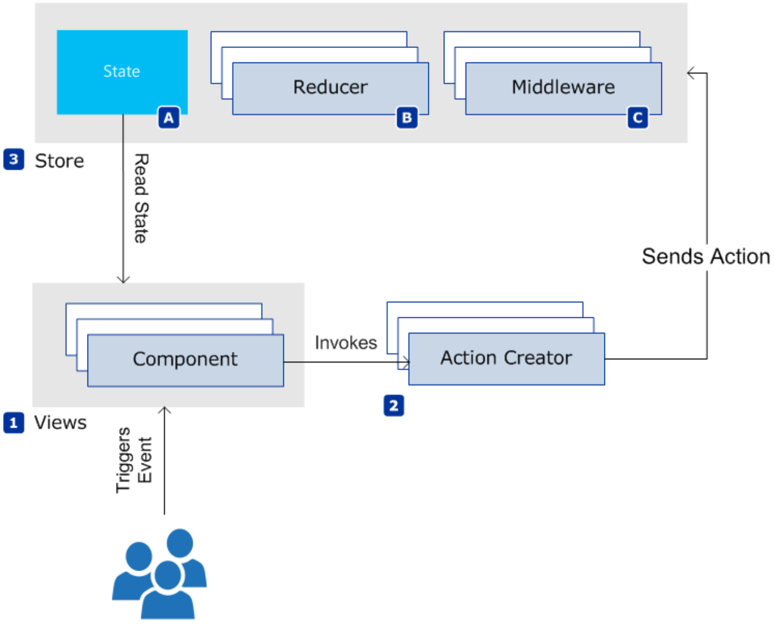


Figure 3: DAIAD@utility web application architecture based on React and Redux

The names of the components of the architecture are in sync with the Redux terminology since the whole application is based on it. In the following, we enumerate each of the components and provide references to the source code.

- **Store**: The core component of the application is the Store. Store holds all the application data and acts as the single source of truth. All components retrieve state data from the Store which in turn is used for driving the rendering process. Store data should not be confused with component internal state e.g., the Timeline control stores internally a timeout interval for triggering periodic updates. This piece of information does not belong to the application state. Yet, the status of whether the timeline should be periodically updated or not is stored inside the Store. Store is initialized by the store module[1].
- **Component**: Components are implemented using React and are separated to presentational and container components. The presentational components are used only for rendering, and they usually have no state and no knowledge of the Store. Such components are the Map[2] and Timeline[3] components. On the contrary, container components have knowledge of Redux and interact with the Store. The application uses three container components, namely, the Root[4], App[5] and ContentRoot[6]. The Root is the top-level component of the UI tree hierarchy and is responsible for initializing the Store and making it available to child components. The App provides localization support for the

---

[1] https://github.com/DAIAD/web-client-utility/blob/master/store/configureStore.js

[2] https://github.com/DAIAD/react-leaflet-wrapper/blob/master/src/Map.js

[3] https://github.com/DAIAD/web-client-utility/blob/master/components/Timeline.js

[4] https://github.com/DAIAD/web-client-utility/blob/master/containers/Root.js

[5] https://github.com/DAIAD/web-client-utility/blob/master/containers/App.js

[6] https://github.com/DAIAD/web-client-utility/blob/master/containers/ContentRoot.js

whole application and renders the ContentRoot as the top level visible component. Finally, ContentRoot controls the general application layout such as navigation menu and content placement.

- **Action Creators** and **Actions**: Users interact with the application using events. When an event is triggered on a component, an appropriate factory method named Action Creator is called and an Action object is generated. Action objects contain information about which action is requested, augmented with any action specific information such as date-time interval information, smart water meter Id or user Id. The action is propagated to the store which is processed by the Middleware and Reducer components. The result of the action is the transformation of the Store which triggers an update to the UI.
- **Middleware**: Middleware components are optional extensions that intercept Actions before reaching the Reducers discussed next. The application is using the React Router Redux[7] middleware in order to handle user navigation and render the appropriate components. The routing configuration is set in the routing[8] module.
- **Reducers**: Reducers are components that update Store state according to a received Action. Since the state has a hierarchical structure as a JavaScript object, reducer components are also organized in a similar hierarchical structure where every reducer partially updates the Store. All the reducers are configured in the reducer module[9]. Depending on the complexity of the application logic, some reducers are divided even further into child reducers e.g. the Map[10] reducer.

1.1.2. DAIAD@commons Web Application

DAIAD@commons web application provides the UI elements required for performing tasks related to amphiro b1 data management including:

- Presentation of general information about recent user activity.
- Simple analysis of water consumption data from amphiro b1 devices such as aggregation.
- Viewing user information.
- Browsing user data for one or more amphiro b1 devices.
- Data visualization from multiple users or groups of users.

The application design and architecture is the same as with DAIAD@utility (see previous sub-section for details). DAIAD@commons can be deployed either as a standalone application or side-by-side with DAIAD@utility web application.

---

[7] https://github.com/reactjs/react-router-redux

[8] https://github.com/DAIAD/web-client-utility/blob/master/routing/routes.js

[9] https://github.com/DAIAD/web-client-utility/tree/master/reducers

[10] https://github.com/DAIAD/web-client-utility/blob/master/reducers/map.js

1.2. Development

In order to streamline the development process of the DAIAD software, a common development environment for all developers has been setup. The development environment includes the installed operating system, the Java Runtime Environment used for executing the application, several server applications like database stores, and the development tools used for authoring source code. Using a common environment makes application behavior more predictable across all deployment sites. Moreover, using the same versions for all development tools and servers, guarantees that the contributed code from each member of the development team is compatible with the existing code base.

1.2.1. Development Environment

All development takes place on Ubuntu 14.04.4 LTS Linux distribution. This is the same version of operating system used for deploying the application on the production site. Having development and production systems using the same version of operating system makes the deployment process less error prone and minimizes any incompatibility issues. For the application execution, the Java Runtime Environment (JRE) 7u95-2.6.4 is used.

Except for the operating system, the development environment includes several server applications that are required for executing the DAIAD software. The required application servers along with their version are:

- PostgreSQL relation database 9.3.11
- PostGIS, the PostgreSQL spatial extension 2.1.2
- Hadoop with YARN 2.6.0
- HBASE NoSQL database 1.0.0. At the deployment site, HBASE 0.98.10 is used but the two versions are compatible.
- Tomcat Java Servlet Container 8.0.33

1.2.2. Tools

In order to accelerate the development process, ease the management of code artifacts, and simplify software building and testing, the following development tools are used:

- Eclipse (Luna)[11] Integrated Development Environment (IDE): Eclipse is the most popular open source IDE used for developing applications using several programming languages; for DAIAD, Java and JavaScript is used. Eclipse has an open architecture that offers great extensibility by mans of plugin and extension implementations. In DAIAD project, several Eclipse plugins are used including GitHub and Maven integration plugins. Moreover, the Web Tools Platform[12] extension is used for enabling Eclipse to develop Web and enterprise (J2EE) applications.
- Apache Maven[13]: Maven is a comprehensive project management tool that supports a rich feature set such as building code, managing dependencies, packaging artifacts, generating code documentation

---

[11] https://eclipse.org/luna/

[12] http://www.eclipse.org/webtools/

[13] https://maven.apache.org/

and projects reports, creating a project's site and enabling communication among development team members to name a few. Maven can be extended by using plugins that apply specific actions at selected project build lifecycle phases. All Maven actions are driven by the Project Object Model (POM) that fully describes a project. The corresponding POM file for the DAIAD project can be found at https://github.com/DAIAD/home-web/blob/master/pom.xml.

- Git[14]: Git is an open source distributed version control system used for managing the source code, as well as other assets such as documentation, of the DAIAD project. In DAIAD, Git coordinates the updates to the central code repository by using pull requests, thus allowing code reviewing and better code integration.

1.3. Architecture

1.3.1. Overview

In Deliverable D1.2 'DAIAD Requirements and Architecture', the architecture of the DAIAD system is described in detail. An overview of the DAIAD architecture is also illustrated in Figure 4. The components are separated in two sections. The lower section contains all external components that are used by the DAIAD system. The top section contains all the components developed as part of the DAIAD project.

The main goal of this section is to describe the DAIAD@utility and DAIAD@commons web applications in addition to the DAIAD services on which the former components rely for implementing their functionality. However, a short description is given for every component for completeness.

---

[14] https://git-scm.com/

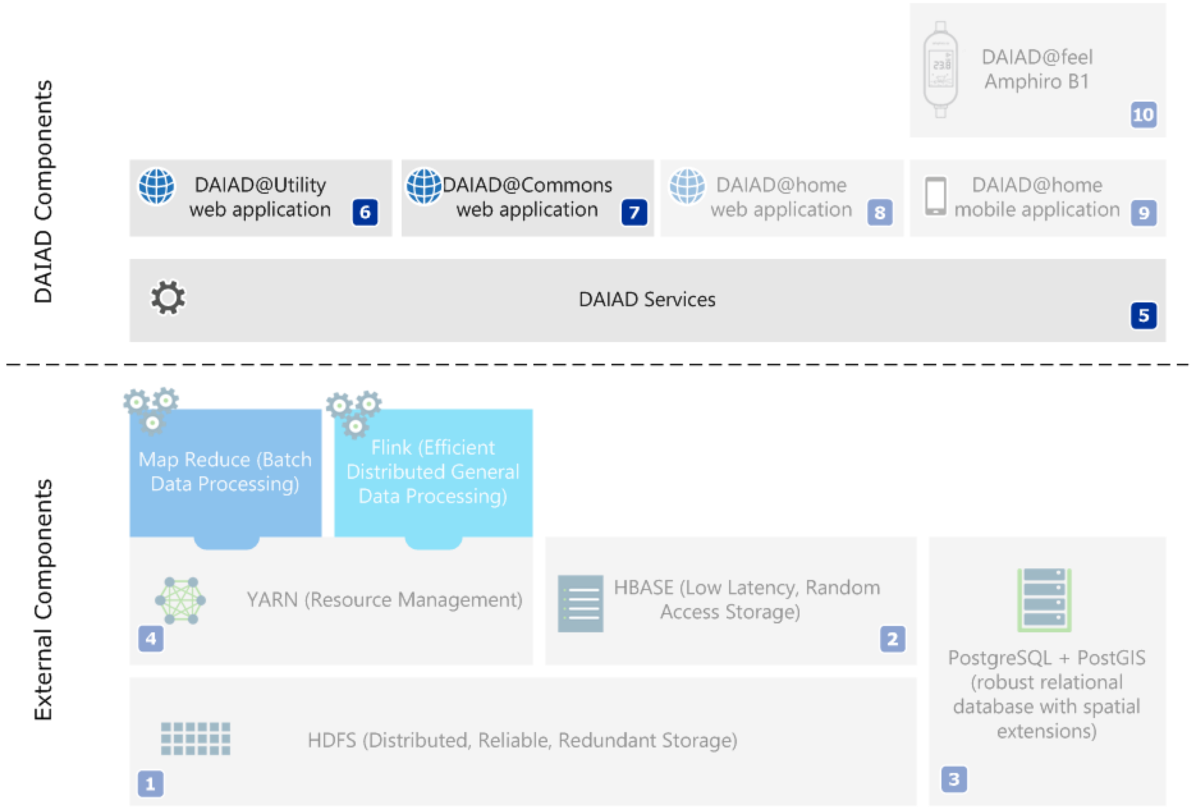

DAIAD Components
External Components
DAIAD@feel
Amphiro B1
10
DAIAD@Utility
web application
6
DAIAD@Commons
web application
7
DAIAD@home
web application
8
DAIAD@home
mobile application
9
DAIAD Services
5
Map Reduce (Batch Data Processing)
Flink (Efficient Distributed General Data Processing)
YARN (Resource Management)
4
HBASE (Low Latency, Random Access Storage)
2
PostgreSQL + PostGIS (robust relational database with spatial extensions)
3
HDFS (Distributed, Reliable, Redundant Storage)
1

Figure 4: DAIAD architecture overview

The major software and hardware components that are part of the DAIAD System are the following:

- **Hadoop Distributed File System (HDFS)** provides reliable and redundant storage for all components located higher in the software stack, and is the central piece of DAIAD's data management scheme. HDFS is used for storing HBase tables, intermediate analysis results and exported data. It also acts as an accessible active archive for storing legacy and backup data.
- **HBase** is a NoSQL database that offers low latency, random data access over HDFS. HBase stores all measurement data generated by sensor devices (amphiro b1) and smart water meters. Moreover, HBase stores and indexes data generated by the analytical processing of measurement data.
- **PostgreSQL** is a relational database used for storing frequently changing data, like community memberships, user profiles and preferences, application metadata, cluster management information such as job execution metadata, etc. In general, PostgreSQL is used for storing tables that contain only a few hundreds of thousands of records. The goal of using a different store for this type of data is to minimize the delete operations on HBase. Moreover, PostGIS, a spatial extension of PostgreSQL, is used for managing spatial information associated with users and devices.
- The **YARN** resource manager coordinates the execution of jobs from the Hadoop MapReduce and Flink data processing frameworks. Both frameworks can use any of the available storage platforms, namely, HDFS, HBase and PostgreSQL, as a data source and as a data sink. More details about job implementation, scheduling and execution are presented in section 1.3.3.5.

All the aforementioned systems are standalone server applications that have been installed and configured for usage by other DAIAD applications developed in the context of the DAIAD project. On top of these systems a set of web services and applications has been developed:

- **DAIAD Services** software component is located on top of all database servers and data processing frameworks. It is the central component that orchestrates the invocation of analysis and forecasting algorithms, interacts with the big data management engine developed in Task 5.1 and provides data and associated metadata to DAIAD@home, DAIAD@commons, and DAIAD@utility web applications and services through suitable programmatic interfaces.
- **DAIAD@commons** provides the user interface required for exploring and analyzing consumption data generated from amphiro b1 devices, following the bottom-up deployment scenario of DAIAD (consumer-centric). It propagates user requests to DAIAD Services through programmatic interfaces and exposes its functionality through secured HTTP APIs in order to be consumed by third party applications and services.
- **DAIAD@utility** provides the user interface required for exploring and analyzing consumption data generated from smart water meters and amphiro b1 devices, following the top-down deployment scenario of DAIAD (utility-centric). It propagates user requests to DAIAD Services through programmatic interfaces and exposes its functionality through secured HTTP APIs in order to be consumed by third party applications and services.
- **DAIAD@home web application** allows users to have access to their water consumption data and perform simple analysis such as aggregation over variable time intervals. The web application complements the DAIAD@home mobile application and allows users to access all of their data without the need to download it locally to their mobile devices.
- **DAIAD@home mobile application** manages the whole lifecycle of consumption data at the household level. It is responsible for gathering data from DAIAD@feel sensors (amphiro b1), storing and managing consumption data, invoking algorithms for analyzing it, and implementing interventions for providing information and stimuli to consumers.

DAIAD@home, DAIAD@commons and DAIAD@utility compose the core of the software developed at the DAIAD project. The last component of the DAIAD system is the DAIAD sensor devices that are used for collecting water consumption data at the fixture level:

- **DAIAD@feel** is a hardware device that consists of a self-powered water flow sensor that collects data about water flow and water temperature at the fixture level. This sensor is integrated in the amphiro b1 device alongside with a LC display for presenting consumption information to the end user at the point of water extraction.

1.3.2. Application Patterns and Design

In the next sections, we provide implementation details for DAIAD@commons, DAIAD@utility and DAIAD Services software components. To facilitate understanding, the reader is invited to consider the following:

- DAIAD@commons, DAIAD@utility and DAIAD Services use additional external libraries and frameworks detailed in Section 1.4. During the presentation that follows, we frequently make references to these libraries and frameworks, since they affect the implementation details of each component.

- We apply the Model View Controller pattern (MVC) and the Single Page Application (SPA) web application design. This pattern and design are used extensively in the DAIAD implementation and they strongly influence the structure of the source code. A short explanation for each follows; a broader coverage of these topics is outside the scope of this document:
    - **Model View Controller (MVC)**. The goal of the Model View Controller pattern is to separate code responsibilities into three parts. The Model, which represents application domain data and logic, the View, which is responsible for the data presentation and the Controller, who receives user interactions and updates the Model appropriately. This separation increases code testability and also improves a developer team's productivity. Nowadays, there are many variants of the MVC pattern and each MVC framework may implement the pattern in different ways. For the DAIAD implementation we are using the Spring Framework and its corresponding MVC module.
    - **Single Page Applications (SPAs)** offer increased UI usability that is in par with desktop applications. In contrast to traditional web applications, a SPA application is initialized by loading only a single web page. After initialization, any additional resources such data or JavaScript code files are loaded dynamically on demand using Asynchronous JavaScript and XML (AJAX) requests. Moreover, client-side code is usually implemented using the MVC pattern or some variant of it.

1.3.3. DAIAD Services

DAIAD Services implement the core features of the DAIAD functionality. DAIAD services implement the MVC pattern based on the Spring Framework. As such, the DAIAD components should be thought mostly as loosely coupled components that extend the Spring Framework as depicted in Figure 5. The main components of DAIAD services are enumerated next.

1.3.3.1. Controllers

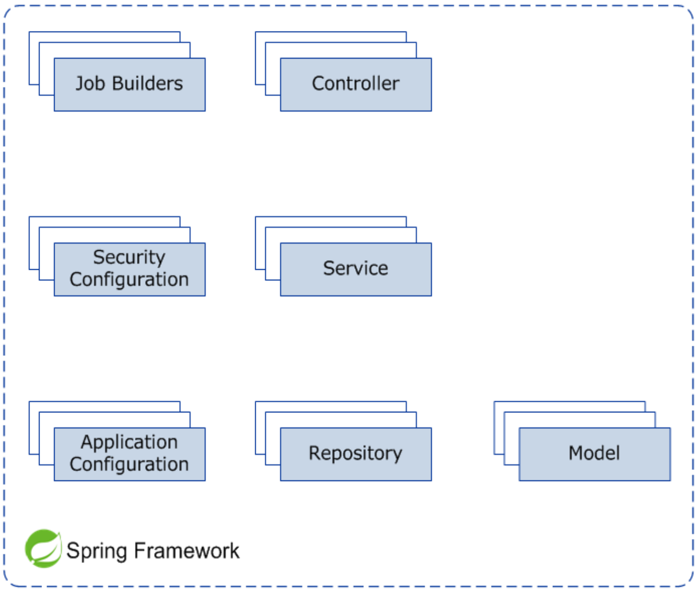


Figure 5: DAIAD components

DAIAD services controllers expose DAIAD functionality to clients. The communication is based on several HTTP APIs that exchange JSON formatted messages. The APIs and hence the corresponding controllers are separated into two main categories: the Action controllers and the HTTP API controllers.

The former are used for exchanging data between the DAIAD services and the DAIAD web applications. They are *stateful* controllers which require a session to be initialized before exchanging any messages. Session initialization is performed through authentication. These controllers do not support the Cross-Origin Resource Sharing (CORS) standard and hence their methods cannot be used by 3rd party applications. Moreover, they offer enhanced security features such as Cross-Site Request Forgery (CSRF) and Session Fixation protection. These latter features are available out of the box thanks to the Spring Security module. Action API is strongly coupled with the DAIAD services version; hence they do not have any specific version information embedded in their URLs. All Action API method URLs have the /action/ prefix. Details on the Action controllers can be found here.

In contrast, the HTTP API controllers are *stateless* and require authentication for every request. They support CORS and hence can be used for creating 3rd party applications that utilize the DAIAD services. Their main task is to exchange data with the DAIAD@home mobile application. All HTTP API controller methods are versioned and have their version embedded into their URLs along with the /api/ prefix e.g. /api/v1/. Details on the HTTP API controllers can be found here.

According to the MVC definition, controllers are responsible for updating the application model and executing application logic. Nevertheless, DAIAD services controllers do not interact directly with the model. Instead, services and repositories are used for encapsulating application logic and modifying the model.

The functionality provided by Action and HTTP API controllers may overlap. Moreover, the permission requirements for different controller methods may vary e.g. the scheduler controller actions require administrative permissions while the data query controller actions can be invoked by simple users. In addition, the same controller methods may be called with different permissions. In this case, the data accessible to the caller depends on the granted permissions. Detailed information about the Action and HTTP API controllers' methods can be found at the project API documentation pages. Similarly, information about the Action controllers' methods is available at project source code documentation pages.

1.3.3.2. Views

DAIAD web applications are implemented as single page applications. As a result, there are almost no views to be rendered at the server side. For every application, there is only a single view that is rendered when the client initializes the application and its sole purpose is to load all the required assets, such as JavaScript and CSS files, in order to bootstrap the application. After the initialization, the client only exchanges messages with the DAIAD services using the Action controllers.

1.3.3.3. Model

DAIAD services data model consists of two types of classes namely Simple Data Transfer Objects (DTOs) and Domain Objects. The DTOs are simple, transient objects that are used only for data shaping when creating controller response messages and for transferring data between components. In contrast, Domain Objects represent application entities with unique identifiers that are persisted using a relational database.

The DTO class definitions reside in the eu.data.web.model package hierarchy. Depending on their usage they are further organized into multiple sub-packages, e.g. job scheduling related DTO classes are defined in the eu.data.web.model.scheduling package. DTOs tend to have little, to almost none application logic.

The domain object definitions are located in the eu.data.web.domain package hierarchy. The domain objects are further separated into two categories, namely the administration and the application entities, which are also persisted to different database instances. The former represent system specific entities such as job scheduling information. The latter represent common application entities such as users, profiles and smart water meters. In contrast to the DTOs, which are characterized as an anemic model, domain objects encapsulate rich application logic such as persistence information, validation rules and entity dependencies. Most of this information is declared using annotations thanks to Spring Framework's Aspect Oriented Programming (AOP) features.

1.3.3.4. Repositories

DAIAD services store data into HBase (NoSQL) and PostgreSQL (relational) databases. Persistence to relation databases is transparently managed by the Hibernate Object Relational Mapper (ORM) which significantly simplifies data access. The application code does not interact directly with Hibernate, but instead the Java Persistence API is used. In contrast, data operations for HBase are manually implemented. In either case, all data access code is encapsulated in different classes named repositories.

The repositories publish convenient interfaces for executing the most common data access operations required by the DAIAD services such as creating a user, storing an amphiro b1 measurement or searching for a smart water meter device. All repositories are defined in the eu.daiad.web.repository package. The public methods of each repository are defined in a separate interface. At least one concrete implementation for each interface is provided. As a convention, the name of a repository interface implementation indicates its underlying data store e.g. the repository for managing user data is named JpaUserRepository.

1.3.3.5. Services

DAIAD services application logic is encapsulated in separate components named services. Services are organized into three categories (utility, entity and orchestration services), depending on how broad their functionality is. Services usually interact with repositories or even other services in order to perform their operations.

Utility services are focused in performing simple tasks that usually require little to none state. An example of a utility service is the ValidationUtils service which defines simple validation methods. Utility services are located in the eu.daiad.web.util package.

Entity and orchestration services are defined in the eu.daiad.web.service package hierarchy. An entity service usually operates on a specific entity type and optionally on its dependent entities. Such a service is the UserService which provides methods for creating a user and optionally registering a smart water meter to her account. An orchestration service uses a composition of other services in order to implement complex operations. DefaultMessageService is an example of an orchestration service that uses multiple repositories and services in order to generate user recommendation and alert messages.

1.3.3.6. Configuration

As mentioned earlier, DAIAD services are implemented using the Spring Framework. Spring Framework is automatically configured at runtime by declaring classes that are annotated with the Configuration annotation.

Moreover, in order to further simplify application configuration, Spring Boot is used on top of String Framework. Spring Boot is a strongly opinionated framework that opts for convention over configuration and relies heavily on auto-configuration features. Yet, it is also very extensible and allows developers to diverge from the defaults if required. In order to modify configuration options, developers can write their own auto-configuration class implementations, extend existing classes and override protected methods or implement appropriate interfaces that modify the behavior of existing auto-configuration classes.

In this section, we present the configuration classes that initialize the DAIAD services. Although configuration classes are not a part of the general architecture, they decisively affect the application runtime behavior and hence have been included in this document. DAIAD services configuration classes are declared in eu.daiad.web.configuration package. A short description of the most important configuration classes is shown below.

- AdminPersistenceConfig: Configures system specific domain objects and PostgreSQL schema migration.
- ApplicationPersistenceConfig: Configures application specific domain objects and PostgreSQL schema migration.
- BatchConfig: Initializes Spring Batch service for launching jobs.
- CustomBatchConfigurer: Initializes the data source and job repository for Spring Batch.
- LoggingConfigurer: Configures the logging system in order to log events to a relational database except for files.
- SchedulerConfig: Initializes the task scheduler that schedules job execution.
- SecurityConfig: Configures the application security settings including CORS support, CSRF protection, error handlers and default login and logout URLs.
- WebConfig: Adds URL mappings for default error views and enables modules for serializing date and spatial objects.

1.3.3.7. Security

In DAIAD services, web application security is based on Spring Security framework which by default is configured to use form based authentication. Since DAIAD web applications are Single Page Applications that communicate by exchanging JSON formatted messages, the security system had to be configured appropriately. Security configuration classes are contained in package eu.daiad.web.security. Next, we enumerate the most important extension and configuration classes for Spring Security.

- CsrfTokenResponseHeaderBindingFilter: Request processing filter for adding CSRF token to Action API requests.
- SimpleCORSFilter: Enables CORS support for HTTP API.
- CustomAccessDeniedHandler: Handles security exception when user session is expired.

- CustomAuthenticationProvider: Custom authentication provider that loads users from PostgreSQL database.
- CustomLoginUrlAuthenticationEntryPoint: Suppresses redirection to the default application login page for AJAX requests.
- RESTAuthenticationFailureHandler: Handles AJAX authentication request failures.
- RESTAuthenticationSuccessHandler: Handles AJAX authentication requests successes.
- RESTLogoutSuccessHandler: Handles AJAX logout requests.

1.3.3.8. Job Builders

Except for lightweight client requests, DAIAD services have to execute long running jobs such as computing forecasting data using the Flink Processing Framework, or performing water consumption data pre-aggregation using the Map Reduce Processing Framework. Jobs like these cannot be executed synchronously or locally at the application server hosting the DAIAD services. Instead, they are scheduled for asynchronous execution at the cluster by the scheduler service.

The scheduler service is only responsible for scheduling and initializing job execution and has no knowledge of a job's implementation specific details. At the same time, the scheduler service must have a way to pass external configuration parameters to a job without depending on the job custom implementation. To decouple the scheduler service from the job implementation, the IJobBuilder interface is defined. Every job that needs to be scheduled should implement this interface. A helper abstract implementation of the interface is also available in class BaseJobBuilder. IJobBuilder has only one method that needs to implement the builder pattern and returns a Job instance. The latter is being executed by the scheduler service using the Spring Batch infrastructure. Scheduler service is also able to pass parameters using the Spring Batch job context.

Job builder implementations reside in package eu.daiad.web.jobs.builder. Next, we enumerate all the available job builders.

- BudgetProcessingJobBuilder: Computes snapshots for active budgets monthly or on demand.
- CleanHBaseMeterDataJobBuilder: An ETL job that transfers all amphiro b1 data from tables with older schema version to the table with the most current schema version.
- CommandExecutor: Utility job for executing shell commands.
- ConsumptionClusterJobBuilder: Creates a job that clusters users according to their water consumption.
- CopyUserAmphiroDataJobBuilder: ETL job for copying amphiro b1 data between two users.
- CopyUserComparisonDataJobBuilder: ETL Job for copying comparison and ranking data between two users.
- CopyUserMeterDataJobBuilder: ETL Job for copying meter data between two users.
- DailyStatsCollectionJobBuilder: Creates a job for collecting daily statistics for the site such as number of registered users, number of assigned water meters etc.
- DataExportJobBuilder: An ETL job that exports amphiro b1, smart water meter and weather data for one or more utilities.

- ForecastingJobBuilder: Initializes and submits an Apache Flink job to a YARN cluster for computing water consumption forecasting data.
- HBaseStatusMonitorJobBuilder: Checks the HBase cluster status at regular intervals.
- MessageGeneratorJobBuilder: Creates a job for generating customized alerts and recommendations for all users based on their water consumption.
- MeterDataAggregationJobBuilder: Initializes and submits a Map Reduce (MR) job to a YARN cluster for computing meter data aggregates.
- MeterForecastingDataAggregationJobBuilder: Initializes and submits a Map Reduce (MR) job to a YARN cluster for computing aggregates for forecasting meter data.
- ReportGenerationJobBuilder: Generates monthly reports for individual users.
- SavingsPotentialJobBuilder: Initializes and submits an Apache Flink job to a YARN cluster for computing savings potential and Water IQ data.
- SendMailJobBuilder: Composes messages using an HTML template and sends mails to a list of recipients.
- SqlScriptExecutionJobBuilder: A generic job builder that creates a job for executing one or more SQL scripts to a PostgreSQL database. The actual scripts executed are defined using external parameters.
- UpdateAmphiroDataSchemaJobBuilder: Creates a job for running amphiro b1 data schema transformation.
- WaterMeterDataFileLoadJobBuilder: Loads smart water meter data from files stored at the local filesystem and imports it into HBase.
- WaterMeterDataSftpLoadJobBuilder: Initializes a job for downloading files with smart water meter readings from a SFTP server and importing their contents into HBase.
- WeatherServiceHarvesterJobBuilder: Harvests weather data from one or more weather services.

1.4. Libraries and Frameworks

In this section, we provide a short description and links to the most important external frameworks and libraries used by DAIAD@commons and DAIAD@utility.

1.4.1. Spring

Spring[15] is one of the most popular and mature application development frameworks for Java, featuring a vast ecosystem of projects for developing applications from the mobile to the enterprise and cloud. Spring's vertical tool stack handles almost any programmatic task like security, data access, transaction management, social service provider integration, etc. Yet, its modular architecture allows using only these features required by the solution being developed. Despite its complex architecture, Spring's extensive documentation and supreme extensibility features allows developers to easily configure Spring to their needs. In the next sections, the most important Spring modules used in DAIAD project are enumerated.

---

[15] https://spring.io/

1.4.1.1. Spring Framework

The Spring Framework[16] consists of several core modules that offer the basic building blocks for developing any kind of application. The features provided by the Spring Framework include:

- Inversion of Control (IoC) and Dependency Injection (DI) features for configuring the creation and managing the lifecycle of objects.
- Aspect-Oriented Programming (AOP) features for cleanly decoupling shared functionality such as transaction management and logging.
- Data Access features, including integration with object relation mapping APIs such Java Persistence API (JPA) or Hibernate. Moreover, Spring Framework allows the combination of these APIs with features such as declarative transaction management using AOP as mentioned earlier.
- Spring MVC web application and RESTful Web Service framework for easily building web application and services.

1.4.1.2. Spring Batch

Spring Batch[17] provides methods for configuring and executing jobs for processing large volumes of data. Jobs are organized in one or more processing units named steps. Data processing can be row based e.g., repeatedly processing chunks of rows of a database table or task based e.g. copying a set of files. Step execution flow can be controlled declaratively or programmatically. Steps can be executed sequentially, in parallel or even conditionally, which may result in specific steps not being processed at all. Spring Batch provides the infrastructure for launching and monitoring job execution and offers many features including logging/tracing, transaction management, job processing statistics, job restart, skip, and resource management.

1.4.1.3. Spring Security

Spring Security[18] provides the tools for implementing authentication and authorization to Java applications. For authentication, Spring Security supports several methods such as HTTP Basic Authentication, Form Based Authentication and OpenID Authentication for establishing and managing the application principal i.e. the user who can use the application. Once the application principal is set, role based security is used for controlling authorization. In addition to authentication and authorization, Spring Security offers many advanced features for web applications such as protection against attacks like session fixation, clickjacking, cross site request forgery, etc.

1.4.1.4. Spring Boot

When developing an application using the Spring Framework many configuration options must be set either using external configuration files or programmatically. Spring Boot[19] takes an opinionated view of the Spring platform by promoting convention over configuration and selecting sensible default values for most configuration settings. Thus, an application requires minimum configuration. At the same time, whenever the

---

[16] http://projects.spring.io/spring-framework/

[17] http://projects.spring.io/spring-batch/

[18] http://projects.spring.io/spring-security/

[19] http://projects.spring.io/spring-boot/

default values are not appropriate to the requirements of the application, they can easily be replaced with custom configuration options.

1.4.2. Apache Log4j2

Every application requires some sort of logging in order to support tracing and debugging. Apache Log4j2 [20] is a feature-rich and extensible logging framework that allows fine-grained message logging to several destinations e.g., console output, file system, remote servers, relational database. Apache Log4j2 can be configured programmatically or by using external configuration files. In the latter case, there is the ability to automatically detect changes and reconfigure logging policies during runtime. Moreover, during the reconfiguration process, no logging events are missed.

1.4.3. Joda-Time

Prior to Java 8, date and time class features were too restricted. Joda-Time[21] library provides an alternative to standard Java date and time classes, offering a more comprehensive feature set for creating and managing date-time objects, executing date-time calculations, formatting and parsing date-time expressions and performing time-zone calculations. From Java SE 8 onwards, java.time (JSR-310) is favored over Joda-Time.

1.4.4. ICU - International Components for Unicode

ICU[22] is a set of Java libraries that provide Unicode and Globalization support to software applications. ICU services support tasks such as code page conversion, string comparison according to specific language conventions, formatting date, time and currency values according to a specific locale and performing time calculations using different time zones.

1.4.5. JTS Topology Suite

JTS Topology Suite[23] implements a set of spatial operations over two-dimensional spatial predicates such as intersection, overlapping, distance etc. The goal of the project is to be used for developing applications that manipulate and query spatial datasets. JTS attempts to implement as accurately as possible the OpenGIS Simple Feature Specification[24] (SFS). Wherever SFS specification is unclear, JTS implementation attempts to use a reasonable and consistent alternative. Details about differences against SFS can be found at the official documentation.

1.4.6. Hibernate

Hibernate[25] is one of the most popular Java Object/Relational Mapping (ORM) frameworks that allows users to easily store and query data in relational databases. In addition to its own proprietary Java API, Hibernate also conforms to the Java Persistence API[26] (JPA) specification, thus allowing seamless integration to Java EE

---

[20] http://logging.apache.org/log4j/2.x/

[21] http://www.joda.org/joda-time/

[22] http://site.icu-project.org/

[23] http://docs.geotools.org/latest/userguide/library/jts/index.html

[24] http://www.opengeospatial.org/standards/sfs

[25] http://hibernate.org/

[26] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

application servers. Hibernate uses several techniques such as lazy loading and optimistic locking in order to achieve high performance.

1.4.7. Jadira Framework

Whenever Hibernate persists an object to a relation store, it needs to serialize object properties to the appropriate database specific data types. Out of the box, Hibernate supports the serialization of a restricted set of Java data types, including standard date-time objects. Adding support for new Java data types requires a custom implementation of the UserType[27] interface. The Jadira Framework provides ready to use implementations of the UserType interface for several data types including Joda types which are used extensively in DAIAD.

1.4.8. Apache POI

Apache POI[28] is a set of Java APIs for manipulating several file formats that are based on the Office Open XML standards (OOXML). The project supports Microsoft Word, Excel and PowerPoint file formats. Latest releases also added basic support for Microsoft Visio files.

1.4.9. GeoTools

GeoTools[29] is an open source Java library that provides tools for managing geospatial data. Internally it uses JTS Topology Suite for handling geometry instances. Moreover, it provides APIs for accessing spatial data in several file formats and spatial databases, transforming data between different coordinate reference systems and filtering data using spatial and non-spatial attributes. GeoTools is implemented in accordance to the Open Geospatial Consortium (OGC) standards.

1.4.10. Flyway

Flyway[30] is a database schema migration tool that favors convention over configuration. Database schema migration is implemented either by using SQL scripts written in the target database specific syntax, or using Java code. It supports multiple database vendors and can be invoked either as a standalone tool from the command-line or using the Java API from inside the application.

1.4.11. JSch – Java Secure Channel

JSch[31] is a Java implementation of SSH2[32]. It provides support for secure remote login, secure file transfer, and secure TCP/IP and X11 forwarding. It can automatically encrypt, authenticate, and compress transmitted data.

---

[27] https://docs.jboss.org/hibernate/orm/current/javadocs/org/hibernate/usertype/UserType.html

[28] https://poi.apache.org/

[29] http://www.geotools.org/

[30] https://flywaydb.org/

[31] http://www.jcraft.com/jsch/

[32] https://en.wikipedia.org/wiki/Secure_Shell#Version_2.x

1.4.12. React

React[33] is a JavaScript framework for building interactive User Interfaces. React can be thought as the View in the MVC pattern that allows users to build reusable UI components and promotes composition of existing ones. Each component maintains its internal state which controls the rendering process. Whenever state changes, only the parts of the Document Object Model (DOM) that are affected are updated. This is achieved by using a virtual representation of the DOM that efficiently detects changes to the actual DOM. The latter feature makes React interoperability with other UI libraries more challenging.

1.4.13. Redux

Redux[34] is a predictable state container for JavaScript applications. It mainly targets Single Page Applications where the state management becomes increasingly complicated as user interactions continuously update the application state. The state management becomes even harder since most interactions result in asynchronous requests and responses. Redux attempts to manage state in a predictable way by imposing specific restrictions on how and when state updates can occur. Redux makes a perfect match to React by deferring component state management to Redux.

1.4.14. React-Router-Redux

React Router Redux is a JavaScript library that allows an application implemented using React and Redux to keep the application state in sync with routing information. This feature is achieved by automatically storing additional data about the current URL inside the state. This information is then propagated to React which can in turn suitably change the component tree rendering process. If there is no need for syncing routing information and application state, a simpler implementation can be obtained by using the React Router[35] library. The latter provides support for keeping only the UI in sync with the URL.

1.4.15. React-Bootstrap

React-Bootstrap[36] is a library of reusable UI components for the React framework. It offers the look-and-feel of the Twitter Bootstrap[37] library using the React syntax, but has no dependencies on any 3rd party libraries like jQuery. React-Bootstrap offers a comprehensive list of UI components such as buttons, menus, form input controls, modal dialogs, tooltips, etc. All components can be used as provided or customized using CSS.

1.4.16. Moment

Moment[38] is a JavaScript library for managing date time values that creates a wrapper for the JavaScript standard Date[39] object. It provides an extensive API for parsing, manipulating, comparing and formatting dates. Moment also supports duration objects and internationalization.

---

[33] https://facebook.github.io/react/

[34] http://redux.js.org/

[35] https://github.com/reactjs/react-router

[36] https://react-bootstrap.github.io/

[37] http://getbootstrap.com/

[38] http://momentjs.com/

[39] https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Date

1.4.17. ECharts

ECharts[40] is a comprehensive JavaScript charting library for building interactive charts. It is based on a lightweight rendering framework and supports several chart types including line, column, scatter, pie, radar, candlestick, chord, gauge, funnel and map charts. Moreover, individual charts can be composed to create more complex data representations. ECharts is highly optimized for handling hundreds of thousands of data points, making it a seamless solution for big data analysis.

---

[40] https://ecomfe.github.io/echarts/index-en.html

2. DAIAD@utility walkthrough

In this section, we present a walkthrough of the various features offered by DAIAD@utility, which as discussed is a superset of DAIAD@commons (bottom-up, only amphiro b1 data). The walkthrough serves to demonstrate how all software components and elements developed in the context of WP5 are assembled to deliver the two web applications.

2.1. Layout and Menu

First, we present the DAIAD@utility web application default template. The user interface consists of three main sections, a header, a sidebar menu and a main data content section as illustrated in Figure 6.

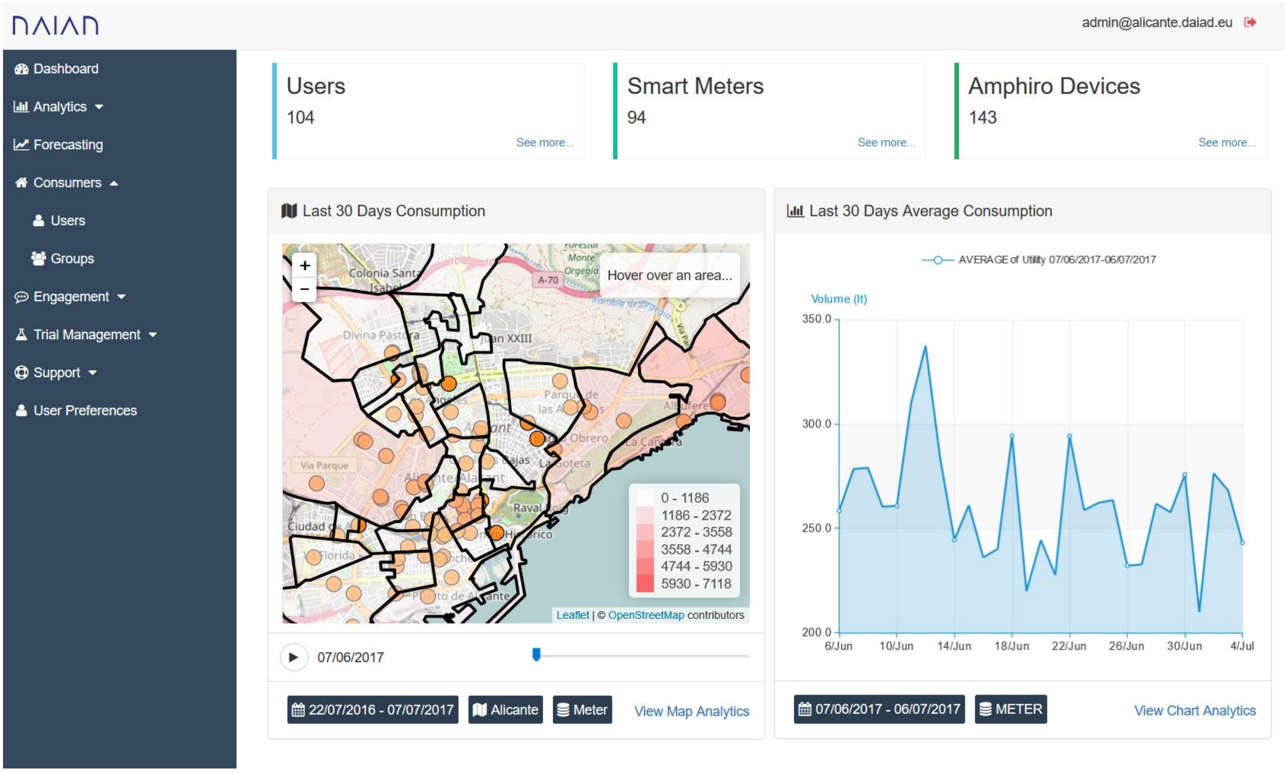


Figure 6: DAIAD@utility web application default layout

The header section contains the DAIAD logo along with actions or links that need to be accessible by the user at any given time, independently from the current page navigation state.

The sidebar menu allows a user to navigate easily between the main application views. In the case that several views offer functionality that shares common features, the corresponding menu items are grouped in collections that are visually represented as collapsible menus. For instance, Users and Groups are grouped under the Consumers menu item.

Finally, the main data content section is used for rendering the currently selected view.

2.2. Dashboard

The dashboard is the default view that is rendered the first time the user successfully logins to the application and provides a quick view of the current system status. Several types of components such as metric counters, charts and maps are used for visualizing data. The location of UI components is configurable and users are allowed to customize the dashboard to their preferences by both resizing and moving components. Moreover, for charts and maps components, users can configure the data selection by setting appropriate filters like time interval and granularity, spatial constraints, consumer selection etc.

Next, we describe in more detail the data visualization components, which have been implemented using the UI elements of D5.3.2 Updated Knowledge Discovery Workbench.

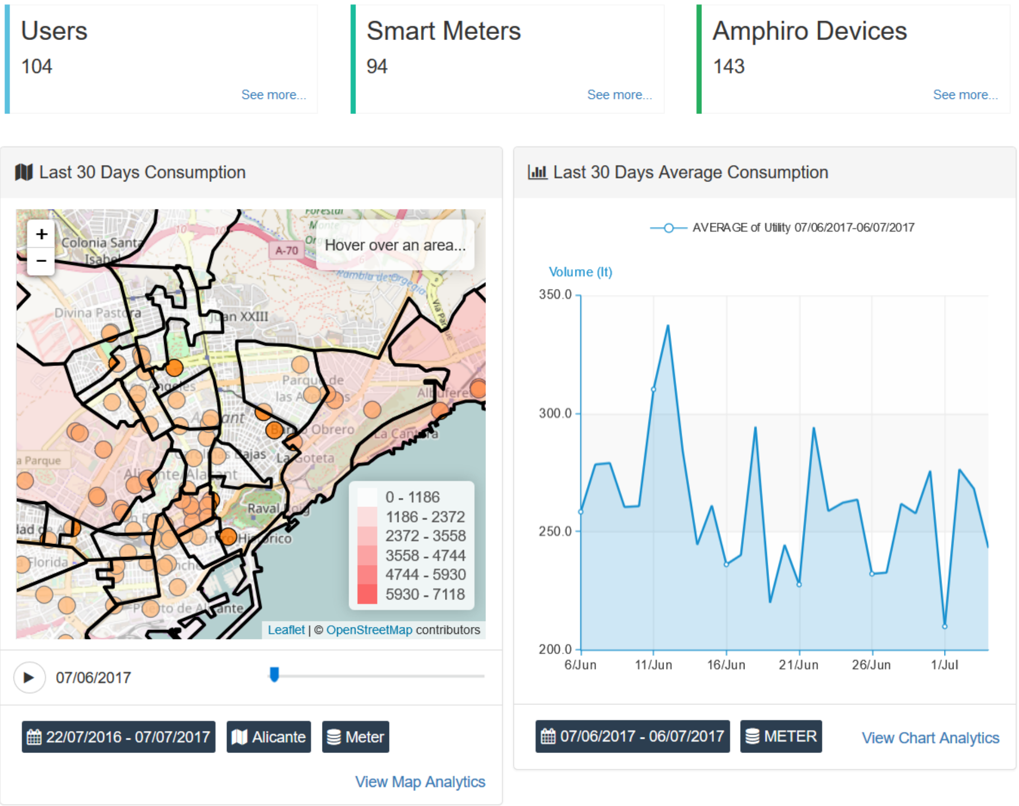


Figure 7: Dashboard view

2.2.1. Counters

Counters are simple visualization components that represent a single numerical value. Optionally, the variance of the rendered value and a text description can be supplied. For values that aggregate detailed data, a link may also exist to the underlying data. For instance, the users' counter in our example a link allows users to navigate to the Users & Groups sub section.

2.2.2. Charts

Chart is one of the most prominent components used in DAIAD@utility. An example of a chart with four data series is displayed in Figure 8. A chart may contain one or more data series and one or two different Y axes.

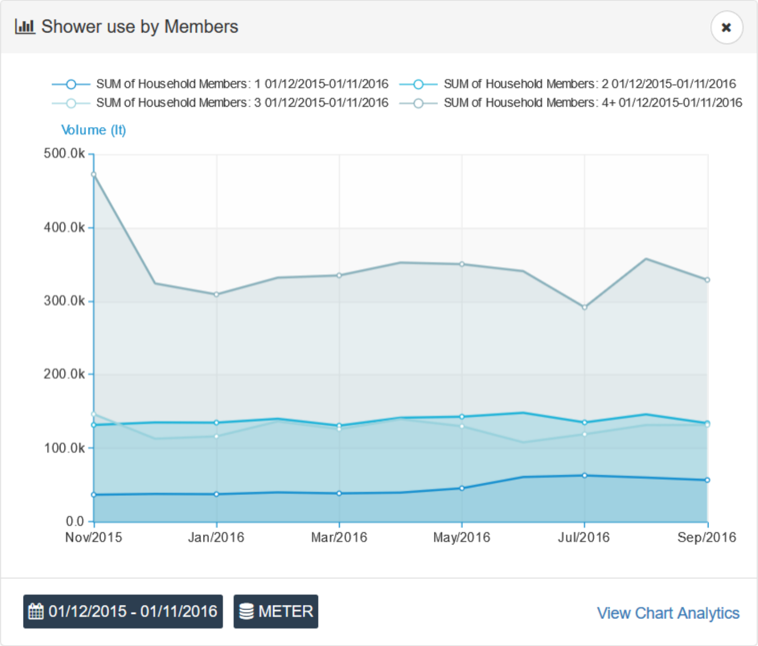
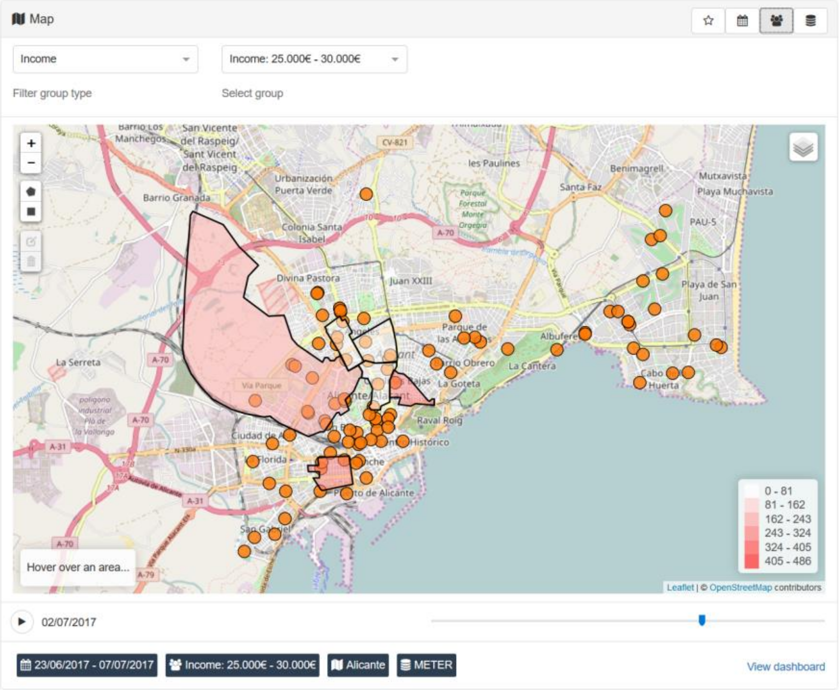


Figure 8: Sample chart

Moreover, a set of configuration parameters is displayed below the chart as a quick reference to the data used for rendering the chart.

2.2.3. Maps

Map is another visualization component that is used extensively in DAIAD@utility. Map component supports three basic types of data layers, namely: vector, heat map and choropleth layers. In Figure 9 a map with a choropleth map layer is shown.


Filter group type: Income

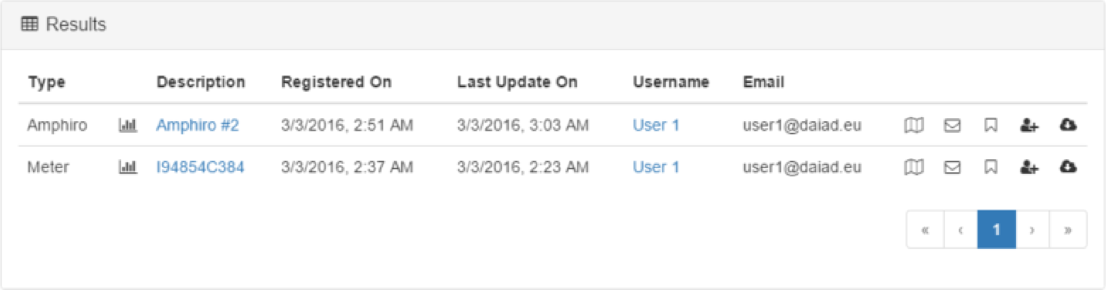
Select group: Income: 25.000€ - 30.000€

Figure 9: Sample heat map

Every map represents data at a specific timestamp. If data is available for more than a single timestamp, e.g., over an interval, time dimension is represented as a timeline slider component. The timeline slider allows users to either select a specific timestamp manually or dynamically view the data evolution over time playing an animation of the data sequence.

2.2.4. Tables

A table is flexible visualization component for displaying tabular data. A table example is illustrated in Figure 10. It consists of a single header and a collection of rows. Each row contains a collection of cells. Cells support several data types including text, date, number, percent, links and actions data types.



Results

Type	Description	Registered On	Last Update On	Username	Email
Amphiro	Amphiro #2	3/3/2016, 2:51 AM	3/3/2016, 3:03 AM	User 1	user1@daiad.eu
Meter	I94854C384	3/3/2016, 2:37 AM	3/3/2016, 2:23 AM	User 1	user1@daiad.eu

Figure 10: Sample table

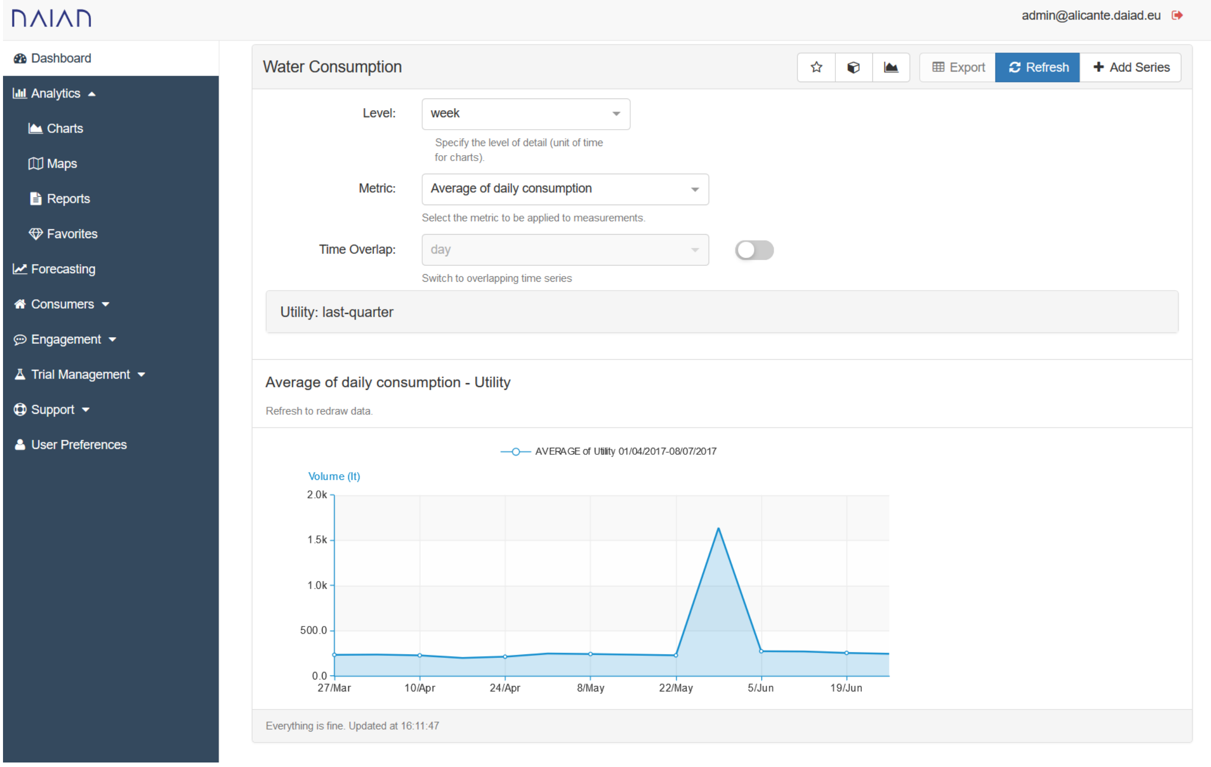
2.3. Analytics

The Analytics section allows users to explore data and execute analysis jobs, exposing the functionality of D5.2.2 ‘Updated Consumption Analytics and Forecasting Engine’. The analysis results are visualized using the UI components described in the previous sections. Data analysis results, as well as the corresponding job configuration parameters, are stored in the cluster in order to be used as input to other analysis jobs or serve as a template for initializing new jobs. The Analytics sections allows users to explore data as Charts and Maps, provides ‘*canned*’ reports, and allows them to set and manage their favorite analytics.

2.3.1. Charts

The Charts sections allows users to explore and visualize water consumption data (SWM and Amphiro b1) at arbitrary time intervals and metrics, in the form of time-series. The Chart sections is presented in Figure 11, and allows the users to:

- Select the source of water consumption data (SWM, amphiro b1)
- Define the chart’s level of temporal detail (*hour, day, week, month, year*), metric (*e.g., average, total, or peak consumption*), temporal coverage (*arbitrary*), and consumer groups (*e.g., all, pre-defined*)
- Visualize additional time-series from the same time period (e.g., peak daily consumption of large families vs small families) or different time-periods (e.g., average daily consumption for 2016 vs. average daily consumption for 2016)
- Export the chart’s data as raw data (csv) or images (png)
- Add the defined chart in the user’s favorites, allowing the user to view the specific chart immediately through the Favorites section (see below) and/or integrate it in her own dashboard.



Water Consumption

Level: week

Specify the level of detail (unit of time for charts).

Metric: Average of daily consumption

Select the metric to be applied to measurements.

Time Overlap: day

Switch to overlapping time series

Utility: last-quarter

Average of daily consumption - Utility

Refresh to redraw data.

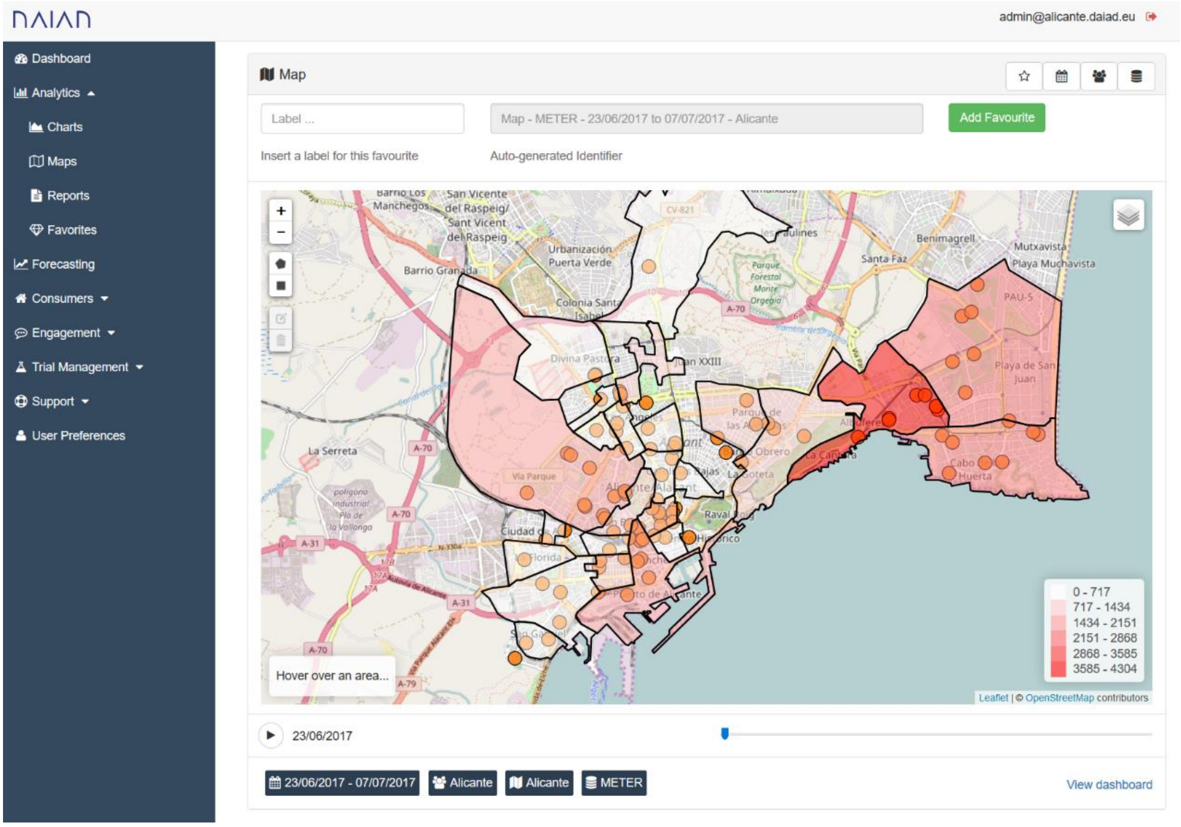
Everything is fine. Updated at 16:11:47

Figure 11: Charts section

2.3.2. Maps

The Charts sections allows users to explore and visualize water consumption data (SWM and Amphiro b1) at arbitrary time intervals and metrics, in the form of an interactive map. The Chart sections is presented in Figure 12, and allows the users to:

- Select the source of water consumption data (SWM, amphiro b1)
- Define the chart's level of temporal coverage (*arbitrary*) and consumer groups (*e.g., all, pre-defined*)
- Visualize point and polyline geometries representing consumers and administrative boundaries in the form of a choropleth, and explore the temporal evolution of consumption at daily intervals.
- Add the defined map in the user's favorites, allowing the user to view the specific map immediately through the Favorites section (see below) and/or integrate it in her own dashboard

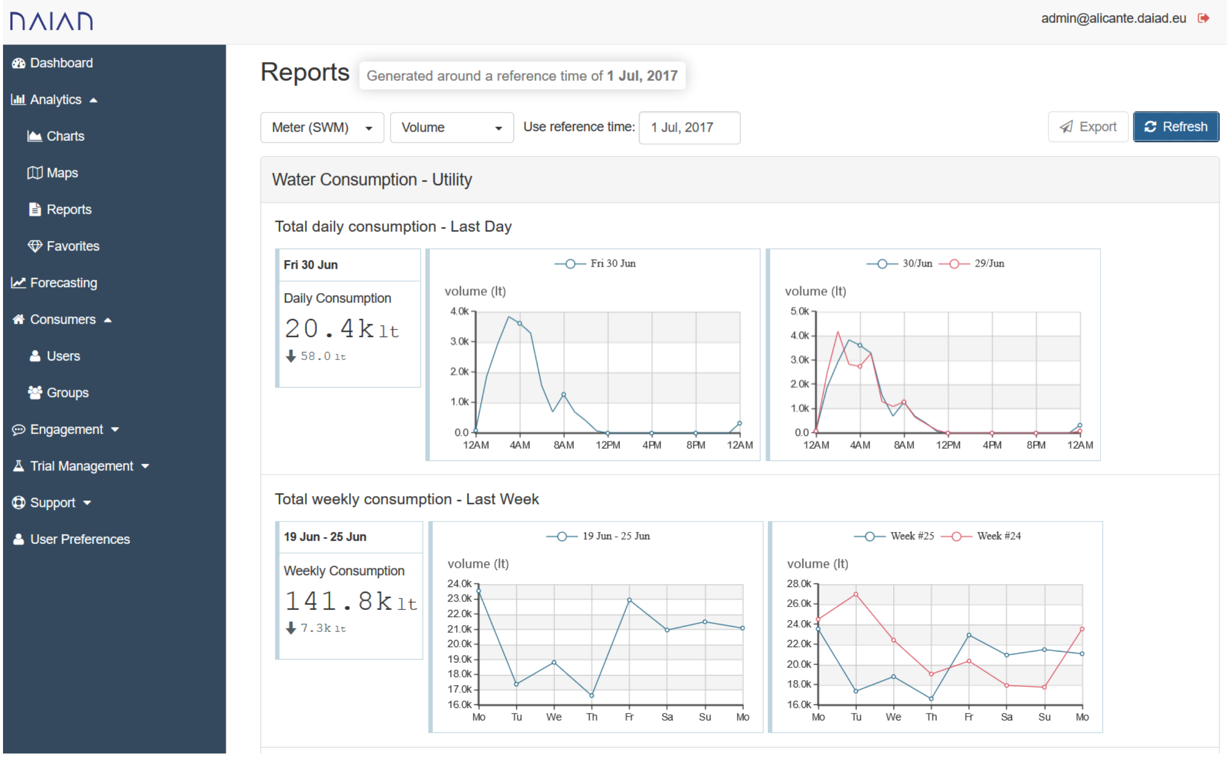


Map - METER - 23/06/2017 to 07/07/2017 - Alicante

Figure 12: Maps section

2.3.3. Reports

The Reports section provides access to a selection pre-defined reports that cover the most frequent monitoring needs of water demand experts (e.g., last, week, month consumption vs the previous corresponding time-periods).



Reports Generated around a reference time of **1 Jul, 2017**

Meter (SWM) · Volume · Use reference time: 1 Jul, 2017

Fri 30 Jun

Daily Consumption

20.4k lt

↓ 58.0 lt

19 Jun - 25 Jun

Weekly Consumption

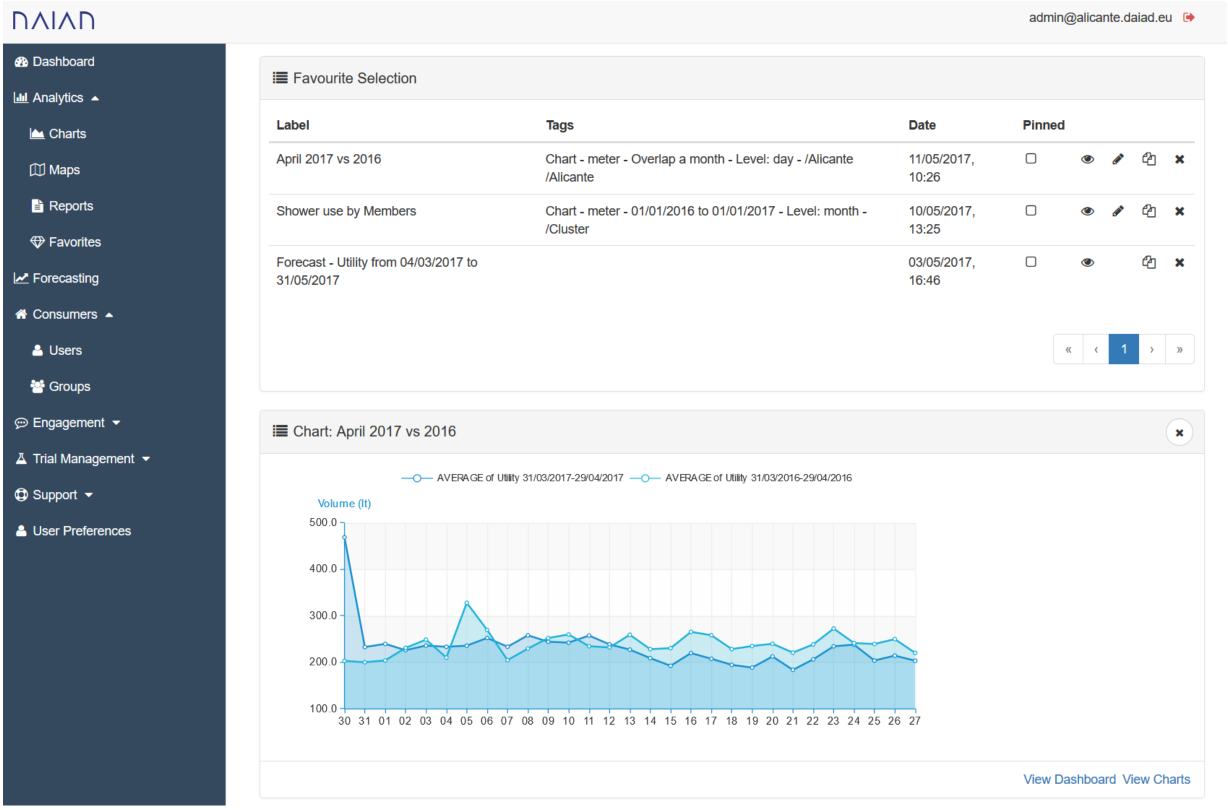
141.8k lt

↓ 7.3k lt

Figure 13: Reports section

2.3.4. Favorites

The Favorites section allows users to manage their favorite custom Chart and Map visualizations (e.g., edit name/description, view/delete, share, add to Dashboard).



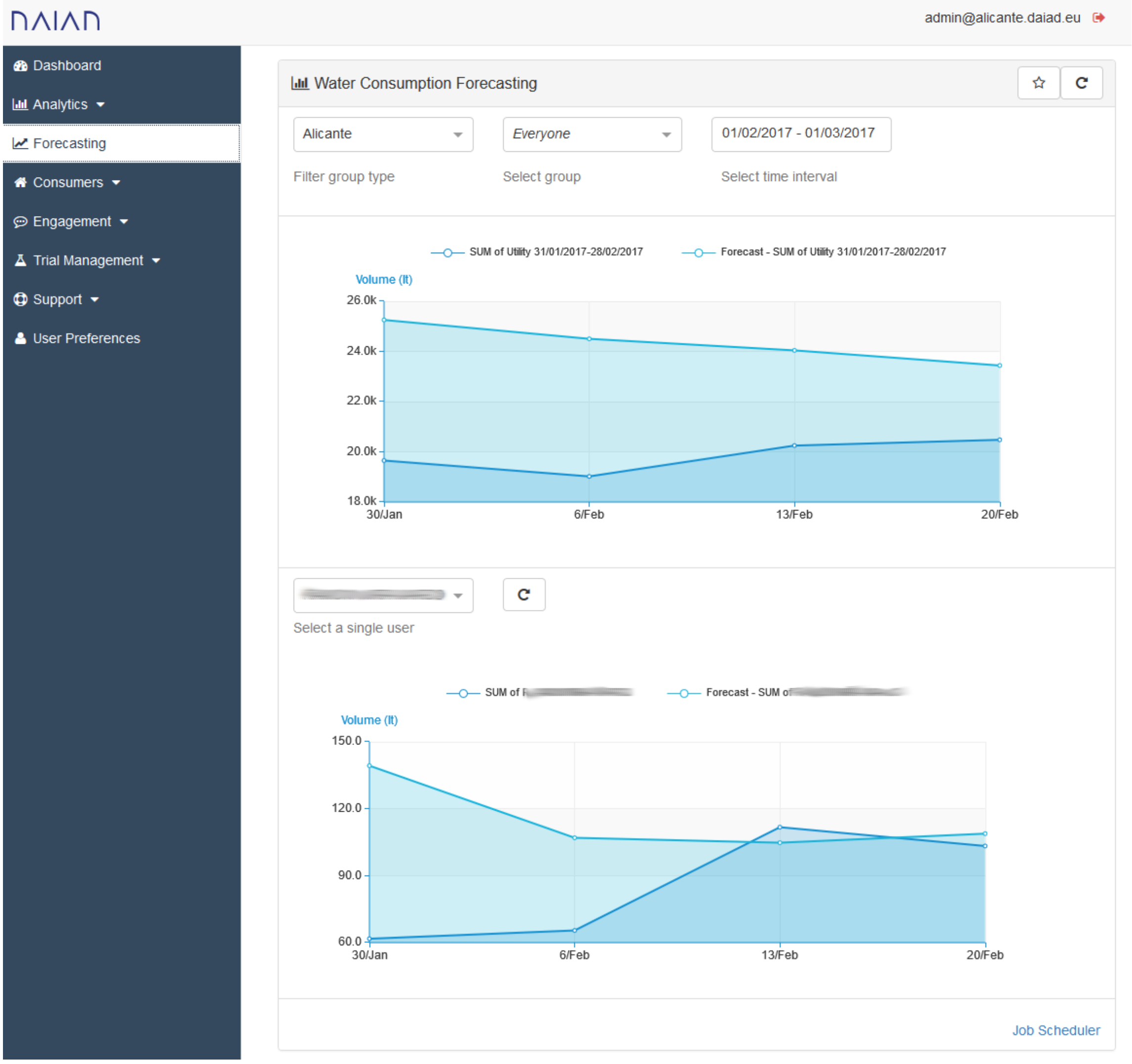
Label	Tags	Date	Pinned
April 2017 vs 2016	Chart - meter - Overlap a month - Level: day - /Alicante /Alicante	11/05/2017, 10:26	☐
Shower use by Members	Chart - meter - 01/01/2016 to 01/01/2017 - Level: month - /Cluster	10/05/2017, 13:25	☐
Forecast - Utility from 04/03/2017 to 31/05/2017		03/05/2017, 16:46	☐

Figure 14: Favorites section

2.4. Forecasting

This section allows users to explore and compare the results of our highly-granular forecasting algorithm at arbitrary levels of detail. The main forecasting view is shown in Figure 15, and allows the users to:

- Select the level of temporal coverage (*arbitrary*) and consumer groups (*e.g., all, pre-defined*)
- Visualize the forecasted consumption vs. the actual consumption for the give period (*if available*)
- View the forecasted consumption of an individual household



Filter group type: Alicante

Select group: Everyone

Select time interval: 01/02/2017 - 01/03/2017

Select a single user: [redacted]

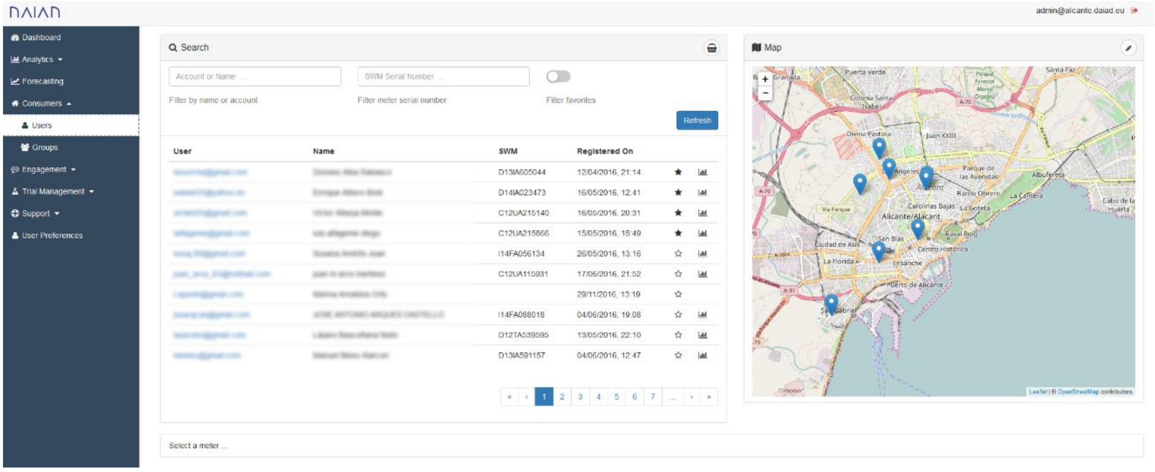
Figure 15: Forecasting example

2.5. Consumers

The Consumers menu allows expert users to view and manage detailed information about *all consumers* (*e.g., demographics, consumption*), and consumer groups (*e.g., based on household size, custom*). The User's section provides an overview of all consumers (*i.e., households where a SWM is installed*), allowing the user to explore, search, filter, and drill-down to specific consumers. The Users view is presented in Figure 16 and allows users to:

- Search for a specific consumer (*email, name/surname*) or SWM ID
- Explore the full list of consumers and/or the search results in table format (user, name, SWM ID, registration data) and map format (*locations of corresponding SWMs*)
- Set a specific consumer as a Favorite and/or add consumers in custom groups
- View detailed information about a specific consumer and/or group
- Visualize the water use of the selected consumers for the last 30 days

Similarly, the Groups section allows users to explore and compare the consumption of specific or custom consumer groups (*e.g., based on household size, members, consumption class*)



Filter by name or account

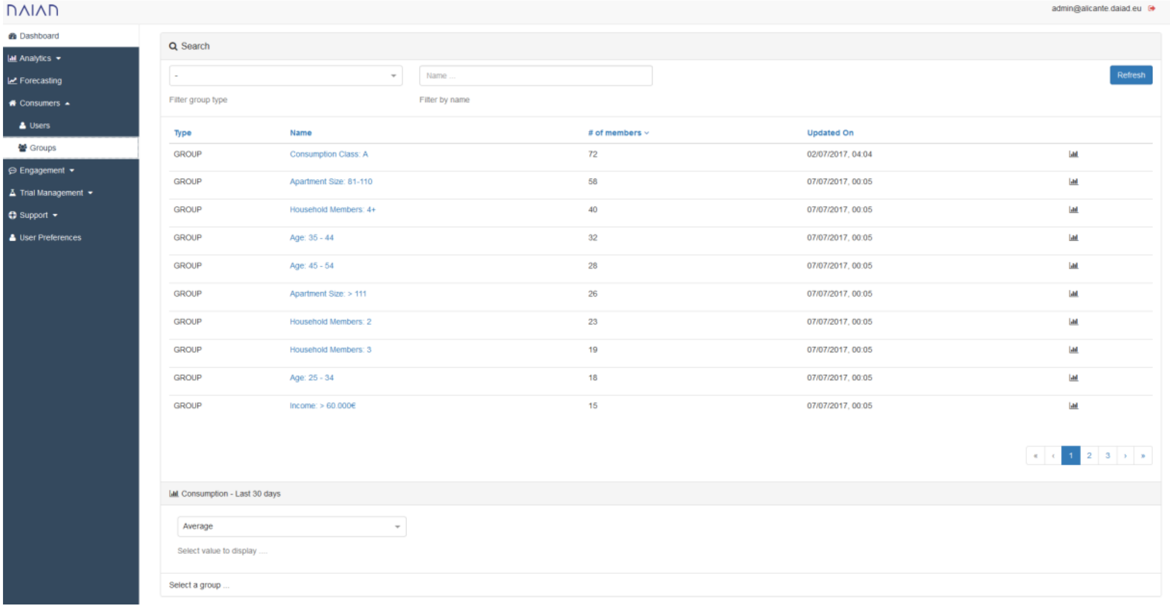
Filter meter serial number

Filter favorites

User	Name	SWM	Registered On		
[redacted]	[redacted]	D13IA605044	12/04/2016, 21:14	★	
[redacted]	[redacted]	D14IA023473	16/05/2016, 12:41	★	
[redacted]	[redacted]	C12UA215140	16/05/2016, 20:31	★	
[redacted]	[redacted]	C12UA215866	15/05/2016, 15:49	★	
[redacted]	[redacted]	I14FA056134	26/05/2016, 13:16	☆	
[redacted]	[redacted]	C12UA115931	17/06/2016, 21:52	☆	
[redacted]	[redacted]		29/11/2016, 13:19	☆	
[redacted]	[redacted]	I14FA088018	04/06/2016, 19:08	☆	
[redacted]	[redacted]	D12TA539595	13/05/2016, 22:10	☆	
[redacted]	[redacted]	D13IA591157	04/06/2016, 12:47	☆	

Select a meter ...

Figure 16: Consumers section



Filter group type

Filter by name

Type	Name	# of members	Updated On
GROUP	Consumption Class: A	72	02/07/2017, 04:04
GROUP	Apartment Size: 81-110	58	07/07/2017, 00:05
GROUP	Household Members: 4+	40	07/07/2017, 00:05
GROUP	Age: 35 - 44	32	07/07/2017, 00:05
GROUP	Age: 45 - 54	28	07/07/2017, 00:05
GROUP	Apartment Size: > 111	26	07/07/2017, 00:05
GROUP	Household Members: 2	23	07/07/2017, 00:05
GROUP	Household Members: 3	19	07/07/2017, 00:05
GROUP	Age: 25 - 34	18	07/07/2017, 00:05
GROUP	Income: > 60.000€	15	07/07/2017, 00:05

Average

Select value to display ...

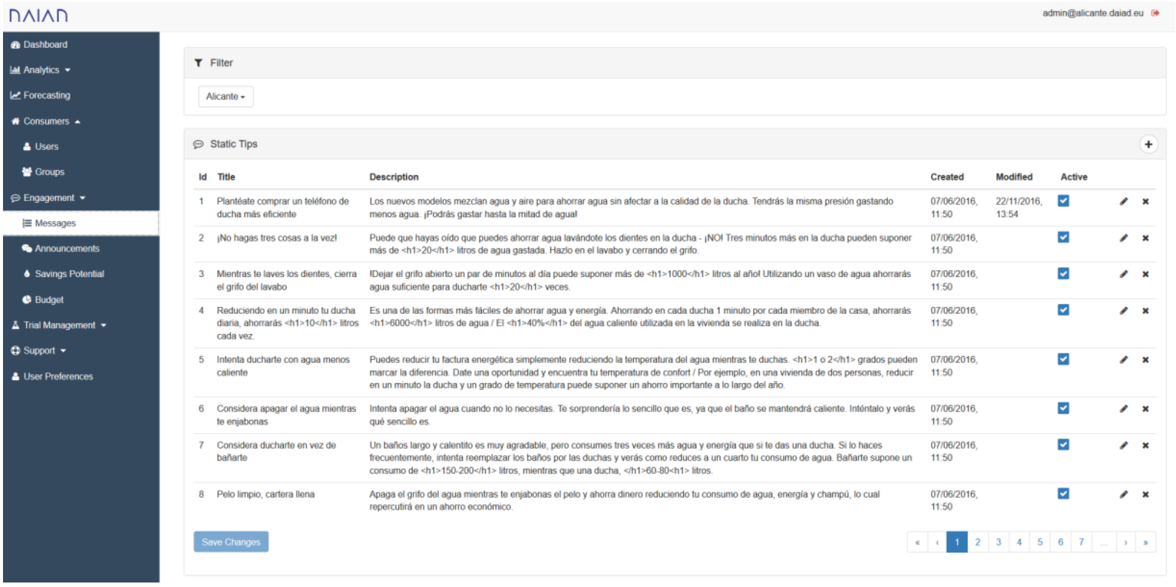
Select a group ...

Figure 17: Groups section

2.6. Engagement

The Engagement section allows users to view and edit all personalized messages (e.g., Tips, Recommendations) automatically provided to users and manage targeted communication campaigns to user via in-system messages (i.e., delivered to users via email and messages through the mobile/web apps). The Messages and Announcement sections are presented in Figure 18 and Figure 19, and allow users to:

- Edit, delete and activate/deactivate individual Messages
- Create a custom announcement and deliver it to an arbitrary number of consumers (*e.g., all, specific groups, custom consumers, single consumer*)



Id	Title	Description	Created	Modified	Active
1	Plantéate comprar un teléfono de ducha más eficiente	Los nuevos modelos mezclan agua y aire para ahorrar agua sin afectar a la calidad de la ducha. Tendrás la misma presión gastando menos agua. ¡Podrás gastar hasta la mitad de agua!	07/06/2016, 11:50	22/11/2016, 13:54	☑
2	¡No hagas tres cosas a la vez!	Puede que hayas oído que puedes ahorrar agua lavándote los dientes en la ducha - ¡NO! Tres minutos más en la ducha pueden suponer más de <h1>20</h1> litros de agua gastada. Hazlo en el lavabo y cerrando el grifo.	07/06/2016, 11:50		☑
3	Mientras te laves los dientes, cierra el grifo del lavabo	!Dejar el grifo abierto un par de minutos al día puede suponer más de <h1>1000</h1> litros al año! Utilizando un vaso de agua ahorrarás agua suficiente para ducharte <h1>20</h1> veces.	07/06/2016, 11:50		☑
4	Reduciendo en un minuto tu ducha diaria, ahorrarás <h1>10</h1> litros cada vez.	Es una de las formas más fáciles de ahorrar agua y energía. Ahorrando en cada ducha 1 minuto por cada miembro de la casa, ahorrarás <h1>6000</h1> litros de agua / El <h1>40%</h1> del agua caliente utilizada en la vivienda se realiza en la ducha.	07/06/2016, 11:50		☑
5	Intenta ducharte con agua menos caliente	Puedes reducir tu factura energética simplemente reduciendo la temperatura del agua mientras te duchas. <h1>1 o 2</h1> grados pueden marcar la diferencia. Date una oportunidad y encuentra tu temperatura de confort / Por ejemplo, en una vivienda de dos personas, reducir en un minuto la ducha y un grado de temperatura puede suponer un ahorro importante a lo largo del año.	07/06/2016, 11:50		☑
6	Considera apagar el agua mientras te enjabonas	Intenta apagar el agua cuando no lo necesitas. Te sorprendería lo sencillo que es, ya que el baño se mantendrá caliente. Inténtalo y verás qué sencillo es.	07/06/2016, 11:50		☑
7	Considera ducharte en vez de bañarte	Un baños largo y calentito es muy agradable, pero consumes tres veces más agua y energía que si te das una ducha. Si lo haces frecuentemente, intenta reemplazar los baños por las duchas y verás como reduces a un cuarto tu consumo de agua. Bañarte supone un consumo de <h1>150-200</h1> litros, mientras que una ducha, </h1>60-80<h1> litros.	07/06/2016, 11:50		☑
8	Pelo limpio, cartera llena	Apaga el grifo del agua mientras te enjabonas el pelo y ahorra dinero reduciendo tu consumo de agua, energía y champú, lo cual repercutirá en un ahorro económico.	07/06/2016, 11:50		☑

Figure 18: Messages section

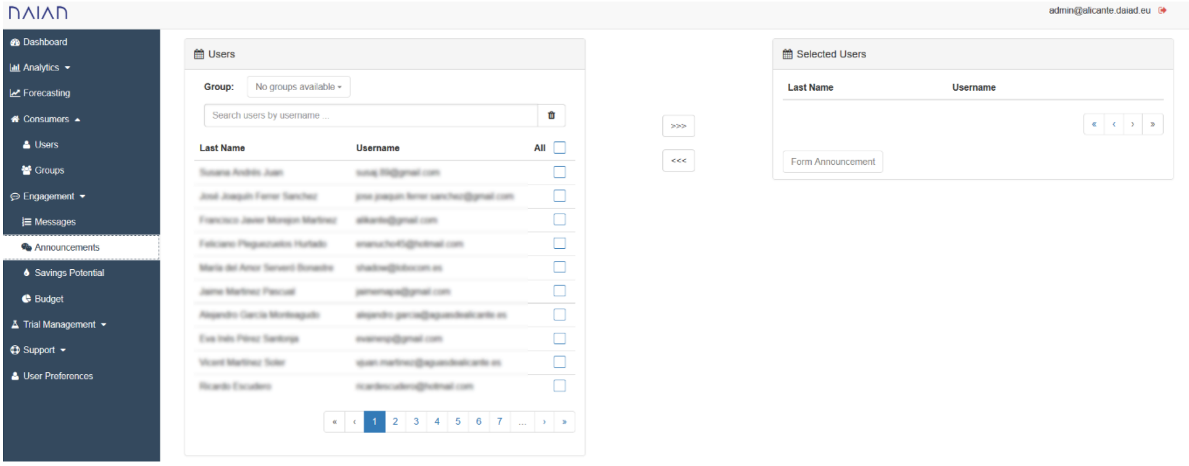
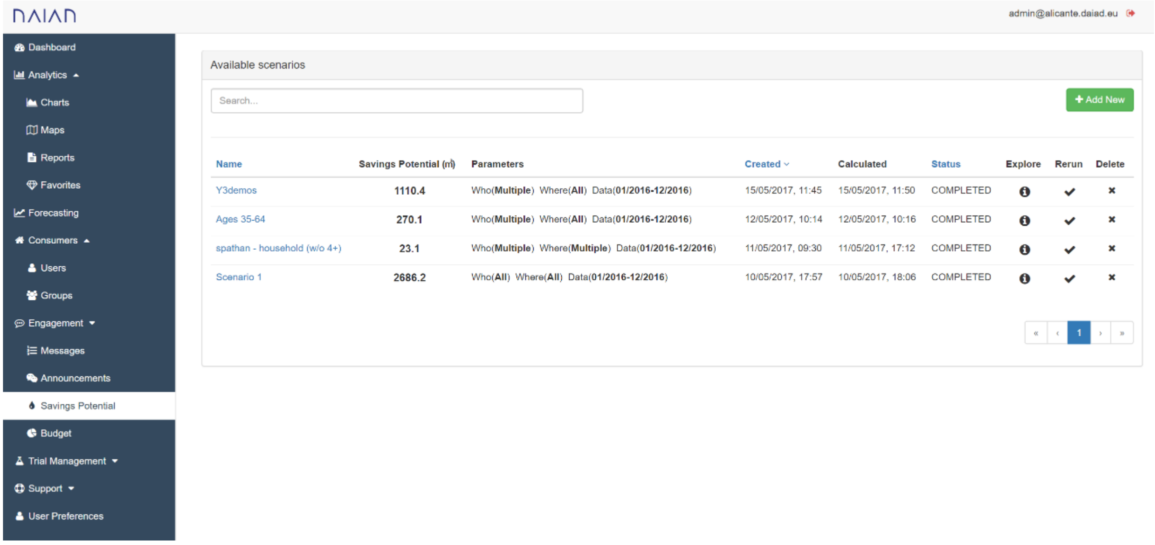


Figure 19: Announcements section

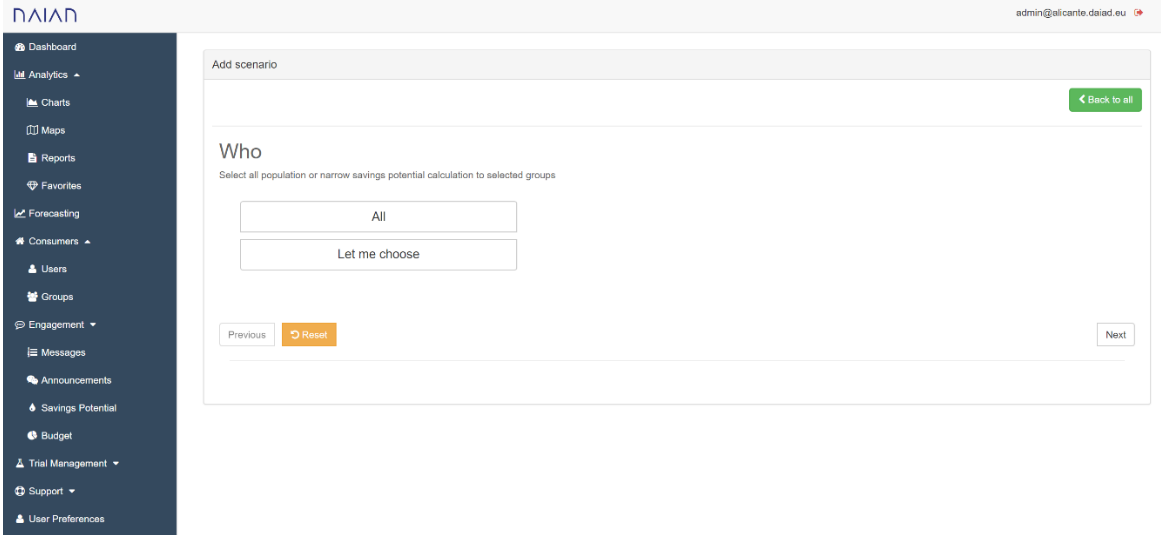
In addition, the Engagement section provides two facilities allowing expert users to explore the water savings potential of their customers, and enforce personalized water savings goals. Specifically, the 'Savings Potential' provides to expert users estimates down at the *household level*, of the maximum water savings that can be achieved (i.e., estimate of elastic consumption), based on the historical water consumption data and any available socio-demographic data available for the provided household (Figure 20). The facility encapsulates and invokes the corresponding consumption analytics service implemented on Apache Flink (see D.5.2.2), and enables the user to select arbitrary population groups (consumption class, income, household size, household members, or the *entire* city), geographical areas (administrative levels), and historical water consumption data (last year or custom) via a simple and intuitive step-by-step wizard (Figure 21, Figure 22). The output of each individual scenario is presented to the user, enabling her to assess its impact across the selected population and dimensions (Figure 23).



Available scenarios

Name	Savings Potential (m³)	Parameters	Created	Calculated	Status	Explore	Rerun	Delete
Y3demos	**1110.4**	Who(**Multiple**) Where(**All**) Data(**01/2016-12/2016**)	15/05/2017, 11:45	15/05/2017, 11:50	COMPLETED	ℹ	✓	✖
Ages 35-64	**270.1**	Who(**Multiple**) Where(**All**) Data(**01/2016-12/2016**)	12/05/2017, 10:14	12/05/2017, 10:16	COMPLETED	ℹ	✓	✖
spathan - household (w/o 4+)	**23.1**	Who(**Multiple**) Where(**Multiple**) Data(**01/2016-12/2016**)	11/05/2017, 09:30	11/05/2017, 17:12	COMPLETED	ℹ	✓	✖
Scenario 1	**2686.2**	Who(**All**) Where(**All**) Data(**01/2016-12/2016**)	10/05/2017, 17:57	10/05/2017, 18:06	COMPLETED	ℹ	✓	✖

Figure 20: Savings potential; list of executed savings scenarios



Add scenario

Who

Select all population or narrow savings potential calculation to selected groups

Figure 21: Savings potential wizard; first page; step-by-step selection of scenario parameters

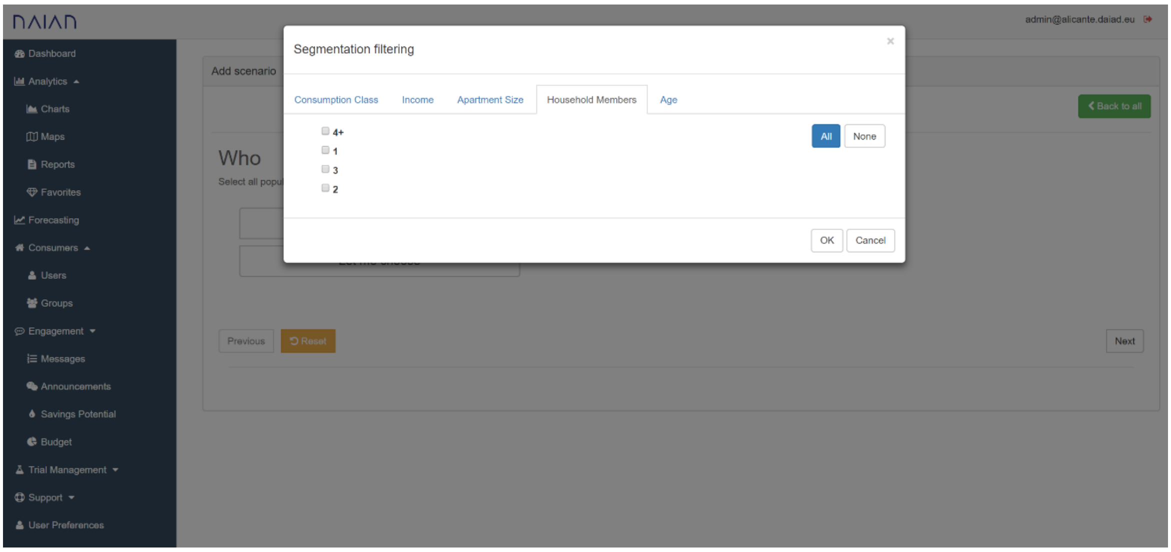
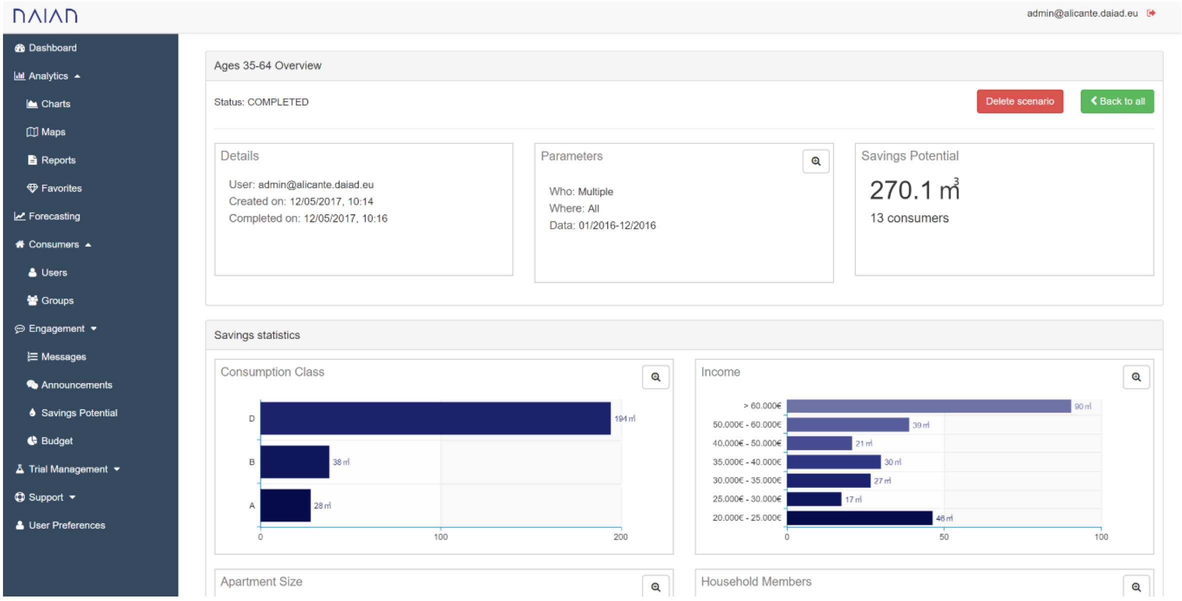


Figure 22: Savings potential wizard; selection of consumer segments



Ages 35-64 Overview

Status: COMPLETED

Details

User: admin@alicante.daiad.eu
Created on: 12/05/2017, 10:14
Completed on: 12/05/2017, 10:16

Parameters

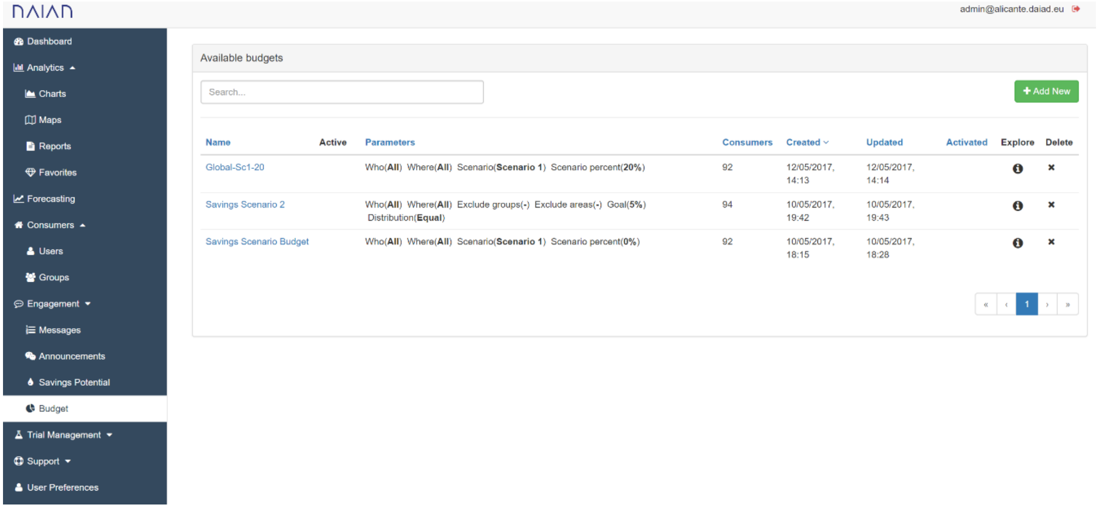
Who: Multiple
Where: All
Data: 01/2016-12/2016

Savings Potential

270.1 m³
13 consumers

Figure 23: Results for a savings scenario displayed across multiple consumers dimensions

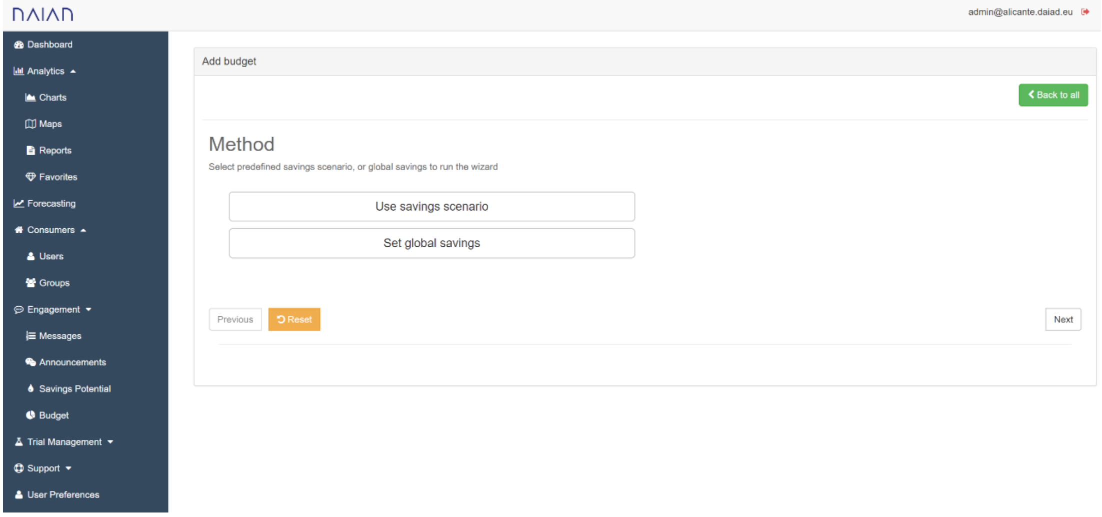
The 'Budget' facility allows expert users to define individual personalized water savings goals for each individual household, and optionally enforce them to the population via DAIAD's in-app water savings goal. The service supports water utilities in urgent or planned water restriction campaigns, both by reaching and engaging consumers, as well as by establishing *fair* water consumption goals (Figure 24). The facility encapsulates and invokes the corresponding consumption analytics service implemented on Apache Flink (see D.5.2.2) through a simple step-by-step wizard. The first step (Figure 21) allows the user to use as the basis of the algorithm an *already executed* Savings potential scenario, or define a global savings goal. In the first case, the estimated maximum water savings per household for the selected Savings Scenario is used with a correction coefficient (e.g., set savings goal to 30% of maximum savings goal). In the second case, the user defines a global goal (Figure 26) and has the option to distribute it to individual households in an *equal* or *fair* manner (Figure 27). The selection of equal implies that the global water savings (e.g., 4.5%) are applied to the entire population (in this example, all must save by 4.5%) or the selected consumer segments. The selection of a fair manner, first enables the user to select arbitrary population segments as a target (i.e., who should be included) based on socio-demographics and location criteria (Figure 28) and then *distributes* the global savings goal to each household, based on an estimate of its elastic water consumption. The output of each individual Budget is presented to the user, enabling her to assess its impact across the selected population and dimensions (Figure 29) down at the household level (Figure 30).



Available budgets

Name	Active	Parameters	Consumers	Created	Updated	Activated	Explore	Delete
Global-Sc1-20		Who(**All**) Where(**All**) Scenario(**Scenario 1**) Scenario percent(**20%**)	92	12/05/2017, 14:13	12/05/2017, 14:14		ℹ	✖
Savings Scenario 2		Who(**All**) Where(**All**) Exclude groups(**-**) Exclude areas(**-**) Goal(**5%**) Distribution(**Equal**)	94	10/05/2017, 19:42	10/05/2017, 19:43		ℹ	✖
Savings Scenario Budget		Who(**All**) Where(**All**) Scenario(**Scenario 1**) Scenario percent(**0%**)	92	10/05/2017, 18:15	10/05/2017, 18:28		ℹ	✖

Figure 24: List of available water savings budgets

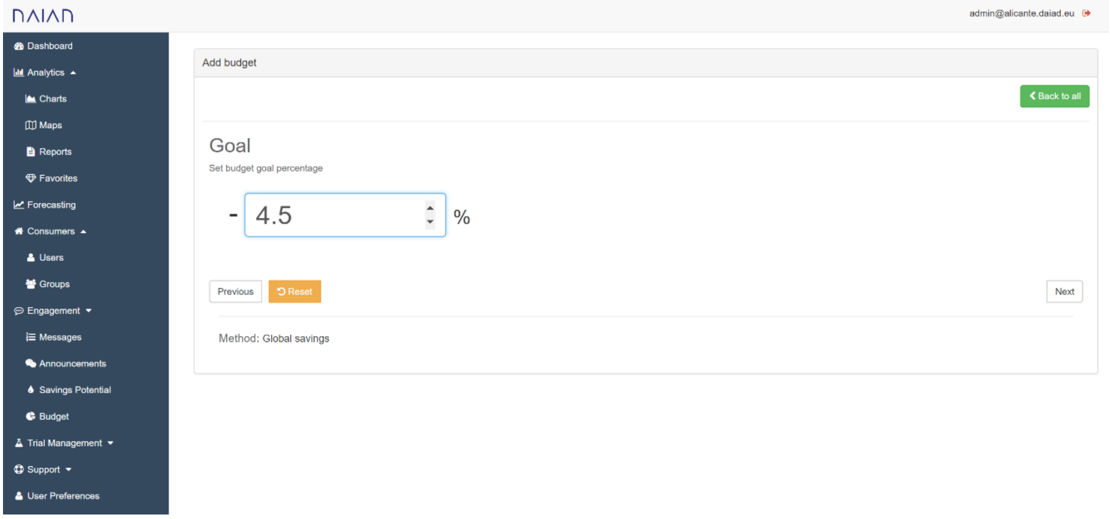


Add budget

Method

Select predefined savings scenario, or global savings to run the wizard

Figure 25: First step of budget scenario; calculation based on available Savings scenario or definition of global savings goal



Add budget

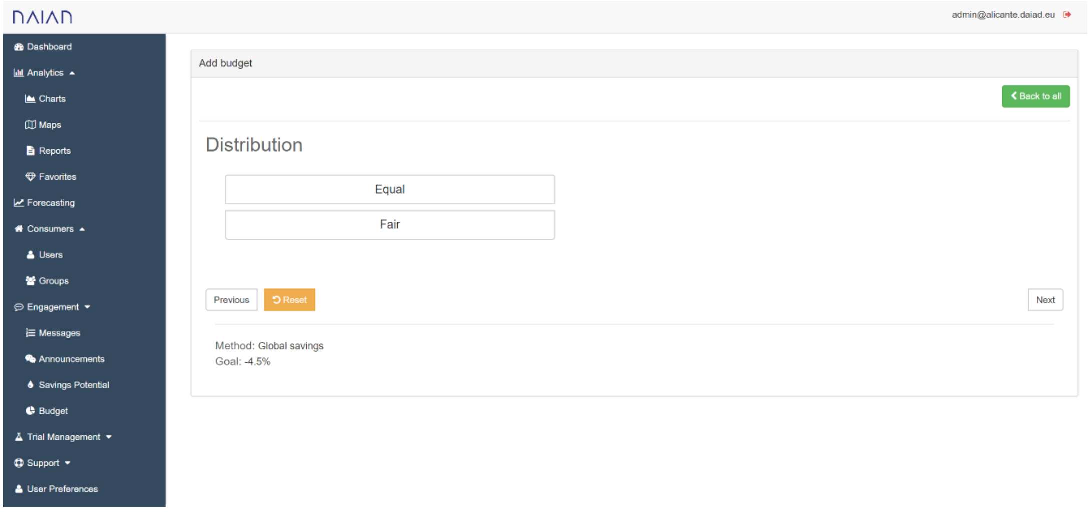
Goal

Set budget goal percentage

- 4.5 %

Method: Global savings

Figure 26: Setting a global water savings goal

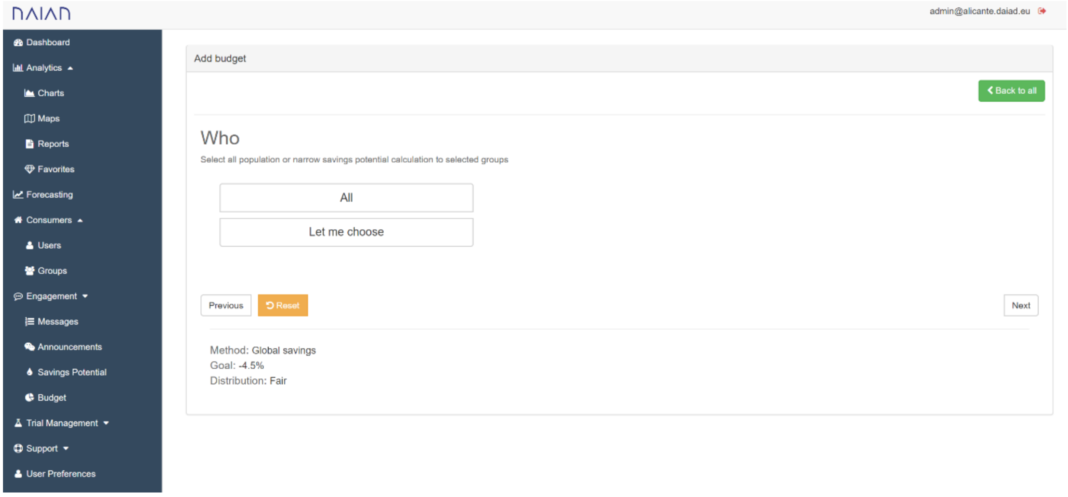


Add budget

Distribution

Method: Global savings
Goal: -4.5%

Figure 27: Decision point on distribution of water savings goal: equally (all the same %) or fairly (based on estimate of elastic consumption)



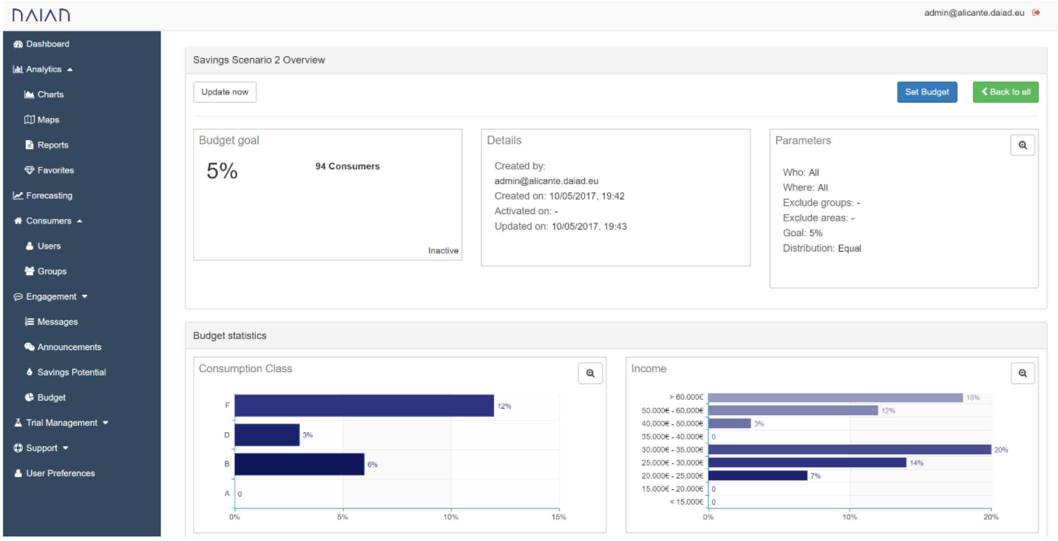
Add budget

Who

Select all population or narrow savings potential calculation to selected groups

Method: Global savings
Goal: -4.5%
Distribution: Fair

Figure 28: Setting the target of a water savings goal; the entire population or arbitrary consumer segments (socio-demographics, location)



Savings Scenario 2 Overview

Budget goal

5%

94 Consumers

Inactive

Details

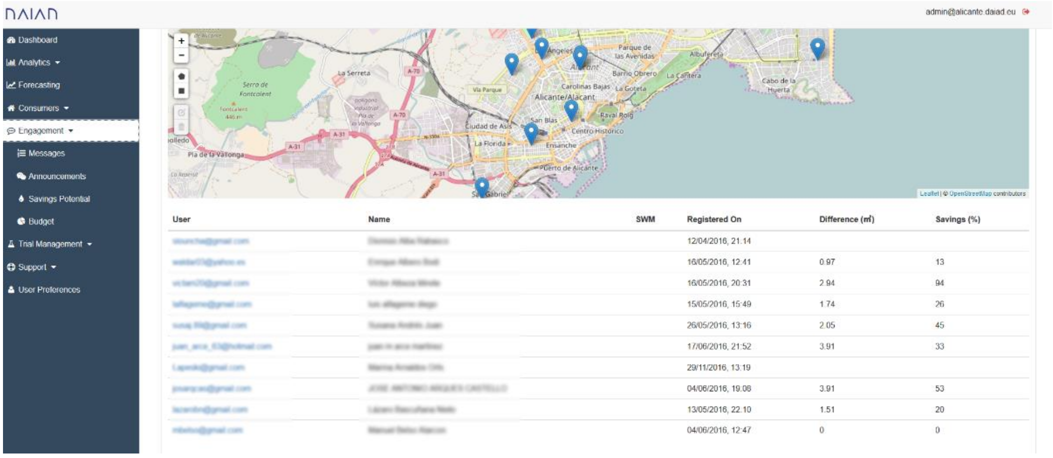
Created by:
admin@alicante.daiad.eu
Created on: 10/05/2017, 19:42
Activated on: -
Updated on: 10/05/2017, 19:43

Parameters

Who: All
Where: All
Exclude groups: -
Exclude areas: -
Goal: 5%
Distribution: Equal

Budget statistics

Figure 29: Results of a budget scenario presented across various dimensions



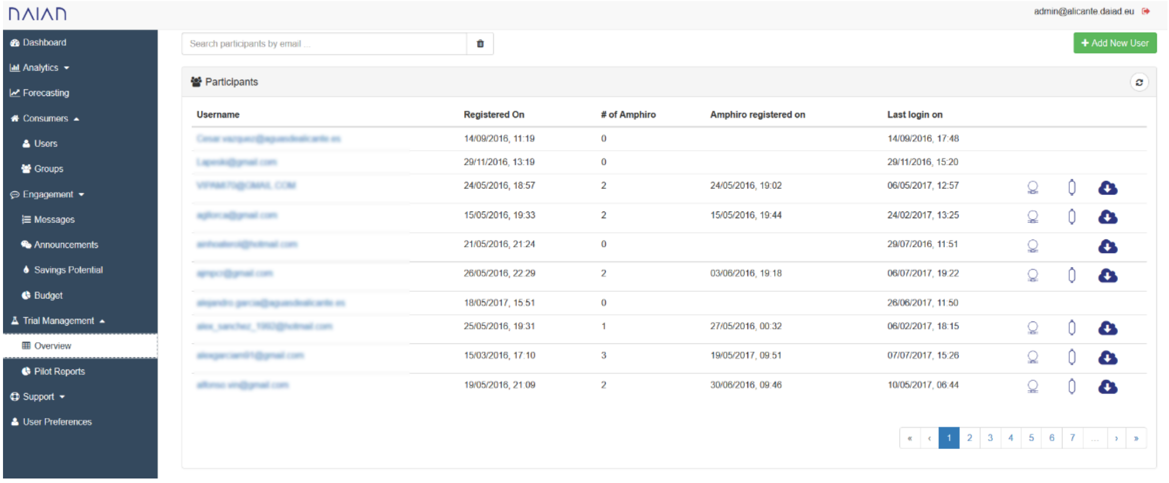
User	Name	SWM	Registered On	Difference (m³)	Savings (%)
[redacted]	[redacted]		12/04/2016, 21:14		
[redacted]	[redacted]		16/05/2016, 12:41	0.97	13
[redacted]	[redacted]		16/05/2016, 20:31	2.94	94
[redacted]	[redacted]		15/05/2016, 15:49	1.74	26
[redacted]	[redacted]		26/05/2016, 13:16	2.05	45
[redacted]	[redacted]		17/06/2016, 21:52	3.91	33
[redacted]	[redacted]		29/11/2016, 13:19		
[redacted]	[redacted]		04/06/2016, 19:08	3.91	53
[redacted]	[redacted]		13/05/2016, 22:10	1.51	20
[redacted]	[redacted]		04/06/2016, 12:47	0	0

Figure 30: Results of a budget scenario presented for each individual household

2.7. Trial Management

This section provides a suite of remote monitoring and management facilities for organizing, performing, and evaluating water consumption studies in small-scale panels of the entire consumer population to assess the effectiveness of interventions before their roll-out across all consumers. The Trial Management functionality has been applied in the context of WP7 to organize and monitor the DAIAD Trials, and has been integrated into the system as an important function to support actionable decision making and investment planning regarding consumer engagement and demand management. The Overview and Pilot Reports sections are presented in Figure 31 and Figure 32 respectively, and allow the user to:

- Search for an individual member of the panel population (name/surname, account, email)
- View and explore information for all panel members (*e.g., registration, last login*)
- Visualize and download the member's water consumption during the timeline of the study and for all available water consumption sensors
- Visualize the evolution of the study and statistics about water consumption



Username	Registered On	# of Amphiro	Amphiro registered on	Last login on
[redacted]	14/09/2016, 11:19	0		14/09/2016, 17:48
[redacted]	29/11/2016, 13:19	0		29/11/2016, 15:20
[redacted]	24/05/2016, 18:57	2	24/05/2016, 19:02	06/05/2017, 12:57
[redacted]	15/05/2016, 19:33	2	15/05/2016, 19:44	24/02/2017, 13:25
[redacted]	21/05/2016, 21:24	0		29/07/2016, 11:51
[redacted]	26/05/2016, 22:29	2	03/06/2016, 19:18	06/07/2017, 19:22
[redacted]	18/05/2017, 15:51	0		26/06/2017, 11:50
[redacted]	25/05/2016, 19:31	1	27/05/2016, 00:32	06/02/2017, 18:15
[redacted]	15/03/2016, 17:10	3	19/05/2017, 09:51	07/07/2017, 15:26
[redacted]	19/05/2016, 21:09	2	30/06/2016, 09:46	10/05/2017, 06:44

Figure 31: Pilot overview section

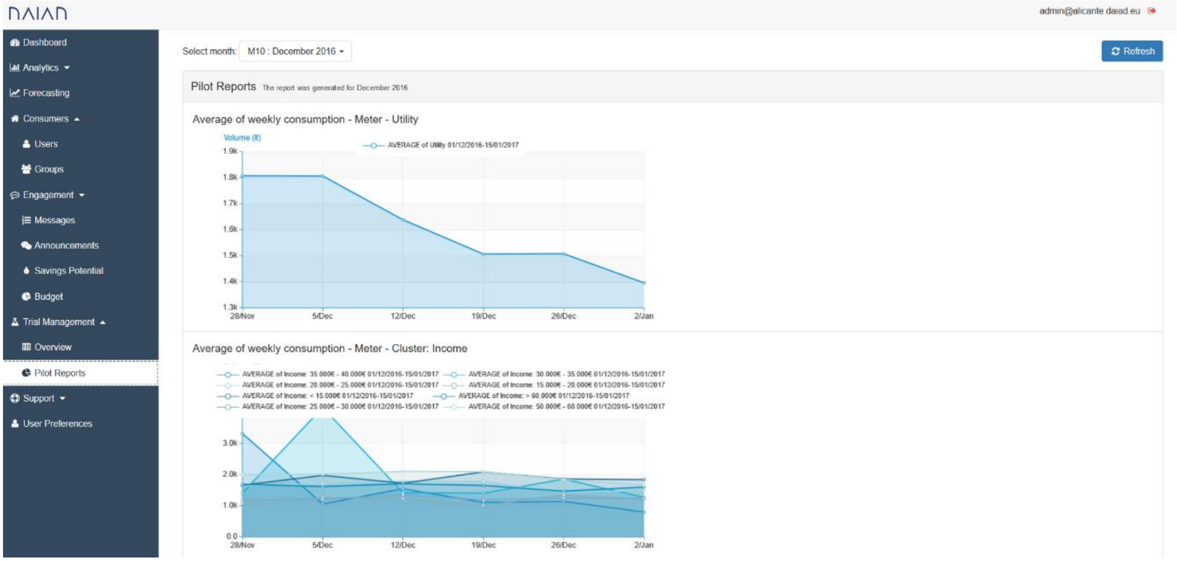

admin@alicante.daiad.eu
Dashboard
Analytics
Forecasting
Consumers
Users
Groups
Engagement
Messages
Announcements
Savings Potential
Budget
Trial Management
Overview
Pilot Reports
Support
User Preferences
Select month: M10 : December 2016
Refresh
Pilot Reports The report was generated for December 2016
Average of weekly consumption - Meter - Utility
Volume (lt)
AVERAGE of Utility 01/12/2016-15/01/2017
1.9k
1.8k
1.7k
1.6k
1.5k
1.4k
1.3k
28/Nov
5/Dec
12/Dec
19/Dec
26/Dec
2/Jan
Average of weekly consumption - Meter - Cluster: Income
AVERAGE of Income: 35.000€ - 40.000€ 01/12/2016-15/01/2017
AVERAGE of Income: 30.000€ - 35.000€ 01/12/2016-15/01/2017
AVERAGE of Income: 20.000€ - 25.000€ 01/12/2016-15/01/2017
AVERAGE of Income: 15.000€ - 20.000€ 01/12/2016-15/01/2017
AVERAGE of Income: < 15.000€ 01/12/2016-15/01/2017
AVERAGE of Income: > 60.000€ 01/12/2016-15/01/2017
AVERAGE of Income: 25.000€ - 30.000€ 01/12/2016-15/01/2017
AVERAGE of Income: 50.000€ - 60.000€ 01/12/2016-15/01/2017
3.0k
2.0k
1.0k
0.0
28/Nov
5/Dec
12/Dec
19/Dec
26/Dec
2/Jan

Figure 32: Pilot reports section

2.8. Support

The Support section assembles several facilities supporting system administrators across various aspects of the DAIAD system, including: (a) Mode management facility (Figure 33), allowing them to remote control, monitor, and troubleshoot the functionality of the mobile/web app and amphiro b1 for individual consumers, (b) Data management (Figure 34), allowing them to manually upload select data sets (*e.g., SWM IDs, external water consumption data*), (c) Data export (Figure 35) allowing them to download the entire dataset of water consumption and consumer information available, and (d) Messages (Figure 36), which presents statistics regarding the generation and delivery of Messages automatically generated and sent to consumers.

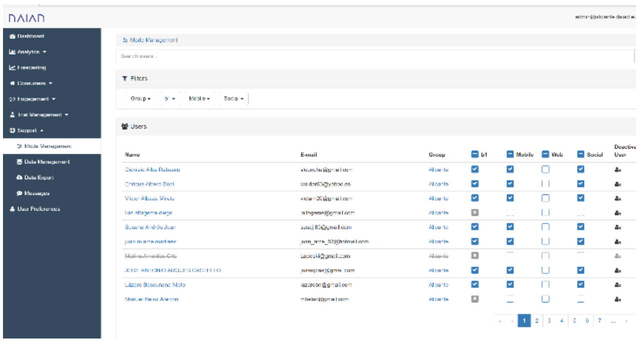


Figure 33: Mode Management

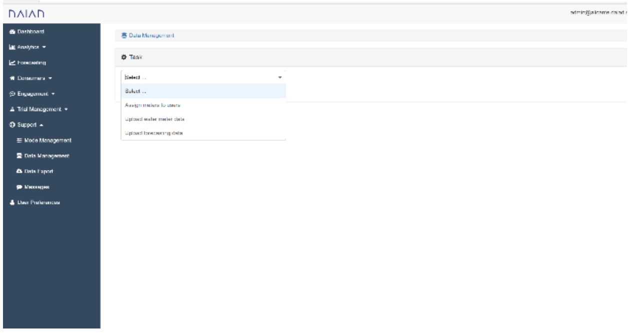


Figure 34: Data management

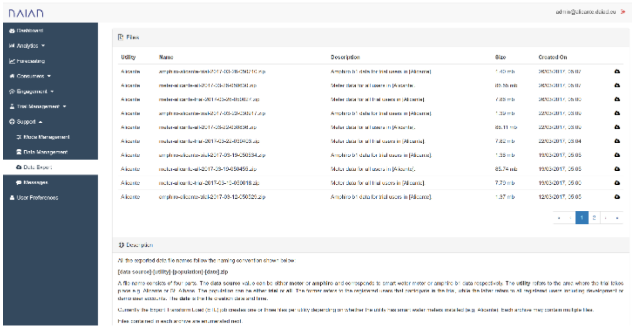


Figure 35: Data export

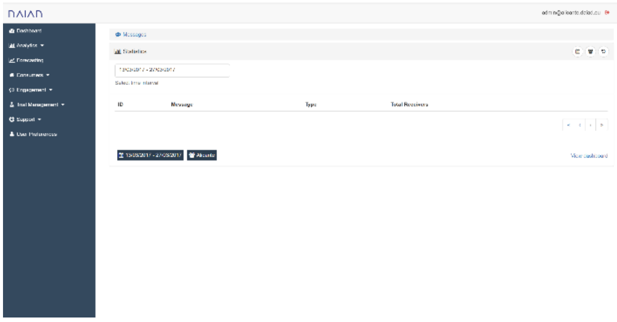
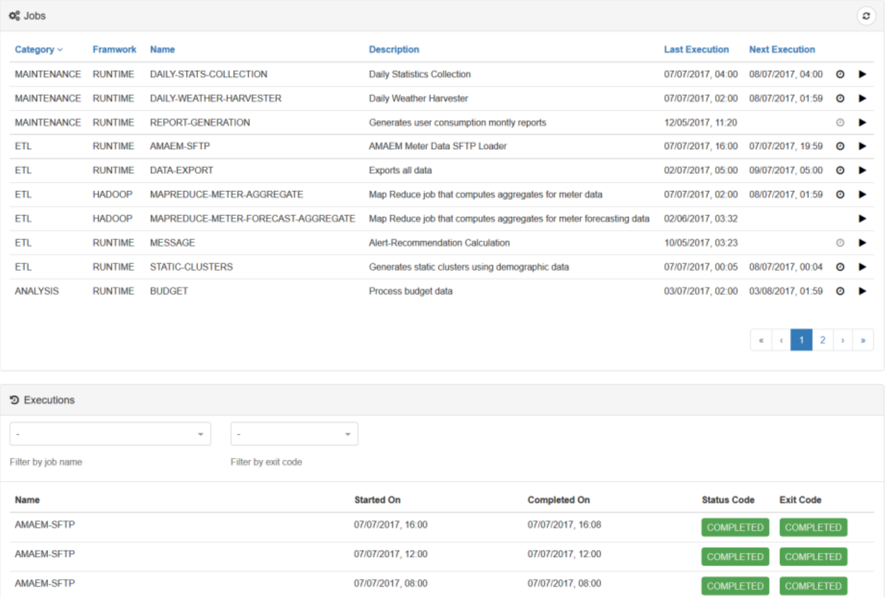


Figure 36: Messages

2.9. Job Management

DAIAD@utility supports several types of long running jobs including analysis, forecasting and generic data curation jobs that must be monitored and report their status. To this end, a dedicated view has been implemented that manages all types of jobs in a unified manner as shown in Figure 37. Job management view allows users to monitor the status of all scheduled jobs as well as iterate the execution history of all jobs. Users can disable scheduled jobs or execute jobs whenever required.



Category	Framwork	Name	Description	Last Execution	Next Execution
MAINTENANCE	RUNTIME	DAILY-STATS-COLLECTION	Daily Statistics Collection	07/07/2017, 04:00	08/07/2017, 04:00
MAINTENANCE	RUNTIME	DAILY-WEATHER-HARVESTER	Daily Weather Harvester	07/07/2017, 02:00	08/07/2017, 01:59
MAINTENANCE	RUNTIME	REPORT-GENERATION	Generates user consumption montly reports	12/05/2017, 11:20	
ETL	RUNTIME	AMAEM-SFTP	AMAEM Meter Data SFTP Loader	07/07/2017, 16:00	07/07/2017, 19:59
ETL	RUNTIME	DATA-EXPORT	Exports all data	02/07/2017, 05:00	09/07/2017, 05:00
ETL	HADOOP	MAPREDUCE-METER-AGGREGATE	Map Reduce job that computes aggregates for meter data	07/07/2017, 02:00	08/07/2017, 01:59
ETL	HADOOP	MAPREDUCE-METER-FORECAST-AGGREGATE	Map Reduce job that computes aggregates for meter forecasting data	02/06/2017, 03:32	
ETL	RUNTIME	MESSAGE	Alert-Recommendation Calculation	10/05/2017, 03:23	
ETL	RUNTIME	STATIC-CLUSTERS	Generates static clusters using demographic data	07/07/2017, 00:05	08/07/2017, 00:04
ANALYSIS	RUNTIME	BUDGET	Process budget data	03/07/2017, 02:00	03/08/2017, 01:59

Filter by job name

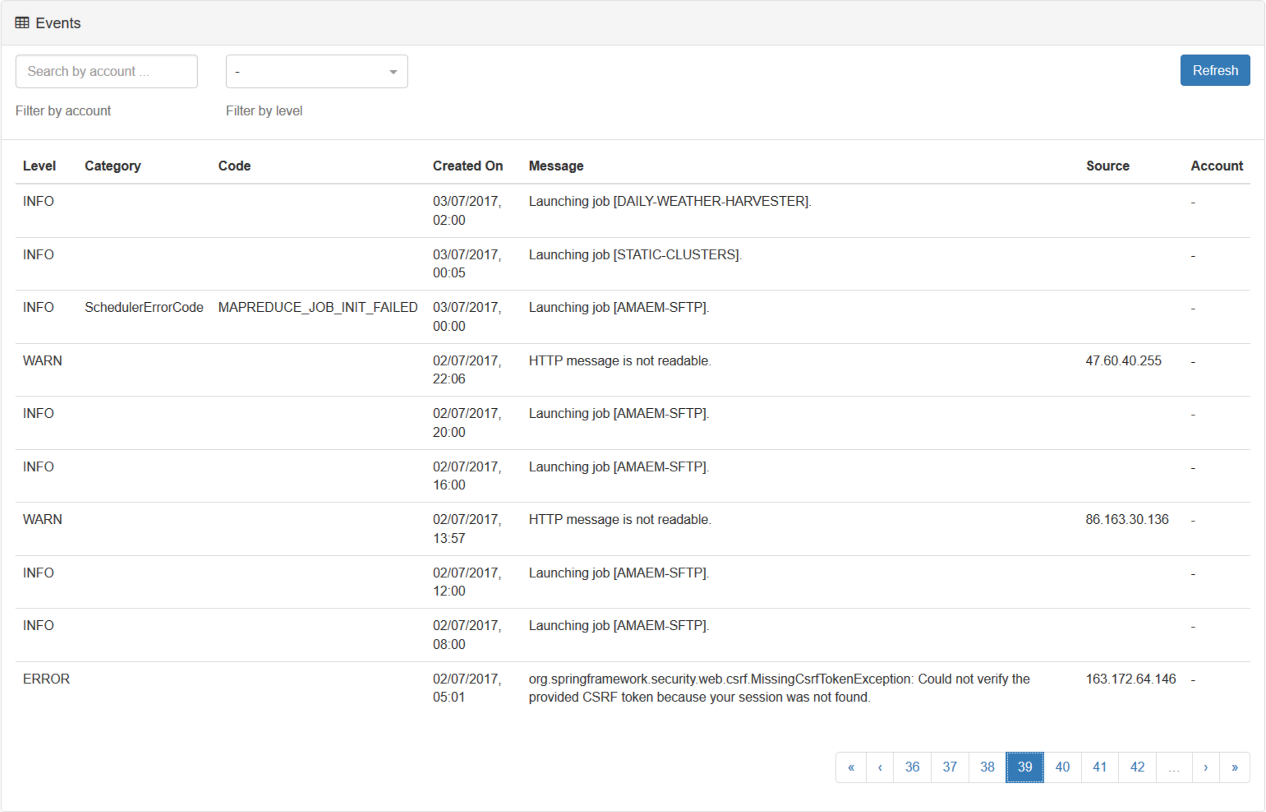
Filter by exit code

Name	Started On	Completed On	Status Code	Exit Code
AMAEM-SFTP	07/07/2017, 16:00	07/07/2017, 16:08	COMPLETED	COMPLETED
AMAEM-SFTP	07/07/2017, 12:00	07/07/2017, 12:00	COMPLETED	COMPLETED
AMAEM-SFTP	07/07/2017, 08:00	07/07/2017, 08:00	COMPLETED	COMPLETED

Figure 37: Job management view

2.10. Logging

DAIAD@utility consists of numerous software modules and each of them may log messages of several severity levels, including *debug, information, warning or error*. To integrate all messages from all modules and enable searching them in a unified manner, a log viewer has been implemented as shown in Figure 38. The log viewer allows administrators to search messages based on either the account that generated the message or the severity level. The former feature is particularly useful when troubleshooting trial user issues.



Level	Category	Code	Created On	Message	Source	Account
INFO			03/07/2017, 02:00	Launching job [DAILY-WEATHER-HARVESTER].		-
INFO			03/07/2017, 00:05	Launching job [STATIC-CLUSTERS].		-
INFO	SchedulerErrorCode	MAPREDUCE_JOB_INIT_FAILED	03/07/2017, 00:00	Launching job [AMAEM-SFTP].		-
WARN			02/07/2017, 22:06	HTTP message is not readable.	47.60.40.255	-
INFO			02/07/2017, 20:00	Launching job [AMAEM-SFTP].		-
INFO			02/07/2017, 16:00	Launching job [AMAEM-SFTP].		-
WARN			02/07/2017, 13:57	HTTP message is not readable.	86.163.30.136	-
INFO			02/07/2017, 12:00	Launching job [AMAEM-SFTP].		-
INFO			02/07/2017, 08:00	Launching job [AMAEM-SFTP].		-
ERROR			02/07/2017, 05:01	org.springframework.security.web.csrf.MissingCsrfTokenException: Could not verify the provided CSRF token because your session was not found.	163.172.64.146	-

Figure 38: Log viewer