REPORT ON DELIVERABLE 1.3

# Beta DAIAD integrated prototype

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 24, 29/02/2016 |
| Actual Submission Date | 08/07/2016 |
| Work Package | WP1 DAIAD Requirements, Architecture and Integration |
| Tasks | 1.4 Integration |
| Type | Prototype |
| Approval Status | Submitted for approval |
| Version | 1.2 |
| Number of Pages | 88 |
| Filename | D1.3_DAIAD_beta_integrated_prototype.pdf |

# Abstract

This report presents an overview of the Prototype Deliverable D1.3 "Beta DAIAD integrated prototype", which includes all DAIAD software components currently under evaluation in the context of our user trials. First, we present our methodologies, practices, and infrastructures for integration and testing. Then, we provide an overview of the current architecture of the Beta DAIAD integrated prototype, its subsystems, as well as libraries and external frameworks used. Finally, we present the characteristics of the production deployment environment.

# History

| version | date | reason | revised by |
|---|---|---|---|
| 0.1 | 11/01/2016 | First draft | Spiros Athanasiou |
| 0.4 | 08/02/2016 | Added content to various Sections | Yannis Kouvaras |
| 0.8 | 17/02/2016 | Various edits | Yannis Kouvaras, Nikos Georgomanolis |
| 1.0 | 29/02/2016 | Final version | Spiros Athanasiou |
| 1.1 | 01/06/2016 | Corrected Figure references – updated report template for MS Office 365/PDF style incompatibility | Spiros Athanasiou |
| 1.2 | 08/07/2016 | Updated all sections and introduced additional sub-sections following reviewers' feedback received during the Y2 review meeting; document re-authored as a Report deliverable | Yannis Kouvaras, Michalis Alexakis, Nikos Georgomanolis, Stelios Manousopoulos, Nikos Karagiannakis, Giorgos Giannopoulos, Pantelis Chronis, Sofia Karagiorgoy, Giorgos Chatzigeorgakidis, Spiros Athanasiou |

# Author list

| organization | name | contact information |
|---|---|---|
| ATHENA RC | Spiros Athanasiou | spathan@imis.athena-innovation.gr |
| ATHENA RC | Nikos Georgomanolis | ngeorgomanolis@imis.athena-innovation.gr |
| ATHENA RC | Yannis Kouvaras | jkouvar@imis.athena-innovation.gr |
| ATHENA RC | Michalis Alexakis | alexakis@imis.athena-innovation.gr |
| ATHENA RC | Stelios Manousopoulos | smanousopoulos@imis.athena-innovation.gr |
| ATHENA RC | Nikos Karagiannakis | nkaragiannakis@imis.athena-innovation.gr |
| ATHENA RC | Giorgos Giannopoulos | giann@imis.athena-innovation.gr |
| ATHENA RC | Pantelis Chronis | pchronis@imis.athena-innovation.gr |
| ATHENA RC | Sofia Karagiorgoy | karagior@imis.athena-innovation.gr |
| ATHENA RC | Giorgos Chatzigeorgakidis | gchatzi@imis.athena-innovation.gr |

# Executive Summary

This report presents an overview of the Prototype Deliverable D1.3 "Beta DAIAD integrated prototype", which includes all DAIAD software components currently under evaluation in the context of our user trials.

In Section 1, we present the software development, integration and testing methodologies applied in the project, emphasizing the organizational and technical details that strongly influence integration tasks. In the following we analyze how our methodologies are applied in practice to integrate and test the DAIAD system in its entirety. The reader is invited to visit the corresponding Annexes which provide more detailed information about our practices.

In Section 2, we present the actual Beta DAIAD Integrated System, as it is currently deployed in a production setting. First, we provide an overview for its architecture and detail its major components. Next, we enumerate the major libraries and frameworks applied in the system. Last, we present how documentation is produced and provide links to the actual documentation of the system.

In Section 3, we present the production system applied to deploy DAIAD. First, we briefly present its capabilities and characteristics, as well as its initialization for the needs of the project. Then, we present the actual production architecture in terms of cloud VMs/nodes/groups. Finally, we present how deployment is orchestrated and automated.

# Abbreviations and Acronyms

| | |
|---|---|
| API | Application Programming Interface |
| AJAX | Asynchronous JavaScript and XML |
| AOP | Aspect Oriented Programming |
| APK | Android application package |
| AWS | Amazon Web Services |
| BT | Bluetooth |
| CI | Continuous Integration |
| CORS | Cross-Origin Resource Sharing |
| CRON | Command Run On |
| CSRF | Cross-Site Request Forgery |
| CSS | Cascading Style Sheets |
| CSV | Comma Separated Values |
| DI | Dependency Injection |
| DOM | Document Object Model |
| DOM | Document Object Model |
| DTO | Data Transfer Object |
| DTO | Data Transfer Object |
| ETL | Extract, Transform and Load |
| HDFS | Hadoop Distributed File System |
| HTTP | Hypertext Transfer Protocol |
| ICU | International Components for Unicode |
| IPv4 | Internet Protocol version 4 |
| IPv6 | Internet Protocol version 6 |
| Java EE | Java Enterprise Edition |
| JPA | Java Persistence API |
| JSON | JavaScript Object Notation |
| JTS | Java Topology Suite |

| | |
|---|---|
| MR | MapReduce |
| MVC | Model View Controller |
| MVP | Minimum Viable Product |
| OGC | Open Geospatial Consortium |
| OOXML | Open Office Extensible Markup Language |
| OpenGIS | Open Geographical Information System |
| ORM | Object Relational Mapper |
| RERO | Release Early, Release Often |
| REST | Representational State Transfer |
| RF | Radio Frequency |
| RPC | Remote Procedure Call |
| SFS | Simple Feature Specification |
| SFTP | SSH File Transfer Protocol |
| SPA | Single Page Application |
| SQL | Structured Query Language |
| SSH | Secure SHell |
| SSH2 | Secure SHell 2 |
| SWM | Smart Water Meter |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| UI | User Interface |
| URL | Uniform Resource Locator |
| UTC | Coordinated Universal Time |
| VM | Virtual Machine |
| WGS84 | World Geodetic System 1984 |
| YAML | Yet Another Markup Language |

# Table of Contents


1. Integration ..... 12

1.1. Software Development principles ..... 12

1.1.1. Agile ..... 12

1.1.2. Release Early, Release Often (RERO) ..... 13

1.1.3. Benevolent dictatorship and tight commit control ..... 13

1.2. Integration and Testing ..... 15

1.2.1. Software integration principles ..... 15

1.2.2. Integration and Testing Methodologies ..... 15

1.3. Integration and Testing Practices ..... 16

1.3.1. Usage workloads ..... 16

1.3.2. Infrastructures ..... 17

1.3.3. End-user devices and equipment ..... 17

1.3.4. Staging environment ..... 19

1.3.5. Production environment ..... 19

1.3.6. Tester groups ..... 19

1.3.7. Testing Data ..... 20

2. Beta DAIAD Integrated System ..... 21

2.1. Architecture ..... 21

2.1.1. DAIAD Services ..... 24
2.1.2. DAIAD@utility Web Application ..... 28
2.1.3. DAIAD@commons Web Application ..... 30
2.1.4. DAIAD@home Web Application ..... 31
2.1.5. DAIAD@home Mobile Application ..... 31
2.2. Libraries and Frameworks ..... 33
2.2.1. Spring ..... 33
2.2.2. Apache Log4j2 ..... 34
2.2.3. Joda-Time ..... 35
2.2.4. ICU - International Components for Unicode ..... 35
2.2.5. JTS Topology Suite ..... 35
2.2.6. Hibernate ..... 35
2.2.7. Jadira Framework ..... 35
2.2.8. Apache POI ..... 36
2.2.9. GeoTools ..... 36
2.2.10. Flyway ..... 36
2.2.11. JSch – Java Secure Channel ..... 36
2.2.12. React ..... 36
2.2.13. Redux ..... 36
2.2.14. React-Router-Redux ..... 37
2.2.15. React-Bootstrap ..... 37
2.2.16. Moment ..... 37
2.2.17. ECharts ..... 37
2.2.18. Flot ..... 37
2.2.19. Apache Cordova ..... 38
2.3. Documentation ..... 38

2.3.1. Introduction ..... 38
2.3.2. Tools ..... 38
2.3.3. JSDoc ..... 40
3. Production Deployment ..... 42
3.1. Introduction ..... 42
3.2. Synnefo ..... 42
3.3. Production Architecture ..... 44
3.3.1. Deployment Orchestration ..... 46
3.3.2. Software versions ..... 46
4. Annex: HBase Schema ..... 47
4.1.1. amphiro-measurements-v2 ..... 47
4.1.2. amphiro-sessions-by-time-v2 ..... 47
4.1.3. amphiro-sessions-by-user-v2 ..... 48
4.1.4. meter-measurements-by-time ..... 48
4.1.5. meter-measurements-by-user ..... 48
4.1.6. meter-forecast-by-time ..... 48
4.1.7. meter-forecast-by-user ..... 48
5. Annex: PostgreSQL Schema ..... 49
5.1. System Database ..... 49
5.1.1. daily_counter ..... 49
5.1.2. scheduled_job ..... 50
5.1.3. scheduled_job_parameter ..... 50
5.1.4. upload ..... 51
5.2. Application Database ..... 52

5.2.1. utility ..... 52
5.2.2. account ..... 52
5.2.3. role ..... 53
5.2.4. survey ..... 54
5.2.5. account_profile ..... 54
5.2.6. cluster ..... 55
5.2.7. group ..... 55
5.2.8. group_member ..... 56
5.2.9. device ..... 56
5.2.10. log4j_message ..... 56
5.2.11. account_alert ..... 57
5.2.12. alert_translation ..... 57
6. Annex: Benchmarking Environment ..... 59
7. Annex: Staging Environment ..... 60
8. Annex: Software versions ..... 61
9. Annex: Usage Workloads ..... 63
10. Annex: Mobile devices ..... 77
11. Annex: Testing Data ..... 80
11.1. Smart Water Meter Data ..... 80
11.1.1. Data Format ..... 80
11.1.2. Historical SWM Data (1K SWMs) ..... 80
11.2. Amphiro data ..... 81

11.2.1. Amphiro historical (historical extractions) ..... 82
11.2.2. Amphiro data (real-time) ..... 83
11.3. Synthetic data ..... 84
11.3.1. Synthetic data for scalability testing ..... 84
11.3.2. Synthetic data for integration and usability testing ..... 85
11.3.3. Data import ..... 86
12. Annex: Arduino prototype ..... 88

# 1. Integration

## 1.1. Software Development principles

In this section we provide a brief overview of the principles and methodologies followed in the project for software development, integration, testing, and release management during its various stages. In summary, we employ an Agile approach and practices and the RERO (Release Early, Release Often) paradigm. Our methodological framework has been shaped both from our experiences in software development for open source projects (e.g. GeoKnow, PublicaMundi), as well as the particular requirements of DAIAD in terms of complexity, effort, and time constraints.

### 1.1.1. Agile

In the following we highlight specific Agile[1] practices employed in the project and when required, the appropriate context for each one.

- **Pair-programming**. Software developers and researchers (PhD candidates, Research Associates) spend 20-50% of their time (depending on the actual task/functionality and importance) in pair-programming. This approach is also applied during internal testing (unit/integration) to speed-up the discovery and redress of bugs/issues.
- **Software that works**. Our emphasis is placed on delivering actually working software, deployed and tested on a production setting. This is sound both from a methodological perspective, but also absolutely needed to address the complexity of the complete DAIAD system. In contrast to stand-alone tools or server-side software, DAIAD comprises proprietary hardware components, firmware, RF protocols, APIs, mobile apps, web apps, server-side software, various database engines, with all being deployed on a production cloud environment.
- **Continuous change and development**. New or changed user requirements are continuously collected from stakeholders and transferred in the development cycle, ensuring quick respond to change. Once again, this approach is required due to the novelty of DAIAD; it is the first complete integrated software for demand-side water management[2]. This means that both expert users (water utility, policy makers) and every-day users (water consumers) deliver new ideas and needs in a very frequent manner. While a formal user requirements analysis has been performed in Report Deliverable D1.2 'DAIAD Requirements and Architecture', a plethora of new requirements were suggested following the release of our first integrated prototype (MVP[3] – M13) and especially after the release of DAIAD software within the Consortium for internal testing (M16).

[1] http://www.agilemanifesto.org/

[2] Direct DAIAD competitors on a worldwide setting include: Dropcountr (mobile/web app for consumers), Omni-Earth (remote sensing-based analysis of water budget for irrigation), and WaterSense (mobile/web app for consumers, utility back-end only for consumer engagement).

[3] http://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html

### 1.1.2. Release Early, Release Often (RERO)

The RERO paradigm has its foundations deeply rooted in open source projects[4] (e.g. Linux kernel), focusing on maintaining momentum in development, as well as accelerating feedback received from end-users and testers. Its opposite is a strict and feature-based release schedule (i.e. release only if a new functionality is complete). Once again, the aforementioned characteristics of the DAIAD system and its development time-frame, favored rapid vs. infrequent user feedback (e.g. weekly vs. quarterly). The implementation of RERO in the project focuses on short (1-3 weeks) development and testing cycles (i.e. sprints) across all various system components (e.g. APIs, libraries, UI elements). The only exception to RERO applies to the development of embedded DAIAD software (i.e. hardware firmware), which needs to follow a stricter testing/release schedule according to assembly and manufacturing constraints.

### 1.1.3. Benevolent dictatorship and tight commit control

DAIAD is an open source project since its inception as a project proposal. Our motivation for this direction is analyzed in the Annex I (DoW) of our Grant Agreement; in a nutshell, we want to accelerate the uptake of personal water monitoring technologies from consumers and facilitate water utilities to harness innovations beyond their current reach. The governance and commit/attribution models in open source projects[5,6] vary greatly depending on their foundations, age, and popularity. With DAIAD being a very new open source project, focused on an R&D-centric agenda, with zero external contributions, as well as a very tight timeframe for developing, testing, and validating its output, we adopted the following governance and collaboration directions.

- *Benevolent dictator*. Software development is steered by a single person (Spiros Athanasiou) with exclusive control over the directions and course of the software. With no external contributors, this is the preferred governance model for young open source projects (especially during their incubation phase). Before the end of the project, and in the context of our joint exploitation plans to ensure the sustainability of DAIAD as an open source project (see Annex I for details) the governance model will change to PSC-based (Project Steering Committee).
- *Commit Control*. The roles of code authors vs. committers are typically used interchangeably in open source development, but they are not actually the same. Commit roles/rights are a matter governance (commit access), policy (public attribution as incentives for participation) and technology (svn/cvs vs. git/mercurial repos). In the project, we have opted for the following three phases of commit control:
    - M6 to M23. During this phase code authors submit contributions to three (3) Software Leads (Nikos Georgomanolis for mobile software, Yiannis Kouvaras for server-based software, Samuel Schlob for embedded software). Each lead is responsible for receiving, reviewing, improving himself, or requesting further improvements from the authors. If the Leads are satisfied, the source code is committed by them in our public repository (exceptions are embedded software and interventions – see below for details). This approach was adopted due to the strict timeframe of the project for delivering its first beta integrated prototype, the

[4] http://firstmonday.org/article/view/578/499
[5] http://oss-watch.ac.uk/resources/governancemodels
[6] http://producingoss.com/

high number of developers involved and their diverse domains (e.g. mobile, embedded), as well as the highly complex nature of the DAIAD as a complete system.

  - M24-M38. After the first beta integrated version of DAIAD is successfully deployed and used during our 12month trials in a production setting, code authors gain commit access to our public repo (same exceptions as before). This marks the graduation of the complete DAIAD system as a stable software, and allows us to focus on validation, our data-driven R&D work, and the continuous improvement of the system.
  - M39 – M42. This is the last period of the project, during which we will actively invite the community of developers to download and contribute to DAIAD. At this point, governance and commit control of the project will gradually be released to the community following a meritocratic-based model.

- Source code publication. All software developed from DAIAD is available as open source software, with the following two exceptions:
  - Embedded software. The software developed for the amphiro b1, as well as all work related to WP2 are confidential, with IPR owned by Amphiro (see GA, Annex I).
  - Software affecting validation. Specific software components related to the treatment phases of the WP7 Trials (e.g. interventions, app documentation) will be maintained private until the corresponding treatment phase has ended. This significant detail is unfortunately neglected in other studies and work, contaminating the target/control groups of the study, and thus introducing bias that warrants any validation results void.
- Open data publishing. All data produced by DAIAD during its Trials will be anonymized and provided with an open data license, with the following three exceptions:
  - Historical data. The validation of DAIAD in terms of water savings and changes in consumer behavior will be based on comparing control vs. treatment groups, while also taking into account the *seasonal influence* of water use. With water demand rising by 20-30% in our Trial locations during the summer, this significant detail is also neglected in other studies and works that warrant any results void. This data will not be provided with an open data license as they have not been produced by the project.
  - Third-party data. The early research work and directions of DAIAD have been facilitated by data contributed by AMAEM (historical SWM data) and Amphiro (a1/b1 data) in the context of external studies. These datasets were crucial for our R&D work, as actual detailed water consumption data will be produced during the Trials (M25-M36). Due to legal and contractual constraints with third-parties, these data will not be available with an open data license.
  - Data affecting validation. Datasets related to the treatment phases of the WP7 Trials will be maintained private until the corresponding treatment phase has ended, for the same reasons mentioned above.

## 1.2. Integration and Testing

In the section we present our methodology and practices for integrating and testing all software developed in the project.

### 1.2.1. Software integration principles

According to the DAIAD Architecture developed in D1.2 'DAIAD Requirements and Architecture', we have identified the following integration approaches for the various software components of the complete DAIAD system. These have been influenced from (a) our ambition to maximize the opportunities of our software to be repurposed/extended as whole, as well as individual components, (b) the need to ensure scalability and fault tolerance, (c) minimize interdependencies and potential delays in the development process, and (d) practical considerations regarding the different platforms and ecosystems DAIAD embraces.

- **RPC API***s*. The majority of DAIAD software exposes and consumes RPC APIs (HTTP-based) to facilitate performance, scalability, and reuse. These APIs encapsulate and provide access to data management, query, and visualization services, powering web and mobile applications.
- **Loosely coupled**. Lower-level libraries (e.g. visualization) and algorithms (e.g. time-series forecasting) are packaged and reused as loosely coupled components, allowing their continuous improvement and reuse without affecting the operation of other system components.
- **No system-wide dependencies**. In contrast to a monolithic system, we avoid system-wide dependencies, ensuring the major system components are self-sustained and independent from each other. This approach accelerates development, increases fault-tolerance, and allows rapid experimentation and testing of new ideas.
- **Embedded software**. The only exception to the above principles concerns the integration of embedded software, which by definition is tightly integrated (low-level programming) to the underlying hardware components (micro-controllers, firmware). However, even in this case, communication and integration with other DAIAD components is still API-based to simplify development and maximize reuse.

### 1.2.2. Integration and Testing Methodologies

In accordance to the software development principles adopted in the project and presented in the previous section, our integration and testing methodologies are as follows.

- **Continuous integration**. Source code contributions are integrated and tested at very frequent intervals (from twice a day, to every couple of days). Due to the API-based and loosely-coupled nature of the DAIAD system, CI is often performed interdependently across its various components. For example, CI for the mobile applications can be performed as often as needed, since communication with the server back-end and the DAIAD hardware (providing the APIs) is API-based (which is very stable).
- **Integration testing**. We have adopted the following two methodologies for integration testing to accelerate development and ensure scalability.
    - *Big Bang*. This testing process is applied for the DAIAD system as a whole, following specific usage workloads that cover all usage aspects of the system, testing infrastructure, tester

groups, and data (for details see next section and Annex: Usage Workloads, Annex: Mobile devices, Annex: Testing Data).

- *Risky-hardest*. System APIs, which by definition are slowly changing and affect the entire system operation, are additionally tested first using the Risky-hardest methodology. When a new API version is available, providing a solution to a previously identified problem or new functionality, integration testing begins by focusing on the API itself.

**Staging and Production testing**. For development, integration, and testing purposes we use a number of computing infrastructures presented in detail in Section 3, Annex: Benchmarking Environment, and Annex: Staging Environment.

- Integration testing is performed in our Staging environment, which replicates the Production environment in terms of software components and capabilities. Further, we exploit the Testflight (iOS-specific) and Beta Testing (Android-specific) facilities for publishing and testing our mobile applications in our internal tester groups. During the last half of Year 2, integration testing on the actual Production environment was prioritized for the following reasons:
  - During M16 we discovered that several functionalities successfully tested in the Staging Production environment, were not properly functioning in the Production environment. All of them were related either to hardware (e.g. manufacturing tolerances, firmware problems), or mobile-devices (e.g. Bluetooth stack implementation) and could not have been replicated in the Staging environment (e.g. device pairing problems, out of sequence data). As a result, we opted to roll-out new software in the Production environment at much more frequent intervals, introduce certain facilities for troubleshooting hardware-centric issues on the production system, and realign our internal tester group to use the actual production system (see next section for details).
  - The coordination, implementation, and monitoring of the pilot was performed from M18 using the DAIAD Production environment and its incorporated facilities for treatment testing. This planned activity in the context of WP7 necessitated the rapid discovery, troubleshooting, and redress of system issues (e.g. unsupported mobile devices), which were then integrated and tested on the production environment.

## 1.3. Integration and Testing Practices

In this section we present how our testing and integration methodologies are applied in practice for producing the integrated DAIAD system. We first present in the usage workloads applied for integration and testing, the various infrastructures and services, our testing groups, and data used. More detailed information is provided in the corresponding Annexes and referenced sections of this document.

### 1.3.1. Usage workloads

Integration testing is performed based on a scripted list of user workloads grouped around the five interfaces of the complete DAIAD system: DAIAD@know in-situ interventions (i.e. amphiro b1), DAIAD@home (mobile application, iOS/Android), DAIAD@home (web application), DAIAD@commons (web application),

DAIAD@utility (web application). The corresponding usage workload scripts are provided in Annex: Usage Workloads and are updated constantly to reflect to new/adapted functionalities introduced during the course of the project.

## 1.3.2. Infrastructures

Integration and testing for DAIAD comprises several infrastructures, devices, and equipment. In the following we provide a brief summary for each one, its purpose of use, as well as additional information in the corresponding Annexes and referenced sections of this document.

## 1.3.3. End-user devices and equipment

DAIAD's entry points for consumers and users are mobile devices (iOS/Android mobile/tablets) and web browsers. Such heterogeneity in supported devices, the end-to-end complexity of the complete DAIAD system, and the requirement to validate DAIAD by actual consumers in a real-world setting, has resulted in an extremely diverse and rich assembly of equipment for integration and testing.

### 1.3.3.1. Mobile devices

During development we use the available emulators for the iOS and Android operating systems, which include facilities for testing our mobile application in various target operating system's and devices (i.e. tablet/mobile, screen size). However, it is a well-known fact that mobile applications rarely operate as intended in real-world devices due to the heterogeneity in hardware/software, as well as market/carrier-specific localizations. Mobile device farms, such as those offered by Amazon[7], Google[8], and Xamarin[9], allow developers to automatically or manually test their application in hundreds of *remote* devices. Unfortunately, these facilities are not useful for mobile applications focused on RF-based connectivity with external peripherals (e.g. activity trackers), which is the case of DAIAD (Bluetooth 4.0 connectivity). As a result, integration and testing can only happen using *actual mobile devices* and peripherals.

We have assembled a large collection of mobile phones and tablets for the iOS and Android operating systems, which are presented in Annex: Mobile devices. It comprises devices purchased for the needs of the project, devices owned by Consortium members, devices owned by outside parties (friends, family, co-workers), and devices used by Trial participants. These are used as follows:

- Devices owned by the project are used for daily and periodic (1-3 weeks) development, integration and testing based on local builds, beta builds (iOS Testflight, Google Play Beta), and the actual public DAIAD mobile application.
- Devices owned by Consortium members are used for periodic (1-3 weeks) and ad hoc (randomly) integration and testing purposes based on beta builds (iOS Testflight, Google Play Beta), and the actual public DAIAD mobile application.
- Devices owned by outside parties are used for ad hoc (randomly) or specific (e.g. device/OS issue) integration and testing purposes on beta builds (iOS Testflight, Google Play Beta), and the actual public DAIAD mobile application.

[7] https://aws.amazon.com/device-farm/

[8] https://developers.google.com/cloud-test-lab/

[9] https://www.xamarin.com/test-cloud

- Devices used by Trial participants are used only to explore and address device-specific issues, which cannot be replicated in another identical device. This is extremely common for Android devices (especially mid to low-cost) due to market-specific localizations.

For mobile devices that are within physical reach to our developers, testing is performed by connecting them to a workstation and monitoring their status in developer mode. For mobile devices in remote locations we have developed a specific debug mode for the DAIAD mobile application (pressing 10 times on the central gauge), which allows us to receive full debug logs in situ.

Finally, it is worth mentioning that as with all iOS applications, the DAIAD mobile app undergoes exhaustive testing from Apple (App Review program[10]) before each new version of our application is published. These reviews vary in time (2-15 days), are performed on actual DAIAD hardware (a b1 prototype was sent to Apple by M16) and result in feedback that must be addressed before the application is published.

#### 1.3.3.2. Amphiro b1 devices

The system is tested against multiple amphiro b1 devices (28 thus far), which are randomly selected from the assembly line. The purpose of these extensive integration tests is to ensure that any differences in manufacturing (e.g. production tolerances, firmware bugs) are properly identified and addressed. This process is the norm for mobile-device peripherals, as the transition from prototype to manufacturing involves several potential sources for errors.

These devices are tested both in situ (i.e. actual showers) and in the laboratory under controlled conditions (i.e. recirculated water flow in the DAIAD aquarium). The former provides integration testing in conditions that resemble a real-world setting, while the latter offers extensive insights into actual operation (i.e. monitor BT packets). A welcome side-effect of the controlled testing is that devices are used well beyond their operation window (6-10 hours/day of constant operation) and also tested for mean time to failure (MTTF). For example, 4 months of lab testing corresponds to 5-7 years of device use in an average EU household (2.3 people showering every day, 15min/shower). Finally, in very few cases (2 times during Y2) we use the Arduino-based prototypes (see Annex: Arduino prototype) for integration testing and troubleshooting low-level Bluetooth operations.

#### 1.3.3.3. Target Browsers/OSs

The web-based end-points of the DAIAD system are tested against the following target browsers and operating systems (most recent version, unless specified otherwise).

- Mozilla Firefox (Windows 7, Windows 10 Home/Professional, Debian 8, Ubuntu 14.0+, Mac OS X 10.11.2+)
- Google Chrome (Windows 7, Windows 10 Home/Professional, Debian 8.0, Ubuntu 14.0+, Mac OS X 10.11.2+)
- Microsoft Edge (Windows 10 Home/Professional)
- Apple Safari (Mac OS X 10.11.2+)

[10] https://developer.apple.com/app-store/review/

### 1.3.4. Staging environment

The staging environment is used to test the complete DAIAD system before roll-out in the production environment. The major difference between staging and production concerns the DAIAD mobile applications.

- iOS versions are local builds, i.e. not part of the Testflight program nor the actual published application. They are built and installed against target devices physically connected with a development workstation.
- Android versions are local or beta builds, i.e. not the actual published applications. Their installation is performed either by a physical connection with the development workstation or by simply sending application's apk file.

The specifications of the staging environment are provided in Annex: Staging Environment.

### 1.3.5. Production environment

The production environment is used to test the complete DAIAD system in actual production operation. Its full specifications are provided in Section 3.

### 1.3.6. Tester groups

Integration and testing is performed by several tester groups comprising core testers, internal testers, invited testers, and actual trial participants. For each group we summarize their tasks and context of participation within integration and testing.

- Core testers. This group includes the actual system developers (6-10 individuals) which integrate and test the DAIAD system following the practices and methodologies analyzed in this section.
- Internal testers. This group includes Consortium members (15-18 individuals) not directly involved in the development process. All of them have DAIAD hardware and software installed in their households and use new system versions (even unstable) both during their everyday lives (i.e. homes) and in a laboratory setting (i.e. follow usage workflows).
- Invited testers. This group involves members outside the Consortium members (10-15 individuals) not directly involved in the project. DAIAD hardware and software is distributed to them on an ad hoc basis for 2-4 weeks and they have access to the latest public version of the system (i.e. Production environment).
- Trial participants. This group comprises Trial participants (8 thus far) which have experienced issues on the Production system that could not be replicated. In case our WP7 Helpdesk cannot resolve an issue reported by a user, a follow-up communication attempts to extract further information (e.g. specific OS/device/carrier) in order to replicate the problem. If this is not possible, we then request the user's assistance and guide him (email or skype) on turning on the hidden debug mode.

### 1.3.7. Testing Data

In this subsection we provide a brief overview of the data used for integration and testing purposes. For each data set we highlight its purpose of use and characteristics, with more information being available in Annex: Testing Data.

- *Historical SWM Data (1,000 meters/12 months)*. This dataset includes water consumption time series (1h readings) for 1,000 smart water meters (sample of the 100,000 SWMs currently operated from AMAEM) for a one-year period.
- *Amphiro data (historical extractions)*. This dataset includes shower extraction events produced from a1/b1 devices in the context of external studies.
- *Amphiro data (real-time extractions)*. This dataset includes real-time shower extractions produced from b1 devices from the Consortium.
- *Synthetic data*. These datasets are produced on demand to deliver SWM and/or b1 testing data for an arbitrary number of users and time periods.

# 2. Beta DAIAD Integrated System

In this section we present an overview of the current Beta DAIAD integrated system, which is deployed in a production setting and validated in real-world conditions in the context of WP7. First, we revisit its architecture and technological foundations. Then, we present all of its subsystems, as well as libraries and frameworks used for its development. Finally, we present the means by which documentation is produced for the various system components.

## 2.1. Architecture

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 1 and will act as a reference for the DAIAD system presentation that follows. 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.

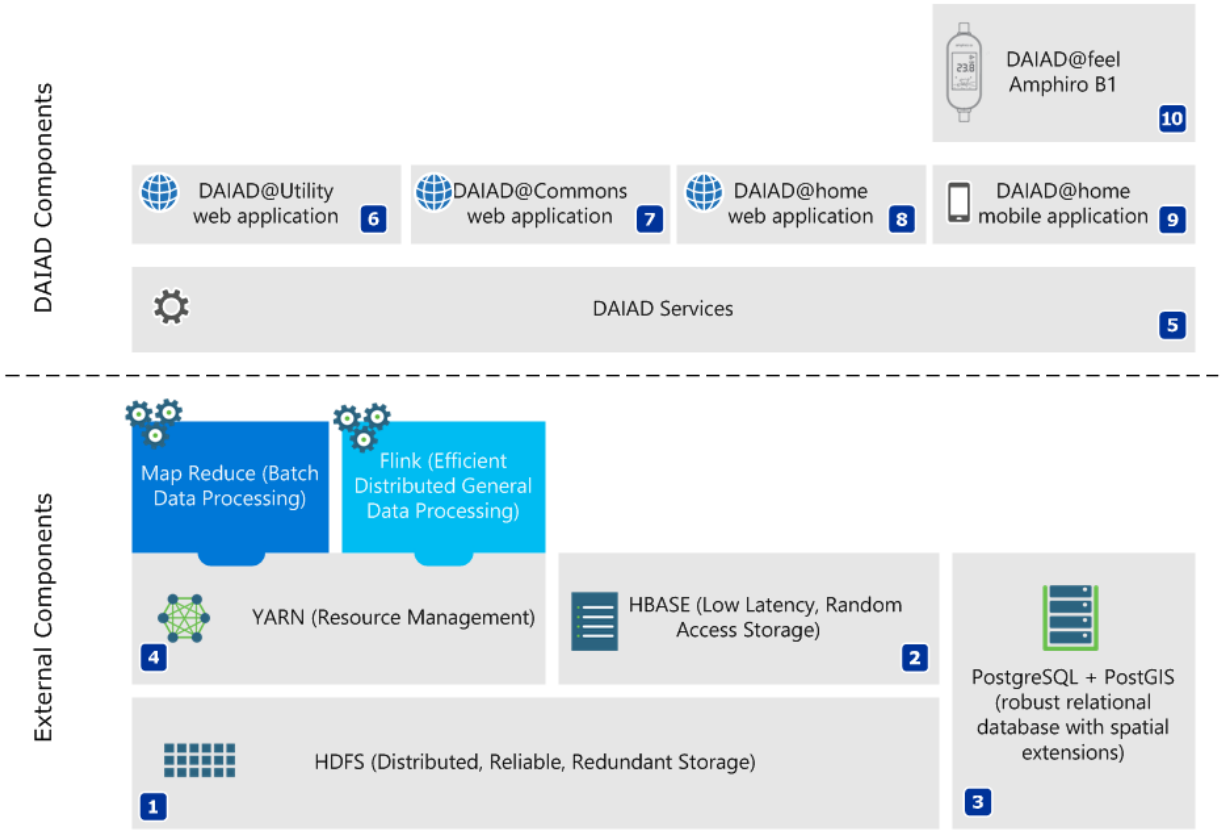



*Figure 1: DAIAD architecture overview*

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. A comprehensive presentation of all HBase tables and their schema is available in Annex: HBase Schema.
- **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. A full description of the database schema is provided in Annex: PostgreSQL Schema.
- 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 2.1.2.5.

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 amphiro b1 device alongside with a display for presenting consumption information to the end user at the point of water extraction.

### 2.1.1. Application Patterns and Design

In the next sub-sections, we present the architecture and provide implementation details for all DAIAD software components. To facilitate understanding, the reader is invited to consider the following:

- All DAIAD software components use additional external libraries and frameworks detailed in section 2.2. During the presentation that follows, we frequently make references to these libraries and frameworks, since they affect the implementation details of each component.
- In DAIAD 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. A short description of the Spring Framework can be found in section 2.2.1.1.

  - **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.

## 2.1.2. DAIAD Services

DAIAD Services implement the core features of the DAIAD functionality. DAIAD services implement the MVC pattern based on the Spring Framework. As a result, the DAIAD components should be thought mostly as loosely coupled components that extend the Spring Framework as depicted in Figure 2. The main components of DAIAD services are enumerated next.

### 2.1.2.1. Controllers

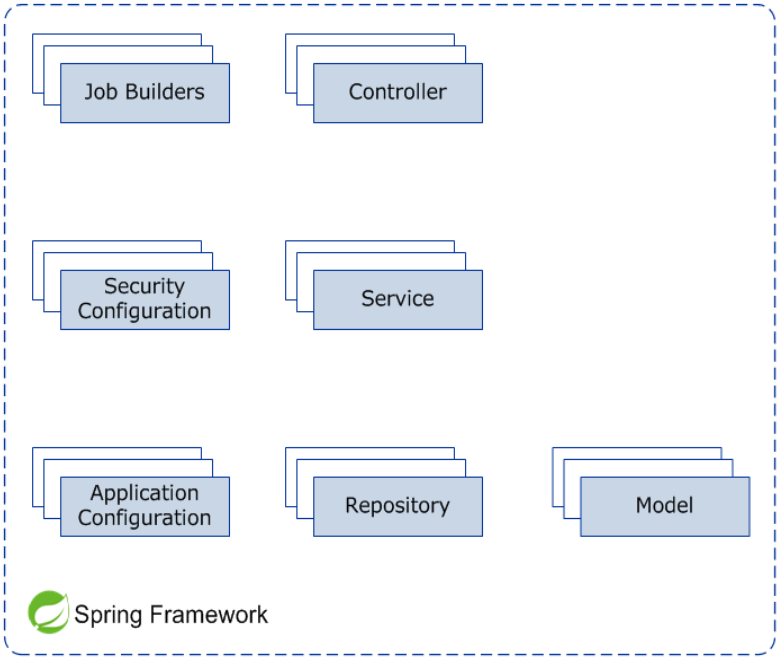



*Figure 2: DAIAD components*

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 features. 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 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.

#### 2.1.2.2. 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.

#### 2.1.2.3. 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.

#### 2.1.2.4. 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.

#### 2.1.2.5. 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.

#### 2.1.2.6. 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.

#### 2.1.2.7. 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.

#### 2.1.2.8. 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. Next we enumerate the job builders that are currently available.

- ConsumptionClusterJobBuilder: Creates a job that clusters users according to their water consumption.
- DailyStatsCollectionJobBuilder: Creates a job for collecting daily statistics for the site such as number of registered users, number of assigned water meters etc.
- MapReduceJobBuilder: Creates a generic job for executing a Map Reduce (MR) job. The MR job being executed is declared using external parameters.
- MessageGeneratorJobBuilder: Creates a job for generating customized alerts and recommendations for all users based on their water consumption.
- 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. For additional details on amphiro b1 data schema see Annex: Testing Data.
- WaterMeterDataSecureFileTransferJobBuilder: Initializes a job for downloading files with smart water meter readings from a SFTP server and importing their contents into HBase.

### 2.1.3. 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 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 2.1.2.1. 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.

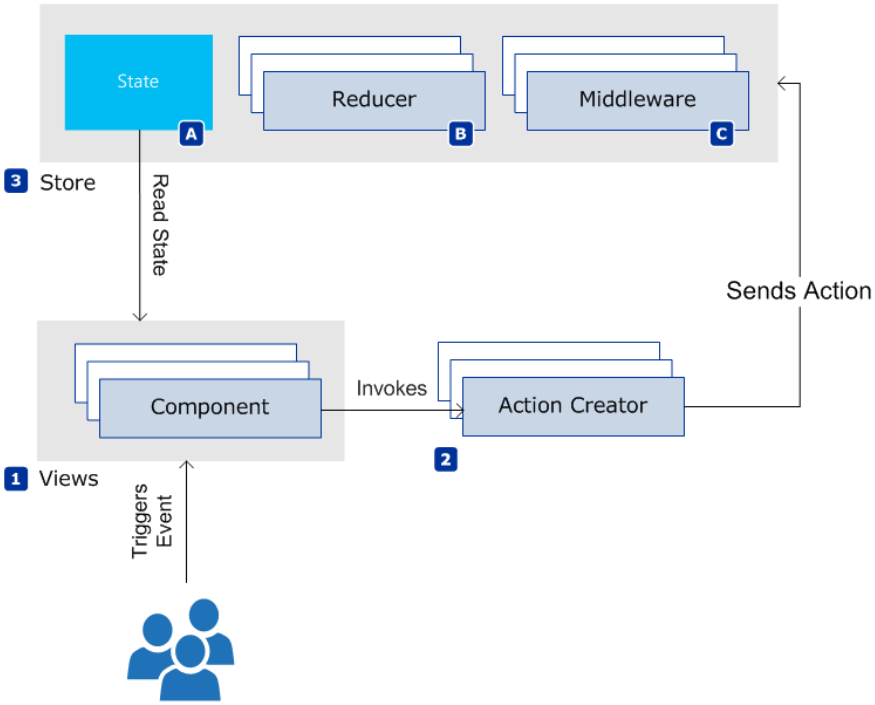



*Figure 3: DAIAD@utility web application architecture based on React and Redux*

The names of the components of the architecture are in sync with the Redux terminology since the whole application is based on it. Next 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 interval 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[11].
- **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[12] and Timeline[13] components. On the contrary, container components have knowledge of Redux and interact with the Store. The application uses three container components, namely, the Root[14], App[15] and ContentRoot[16]. The Root is the top level component of the UI tree hierarchy and is responsible for initializing the

[11] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/store/configureStore.js
[12] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/components/LeafletMap.js
[13] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/components/Timeline.js
[14] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/containers/Root.js
[15] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/containers/App.js
[16] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/containers/ContentRoot.js

Store and making it available to child components. The App provides localization support for the 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[17] middleware in order to handle user navigation and render the appropriate components. The routing configuration is set in the routing[18] 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[19]. Depending on the complexity of the application logic, some reducers are divided even further into child reducers e.g. the Map[20] reducer.

### 2.1.4. 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 a single amphiro b1 device.
- Data visualization from multiple users or groups of users.

The application is designed as a Single Page Application that submits requests to DAIAD services using the Action API described in section 2.1.2.1. Like DAIAD@utility, the application architecture is based on React and Redux JavaScript libraries. In fact, both applications share the same code-base and they are differentiated from each other using authorization and API call parameters as illustrated in Figure 4.

[17] https://github.com/reactjs/react-router-redux
[18] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/routing/routes.js
[19] https://github.com/DAIAD/home-web/tree/master/src/main/resources/public/assets/js/src/utility/reducers
[20] https://github.com/DAIAD/home-web/blob/master/src/main/resources/public/assets/js/src/utility/reducers/map.js

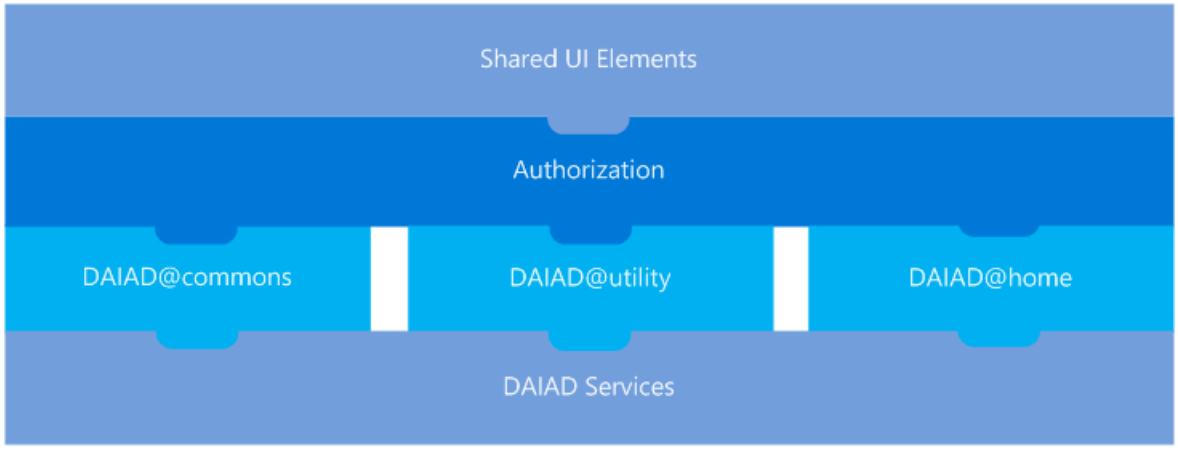



*Figure 4: DAIAD web applications shared architecture*

Users interact with the UI elements that result to Action API calls. The authorization layer controls the features that an authenticated user can access; in this case the DAIAD@commons features. Finally, DAIAD@commons propagates requests to the appropriate DAIAD Services components.

DAIAD@commons can be deployed either as a standalone application or side-by-side with DAIAD@utility and DAIAD@home web applications.

### 2.1.5. DAIAD@home Web Application

DAIAD@home web application provides access to smart water meter and amphiro b1 data at the household level. The application complements the DAIAD@home mobile application and gives users online access to all of their data without the need to download it locally. The most prominent features of DAIAD@home web application are:

- Creating custom dashboards using several UI widgets like charts, tables and labels.
- Browsing historical data over variable time intervals and aggregation levels.
- Viewing insightful comparisons about the user's water consumption behavior.
- Enumerating registered devices (i.e. SWMs and one or more amphiro b1) and viewing their data.
- Receiving alerts, recommendations and tips about saving water.
- Reading useful utility announcements.

DAIAD@home shares the same architecture with DAIAD@commons and DAIAD@utility applications as discussed in sections 2.1.3 and 2.1.4. The corresponding components and function implementing Action Creators, Reducers and UI Components can be browsed at:

https://github.com/DAIAD/home-web/tree/master/src/main/resources/public/assets/js/src/home.

### 2.1.6. DAIAD@home Mobile Application

DAIAD@home mobile application is a hybrid mobile application implemented using the Apache Cordova framework and designed as a Single Page Application. A generic overview of a Cordova application is shown in Figure 5.

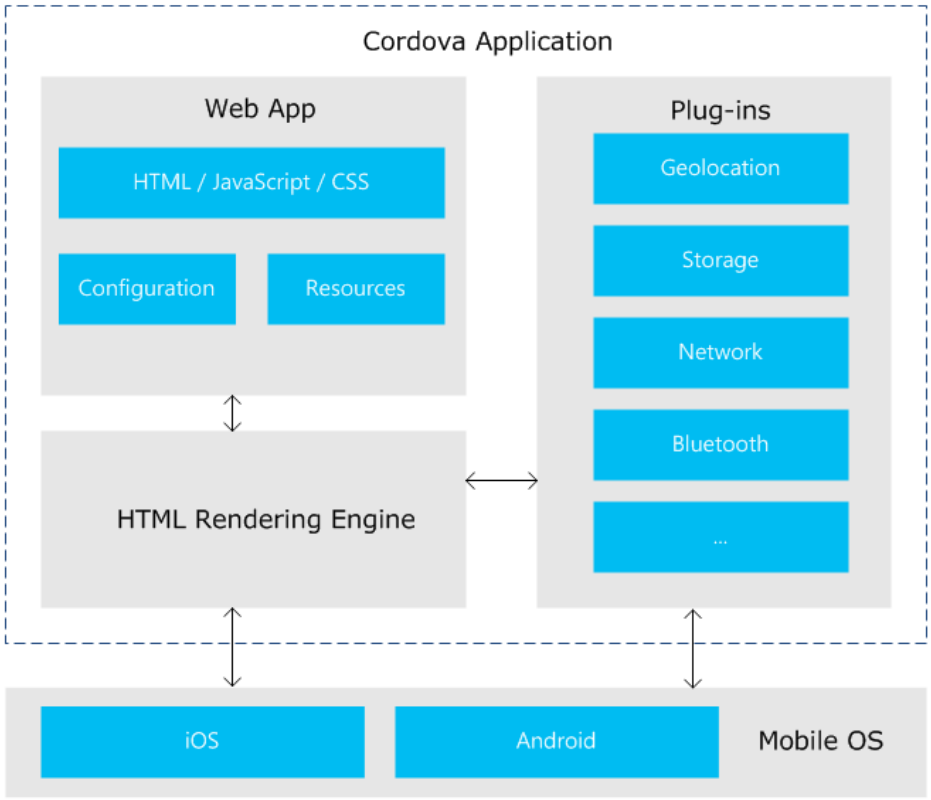



*Figure 5: Cordova Application Architecture*

The Web App block contains the majority of the application code and consists mostly of HTML, JavaScript and CSS files, like any traditional web application. Any required application assets such as image files are also packaged as resources. In addition to the source code and assets, the Web App contains configuration settings for building the application and managing Plug-ins.

Plug-ins, are special extensions of Cordova that provide access to device hardware features such as storage, network connectivity, native UI elements and Bluetooth. The most important plug-ins used by the DAIAD@home mobile application are enumerated below:

- cordova-plugin-ble-central 1.0.5: Access to the Bluetooth low energy (BLE) connectivity
- cordova-plugin-camera 2.1.1: Access to camera
- cordova-plugin-dialogs 1.2.0: Enables OS notifications
- cordova-plugin-file 3.0.0: Provides access to device storage
- cordova-plugin-geolocation 2.1.0: Allows the application to query the user's location
- cordova-plugin-globalization 1.0.3: Add globalization support to the application
- cordova-plugin-inappbrowser 1.2.1: Adds support for windows
- cordova-plugin-network-information 1.0.1: Provides information about device's cellular and Wi-Fi connection status and whether the device has internet access
- cordova-plugin-screen-orientation 1.4.2: Offers control over the device screen orientation

- cordova-sqlite-storage 0.8.2[21]: Add access to SQLite relation database

All the plugins, except for the one of SQLite, are available at the Cordova Plugins repository at https://cordova.apache.org/plugins/.

Once the application is deployed, the HTML Rendering Engine, part of the Cordova runtime, executes the application and enables it to interact with the plugins and the supported mobile OS functionality. Currently, DAIAD@home mobile application supports iOS and Android.

Finally, DAIAD@home mobile application offers almost the same feature set as the DAIAD@home web application with two distinctive differences. It controls amphiro b1 devices using the Bluetooth plugin and stores all water consumption data locally. Accessing data that is not available, such as smart water meter data, requires a connection to DAIAD Services using the Network plugin.

## 2.2. Libraries and Frameworks

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

### 2.2.1. Spring

Spring[22] 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.

#### 2.2.1.1. Spring Framework

The Spring Framework[23] 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.

[21] https://github.com/litehelpers/Cordova-sqlite-storage
[22] https://spring.io/
[23] http://projects.spring.io/spring-framework/

- Spring MVC web application and RESTful Web Service framework for easily building web application and services.

#### 2.2.1.2. Spring Batch

Spring Batch[24] 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.

#### 2.2.1.3. Spring Security

Spring Security[25] 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.

#### 2.2.1.4. 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[26] 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 default values are not appropriate to the requirements of the application, they can easily be replaced with custom configuration options.

### 2.2.2. Apache Log4j2

Every application requires some sort of logging in order to support tracing and debugging. Apache Log4j2[27] 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.

[24] http://projects.spring.io/spring-batch/
[25] http://projects.spring.io/spring-security/
[26] http://projects.spring.io/spring-boot/
[27] http://logging.apache.org/log4j/2.x/

### 2.2.3. Joda-Time

Prior to Java 8, date and time class features were too restricted. Joda-Time[28] 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.

### 2.2.4. ICU - International Components for Unicode

ICU[29] 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.

### 2.2.5. JTS Topology Suite

JTS Topology Suite[30] 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[31] (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.

### 2.2.6. Hibernate

Hibernate[32] 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[33] (JPA) specification, thus allowing seamless integration to Java EE application servers. Hibernate uses several techniques such as lazy loading and optimistic locking in order to achieve high performance.

### 2.2.7. 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[34] 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.

---

[28] http://www.joda.org/joda-time/
[29] http://site.icu-project.org/
[30] http://www.vividsolutions.com/jts/JTSHome.htm
[31] http://www.opengeospatial.org/standards/sfs
[32] http://hibernate.org/
[33] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
[34] https://docs.jboss.org/hibernate/orm/current/javadocs/org/hibernate/usertype/UserType.html

### 2.2.8. Apache POI

Apache POI[35] 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.

### 2.2.9. GeoTools

GeoTools[36] 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.

### 2.2.10. Flyway

Flyway[37] 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.

### 2.2.11. JSch – Java Secure Channel

JSch[38] is a Java implementation of SSH2[39]. 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.

### 2.2.12. React

React[40] 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.

### 2.2.13. Redux

Redux[41] 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

[35] https://poi.apache.org/
[36] http://www.geotools.org/
[37] https://flywaydb.org/
[38] http://www.jcraft.com/jsch/
[39] https://en.wikipedia.org/wiki/Secure_Shell#Version_2.x
[40] https://facebook.github.io/react/
[41] http://redux.js.org/

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.

### 2.2.14. 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[42] library. The latter provides support for keeping only the UI in sync with the URL.

### 2.2.15. React-Bootstrap

React-Bootstrap[43] is a library of reusable UI components for the React framework. It offers the look-and-feel of the Twitter Bootstrap[44] 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.

### 2.2.16. Moment

Moment[45] is a JavaScript library for managing date time values that creates a wrapper for the JavaScript standard Date[46] object. It provides an extensive API for parsing, manipulating, comparing and formatting dates. Moment also supports duration objects and internationalization.

### 2.2.17. ECharts

ECharts[47] 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.

### 2.2.18. Flot

Flot[48] is a lightweight JavaScript plotting library for jQuery which favors simplicity. When compared to ECharts, the provided feature set is significantly restricted. Flot supports lines, points, filled areas, bars and any combinations of these. Flot's small size makes it ideal for mobile applications.

---

[42] https://github.com/reactjs/react-router
[43] https://react-bootstrap.github.io/
[44] http://getbootstrap.com/
[45] http://momentjs.com/
[46] https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Date
[47] https://ecomfe.github.io/echarts/index-en.html
[48] http://www.flotcharts.org/

### 2.2.19. Apache Cordova

Apache Cordova[49] is a framework for developing hybrid mobile applications. The application is implemented using HTML5, JavaScript and CSS. Cordova offers a set of JavaScript application programming interfaces that allow access to the most common native device APIs including geolocation, storage and network functionality among others. More significantly, these APIs are consistent across multiple platforms, hence, code has to be written only once. In the case that the developer requires access to a native API that is not supported e.g., Bluetooth connectivity, Cordova offers a plugin architecture for implementing JavaScript bindings in order to expose the required native functionality to the JavaScript code.

## 2.3. Documentation

One of the most important aspects of software development is the creation of detailed and consistent code documentation. The existence of well documented code allows the better collaboration between the members of the software development team and parallel development of separate software components that may have dependencies between each other. In this sub-section, we describe the documentation requirements of the project, present the tools used for generating documentation, and finally provide examples and links to the generated documentation for the DAIAD integrated system.

### 2.3.1. Introduction

In the DAIAD project there are four distinct cases where documentation is required. First, project documentation is required in order to easily deploy and configure the DAIAD application in development or production environment. Second, server side code that manages data, job execution and security, requires documentation in order to be accessible and extensible. Likewise, client side code that executes on the browser or mobile devices needs to be documented. Finally, API documentation that allows client side code to interact with server must be well documented in order for the UI developers to access functionality exposed by the DAIAD services.

Depending on the code being documented, several tools have been used for generating documentation. In all cases, the documentation generation process has been integrated in the project build process. Hence, the documentation is updated whenever the code changes. The whole documentation is packaged along with the DAIAD distributable artifacts and becomes accessible once the DAIAD server application has been deployed.

### 2.3.2. Tools

We are using three tools, namely, Javadoc[50], JSDoc[51] and apiDoc[52], for generating documentation for server-side code, client-side code, and API methods signatures respectively. Both Javadoc and JSDoc, generate documentation files from comments inside the source code. This feature makes code self-documented and easier to understand without having to refer to the complete documentation.

[49] https://cordova.apache.org/

[50] http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html

[51] http://usejsdoc.org/

[52] http://apidocjs.com/

For the API documentation, we follow a different approach in which the documentation is separated from the actual implementation code. The reason for this is that the target audience, i.e. the API consumers, is interested only in the public functionality exposed by the API methods. Still, the full implementation documentation exists as part of the server code documentation. Although there are several tools that allow the generation of API documentation from inside the source code such as JSONDoc[53] or Swagger[54], we avoided using them in order to reduce the clutter of additional annotations inside the source code.

For the project documentation, we are using the Apache Maven Site Plugin[55]. The generated artifact is a web site with detailed information on how to deploy and configure the DAIAD web application and services.

Finally, all tools described above have full support for templating, thus allowing the configuration of the layout and styling of the generated documentation. Next we give a brief description of every tool, followed by a simple example and a link to the actual documentation publicly available.

```
/**
 * Web Application entry class.
 */
@SpringBootApplication
@EnableGlobalMethodSecurity(securedEnabled = true)
public class Application extends SpringBootServletInitializer {

    /**
     * Configure the application.
     *
     * @param builder a builder for the application context.
     * @return the application builder.
     */
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(Application.class);
    }

    /**
     * Initializes and starts the application.
     *
     * @param args application arguments.
     * @throws Exception if the initialization of the Spring Application Context fails.
     */
    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }
}
```

*Figure 6: Javadoc example*

### 2.3.2.1. Javadoc

The Javadoc tool is used for generating documentation for Java source code. It does so by parsing the documentation comments from all java source files and produces HTML pages for every package, class and interface. For each class and interface, documentation is generated for public and protected constructors, methods and fields. An example of Javadoc compatible documentation comments is depicted in Figure 6.

The generated documentation of DAIAD is available at:

[53] http://jsondoc.org/

[54] http://swagger.io/

[55] https://maven.apache.org/plugins/maven-site-plugin/

- https:/app.dev.daiad.eu/docs/site/apidocs/eu/daiad/web/Application.html.

### 2.3.3. JSDoc

JSDoc is a documentation generator for JavaScript. Similarly to Javadoc, it scans source code for inline comments and generates an HTML documentation web-site. Moreover, JSDoc allows developers to create pseudo comments in order to generate additional artifacts such as callback signatures and namespaces in the final documentation which may not be part of the actual implementation, but are useful to the developers. A JSDoc comment snipped it displayed next.

```
/**
 * Updates all message options provided and switches to message section
 *
 * @param {Object} options - Contains messages section options to set active message (id, category)
 * @param {String} options.id - Message id to set active
 * @param {Array} options.category - Message category to set active
 */
const linkToMessage = function (options) {
  return function(dispatch, getState) {
    const { id, category } = options;

    if (category) dispatch(setActiveTab(category));
    if (id) dispatch(setActiveMessageId(id));

    dispatch(push('/notifications'));
  };
};
```

*Figure 7: JSDoc example*

The generated documentation of DAIAD is available at:

- https:/app.dev.daiad.eu/docs/client/home/module-MessageActions.html.

#### 2.3.3.1. apiDoc

apiDoc can create API documentation for several languages using annotations either in the source code itself or separate files. Moreover, it supports generating API history which allows developers to easily detect API changes between different versions. Using JavaScript files, we created documentation for all public API methods and the corresponding messages.

The source code for the API documentation can be found at:

- https://github.com/DAIAD/home-web/tree/master/apidoc/src

while the actual generated documentation can be browsed at:

- https://app.dev.daiad.eu/docs/api/index.html.

#### 2.3.3.2. Apache Maven Site Plugin

The Apache Maven Site Plugin generates an HTML web site for the project. In contrast to the aforementioned tools, Site Plugin generates project-wide documentation with information about setting up the DAIAD required databases, configuring application settings, building the DAIAD application from the source code and deploying the application.

The project documentation can be viewed at:

- https://app.dev.daiad.eu/docs/site/index.html.

# 3. Production Deployment

## 3.1. Introduction

The production system of DAIAD is deployed on top of the Synnefo cloud stack[56], within a number of virtual machines. Synnefo is a complete open source cloud stack written in Python that provides Compute, Network, Image, Volume and Storage services, similar to those offered by AWS[57]. On a hardware level, the production system is hosted on Athena RC's own server infrastructure (*120 CPU cores, 92 GB main memory, 10TB storage*), which will be scaled horizontally (*scale-out*) during the course of the WP7 trials depending on system performance and utilization. Figure 8 presents the data center where the physical hardware is hosted.

*Figure 8: DAIAD production servers hosted at the data center of Greek Ministry of Education*

## 3.2. Synnefo

Synnefo manages multiple Ganeti[58] clusters at the backend for handling of low-level VM operations and uses Archipelago[59] to unify cloud storage. To boost third-party compatibility, Synnefo exposes the OpenStack APIs[60] to users. The Cyclades (Synnefo UI) allows administrators to maintain and manage the cloud resources of the system (restart, shut down, start up, delete and create a virtual machine). In Figure 9, an overview of the Synnefo services is presented.

[56] https://www.synnefo.org/
[57] https://aws.amazon.com/
[58] https://code.google.com/p/ganeti/
[59] https://www.synnefo.org/docs/archipelago/latest/
[60] http://developer.openstack.org/api-ref.html

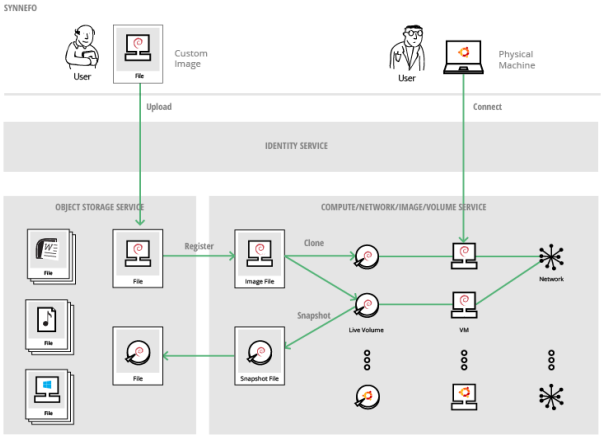



*Figure 9: An overview of the Synnefo services*

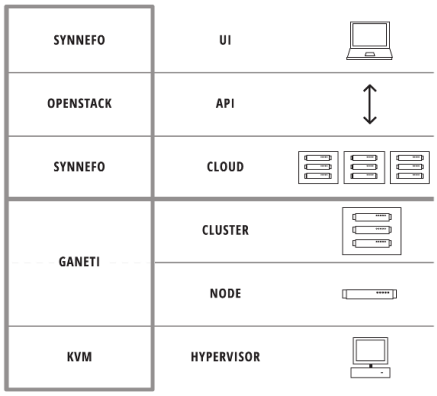



*Figure 10: The system architecture of Synnefo*

Synnefo keeps a clear separation between the traditional cluster management layer and the cloud layer. This unique design approach leads to a completely layered architecture boosting production readiness, maintainability and upgradability. The modular design allows for linear scalability, gradual extensibility and ease of operations. In Figure 11, an overview of the Synnefo architecture is presented, showing the major components on each layer.

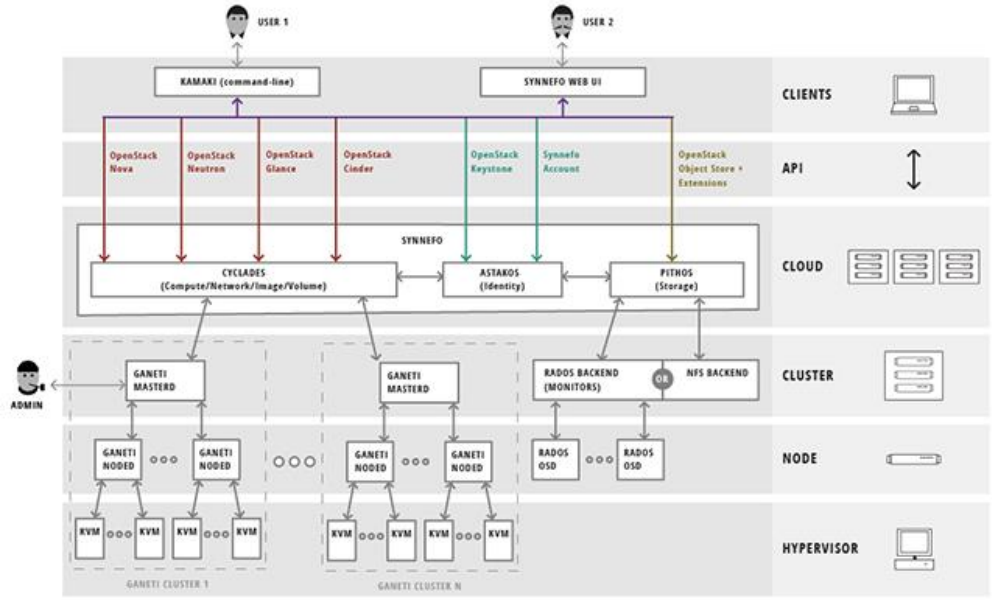



*Figure 11: Detailed description of the Synnefo Architecture*

For all VMs inside the cluster, basic system-wide monitoring is available from the Synnefo layer. The collected statistics can be exported to a table-like format via Synnefo API, or (part of them) can be directly visualized from Synnefo UI (Figure 12).


*Figure 12: Monitoring services for each virtual machine and the network is available from Synnefo deployment*

## 3.3. Production Architecture

Through the Cyclades system (Synnefo UI), we configured a virtual private network, installing Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-44-generic x86_64) virtual machines to host the production system of DAIAD. The beta DAIAD integrated system is deployed as a single cluster (named "c1") divided into 6 virtual node groups - with the provision of spinning up more virtual machines into each group if necessary, through Synnefo and Ansible (see section 3.3.1).

A naming scheme of '*something-cX-nYY*' is used, where X denotes the cluster, and YY denotes a 2-digit identifier for the node group "something". The existing groups are:

- **Administration node** (admin-c1.dev.daiad.eu): This is a single-node group, and acts as a central point for the administration of the entire cluster. It provides terminal access to the internal network of cluster "c1", performs periodic health-checking, maintenance or rolling-update tasks (via Ansible, see section 3.3.1).
- **Data group** (data-c1-nYY.dev.daiad.eu): This node group contains those services that are considered as performing at the data level (e.g. measurements, aggregated results on measurements). It includes HDFS data nodes, YARN node managers, HBase region servers, and Flink task managers. This group typically holds valuable (as possibly irreplaceable) source data, thus services usually operate on a high replication level (~3).

- **Master group** (master-c1-nYY.dev.daiad.eu): This node group contains those nodes that continuously monitor and coordinate the nodes of the "data" group: HDFS name (master/secondary) nodes, HBase master, ZooKeeper.
- **Manager group** (manager-c1-nYY.dev.daiad.eu): This group contains job-managing nodes: YARN resource manager, MapReduce history server, Flink job manager.
- **Application group** (app-c1-nYY.dev.daiad.eu): This group contains nodes that host the web application servers (Tomcat7) and is the only group that has access to the public IPv4 network, as the only one that hosts user-facing (HTTP) services.
- **Relational database group** (rdb-c1-nYY.dev.daiad.eu): The node group that hosts the relational database service (PostgreSQL, enhanced with PostGIS). As mentioned before, the main role of this service is to hold structured user-related data (e.g. location, account info), and is strongly coupled with the "application" group.

All nodes mainly communicate on a private IPv4 network, and of course a certain part of them (*application and administration groups*) are firewalled and exposed to the public IPv4 network. Additionally, IPv6 connectivity is set-up (global addressing) for all nodes. In Figure 13, a snapshot of the private network setup is presented through Synnefo UI.


*Figure 13: The virtual private network setup and IPv6 configuration on Synnefo*

### 3.3.1. Deployment Orchestration

For facilitating the deployment of DAIAD, we created a GitHub repository[61] to store the scripts or configuration files needed for the several node groups. The tool used to automate the deployment/update process is Ansible. The work of parameterizing and completely automating the deployment process is still in progress, as the whole system evolves.

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs. Being designed for multi-tier deployments since day one, Ansible models IT infrastructure by describing how all of the systems inter-relate, rather than just managing one system at a time. It uses no agents and no additional custom security infrastructure, so it's easy to deploy — and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allows administrators to describe their automation jobs in a way that approaches plain English.

Ansible works by connecting to the system nodes and pushing out small programs, called "Ansible Modules" to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished. The library of modules can reside on any machine, and there are no servers, daemons, or databases required. Typically the administrator works with a terminal program, a text editor, and probably a version control system to keep track of changes (e.g. git). Ansible represents what machines it manages using a very simple INI file that puts all of the managed machines in groups. Playbooks can finely orchestrate multiple slices of the infrastructure topology, with very detailed control over how many machines to tackle at a time.

Figure 14 shows the way DAIAD uses Ansible Playbooks in order to deploy software to the target environment, starting from empty Ubuntu 14.04 virtual machines (only network and SSH root access are initially provided from Synnefo). The final result of the application of Ansible Playbooks is the final production environment. In the most common case, software updates are handled using git to perform a fast-forward merge to a predefined branch.

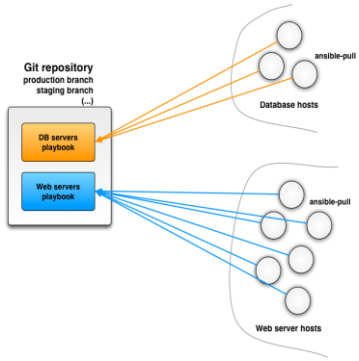



*Figure 14: Ansible playbooks are used in DAIAD to deploy the software stack into the Synnefo virtual servers*

### 3.3.2. Software versions

The current versions for the major external software components comprising the beta DAIAD integrated system are provided in Annex: Software versions.

[61] https://github.com/DAIAD/dev.daiad.eu

# 4. Annex: HBase Schema

DAIAD is using HBase to store water consumption data generated by smart water meters, amphiro b1 devices, and analysis algorithms. For raw smart water meter and amphiro b1 data the application maintains two copies. One is optimized for querying data for a single user/device (i.e. consumer) and the other for querying data over a time interval.

In the next sections we present the HBase tables by displaying the parts that compose their row keys. All tables belong to the *daiad* namespace (not included in the table names for brevity).

### 4.1.1. amphiro-measurements-v2

Table amphiro-measurements-v2 stores time-series for real-time water extraction sessions generated by amphiro b1 devices. The row key structure is shown in Figure 15 and is optimized for searching time series for a specific user and device. For every row, the timestamp, volume, energy and temperature values are stored.

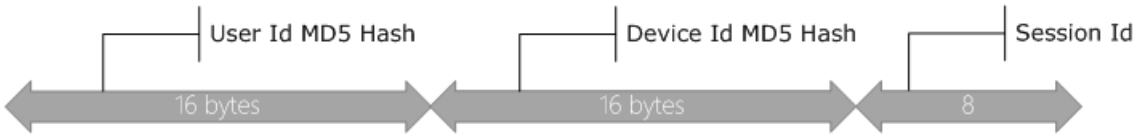



*Figure 15: Table amphiro-measurements-v2 row key structure*

### 4.1.2. amphiro-sessions-by-time-v2

Water extraction sessions (i.e. historical data) generated by amphiro b1 devices are stored in table amphiro-sessions-by-time-v2. The row key structure is optimized for searching sessions for multiple users over a time interval. The time partition is the modulo of the timestamp and the number of the HBase region servers. The timestamp prefix represents the time rounded at the day level. Time partition key section is used for load balancing between the HBase region servers. The timestamp prefix key section is used for compacting multiple sessions inside a single row key.

For every session inside a row key the timestamp, volume, energy, temperature, flow and duration values are stored.

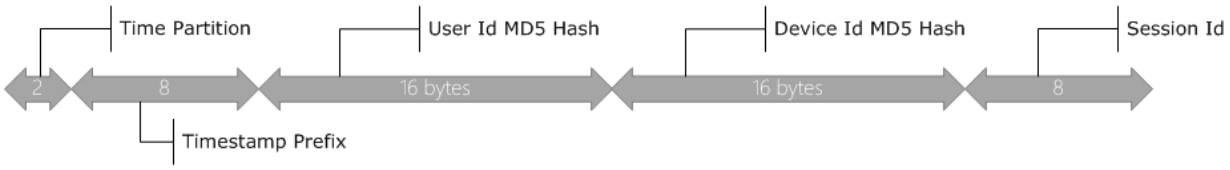



*Figure 16: Table amphiro-sessions-by-time-v2 row key structure*

### 4.1.3. amphiro-sessions-by-user-v2

Similar to amphiro-sessions-by-time-v2 table, the Amphiro-sessions-by-user-v2 table stores amphiro b1 sessions. However, in this case the row key is optimized for querying data for a single user. The corresponding row key structure is the same as the one shown in Figure 15. For each row, the timestamp, volume, energy, temperature, flow and duration values are saved.

### 4.1.4. meter-measurements-by-time

Table meter-measurements-by-time stores smart water meter measurements and is optimized for searching data for multiple users over a time interval. For every measurement the volume and difference since the last measurement are stored. The row key structure is shown in Figure 17.

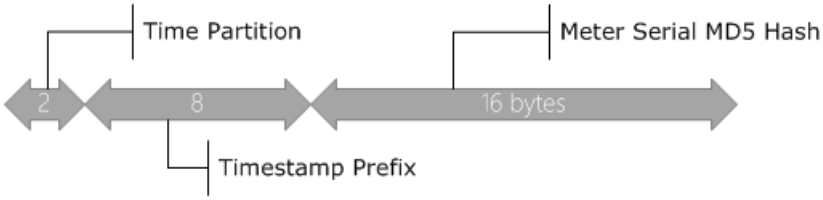



*Figure 17: Table meter-measurements-by-time row key structure*

### 4.1.5. meter-measurements-by-user

Table meter-measurements-by-user stores smart water meter measurements and is optimized for searching data for a single smart water meter device. For every measurement the volume and difference since the last measurement are stored. The row key structure is displayed in Figure 18.

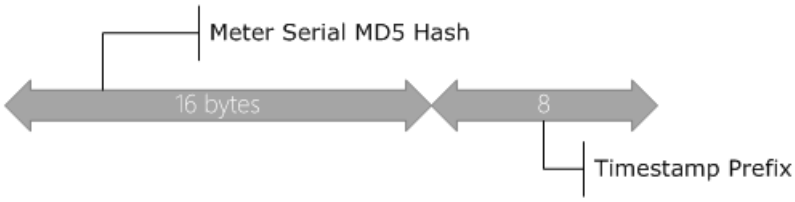



*Figure 18: Table meter-measurements-by-user row key structure*

### 4.1.6. meter-forecast-by-time

Table meter-forecast-by-time is similar to the meter-measurements-by-time table but instead of storing actual measurement data, it stores the results of the forecasting algorithms. The row key has the same structure as in section 4.1.4.

### 4.1.7. meter-forecast-by-user

Table meter-forecast-by-user stores smart water meter data generated by the forecasting algorithms. Its row key has the same structure as in section 4.1.5 and hence it is optimized for querying the data of a single meter.

# 5. Annex: PostgreSQL Schema

In DAIAD we use PostgreSQL for storing frequently changing data, like user accounts and profiles, smart water meter and amphiro b1 registrations, application metadata, cluster management information, etc. In general, PostgreSQL stores tables that contain only a few hundreds of thousands of records. The goal of using a relational database for this type of data is to minimize update and delete operations to HBase.

DAIAD separates data into two database instances. One instance stores user specific information like user accounts and roles. The other one stores system specific information such as scheduled job metadata.

In the next sections we describe the most important tables for every database. The complete schema for both databases expressed in SQL Data Definition Language (DDL) is available at the DAIAD repository[62].

Tables that are generated automatically by 3rd party libraries, i.e. Flyway schema migration tool and Spring Batch Job Repository, are not included in this presentation.

## 5.1. System Database

System database instance is used by the scheduler service for storing data for:

- Job schedule configuration
- Job execution parameters
- Application statistics
- Information about job execution results e.g. uploaded file metadata

### 5.1.1. daily_counter

Stores daily statistics about the DAIAD application runtime. Daily statistics are computed by the job configured in DailyStatsCollectionJobBuilder[63] class.

| Field | Name |
|---|---|
| id | Unique Id |
| utility_id | Reference to utility table |
| date_created | Row creation date |
| counter_name | Counter name |
| counter_value | Counter value |

[62] https://github.com/DAIAD/home-web/tree/master/src/main/resources/db/migration
[63] https://github.com/DAIAD/home-web/blob/master/src/main/java/eu/daiad/web/jobs/DailyStatsCollectionJobBuilder.java

### 5.1.2. scheduled_job

Stores configuration data for scheduler service registered jobs. DAIAD application consults this table about what jobs to schedule and how to create instances of these jobs. Users can schedule jobs for periodic execution based on a fixed interval or a CRON expression. Only one of these options must be specified. Setting a value for both fields will result in a runtime error.

| Field | Name |
|---|---|
| id | Unique job registration Id. |
| category | Job category. Valid values are:<br>MAINTENANCE<br>ETL<br>ANALYSIS<br>FORECASTING |
| container | Execution framework. Valid values are:<br>RUNTIME<br>HADOOP<br>FLINK |
| bean | The name of the class that implements IJobBuilder interface for the specific job. DAIAD application uses this class for creating an instance of the job before execution. |
| name | Unique name of the job. This name is used internally by DAIAD in order to identify a job. |
| description | User friendly job description. |
| data_created | Row creation date. |
| period | Interval in seconds for scheduling the job for periodic execution. |
| cron_expression | A valid CRON expression for scheduling the job. |
| enabled | True if the job must be scheduled; Otherwise False. |

### 5.1.3. scheduled_job_parameter

Stores execution parameters for a registered job. Parameters in this table are passed to the job context before job execution by the scheduler service.

| Field | Name |
|---|---|
| id | Unique Id. |
| scheduled_job_id | Reference to scheduled job table. |
| name | Parameter name. |
| value | Parameter value. |
| hidden | True if the parameter should not be published by the Action API; Otherwise False. |

### 5.1.4. upload

Stores metadata for files downloaded by the job configured by WaterMeterDataSecureFileTransferJobBuilder [64] class.

| Field | Name |
|---|---|
| id | Unique Id |
| source | Remote SFTP server address. |
| remote_folder | Remote folder. |
| local_folder | Local folder. |
| remote_filename | Remote file name. |
| local_filename | Local file name. |
| file_size | Remote file size. |
| date_modified | File last modified date. |
| upload_start_on | Timestamp of download operation start. |
| upload_end_on | Timestamp of download operation end. |
| process_start_on | Timestamp of data process operation start. |
| process_end_on | Timestamp of data process operation end. |
| row_count | Total number of rows in the file. |
| row_processed | Number of rows processed. |
| row_skipped | Number of rows skipped due to validation errors. |

[64] https://github.com/DAIAD/home-web/blob/master/src/main/java/eu/daiad/web/jobs/WaterMeterDataSecureFileTransferJobBuilder.java

## 5.2. Application Database

The application database is used for persisting DAIAD Domain Objects, which include:

- User accounts, roles and profiles
- Registrations for smart water meters and amphiro b1 devices
- Customized alerts, recommendations, tips and announcements for users
- User groups, clusters and cluster segments

### 5.2.1. utility

Stores utility information allowing a single DAIAD installation to support multiple utilities.

| Field | Name |
|---|---|
| id | Unique Id. |
| key | Unique row key. |
| name | Name |
| logo | Optional logo image |
| description | Description |
| date_created | Row creation date |
| default_admin_username | Default administrator name. An account with this username and a random password is automatically created if not already exists on application startup |
| locale | Default locale |
| timezone | Default time zone |
| country | Country name |
| city | City name |

### 5.2.2. account

Stores data about registered users.

| Field | Name |
|---|---|
| id | Unique Id |
| row_version | Row version used by Spring for implementing optimistic row locking |
| name | Parameter name |

| | |
|---|---|
| utility_id | Reference to utility table. |
| key | Unique row key. |
| locale | Default locale. |
| firstname | First name. |
| lastname | Last name. |
| email | Account email address. |
| created_on | Row creation date. |
| last_login_success | Last login success timestamp. |
| last_login_failure | Last login failure timestamp. |
| failed_login_attempts | Number of failed login attempts. |
| change_password_on_login | True if user must change her password after the next successful login operation. |
| locked | True if the user is not allowed to use the application. |
| username | Account username represented by a valid email address. |
| password | Account encrypted password. |
| photo | Optional account image. |
| timezone | Default time zone. |
| country | Country name. |
| city | City name. |
| address | User address. |
| postal_code | User postal code. |
| birthdate | User birth date. |
| gender | User gender. |
| location | User location expressed in WGS84 coordinates. |

### 5.2.3. role

Stores application roles. Roles are used by Spring Security module for performing authorization checks.

| Field | Name |
|---|---|
| id | Unique Id. |
| name | Unique role name. |
| description | Role description. |

### 5.2.4. survey

Contains survey data for users participating in DAIAD trials. Information from this table is used for generating user clusters based on their demographic characteristics.

| Field | Name |
|---|---|
| username | User name. |
| firstname | First name. |
| lastname | Last name. |
| address | Address. |
| city | City. |
| gender | Gender. |
| number_of_showers | Number of showers in the household. |
| smart_phone_os | Smart phone Operating System. |
| tablet_os | Tablet Operating System. |
| apartment_size_bracket | Apartment size. |
| age | Age. |
| household_member_total | Total number of members in the household. |
| household_member_female | Number of female members in the household. |
| household_member_male | Number of male members in the household. |
| income_bracket | Income. |
| meter_id | Smart Water Meter Identifier. |
| utility_id | Reference to utility table. |

### 5.2.5. account_profile

Table for storing account profile information.

| Field | Name |
|---|---|
| id | Unique Id. |
| row_version | Row version used by Spring for implementing optimistic row locking. |
| version | Profile internal version. |
| updated_on | Last modified timestamp. |

| | |
|---|---|
| mobile_mode | DAIAD@home mobile application mode. |
| mobile_config | DAIAD@home mobile application custom configuration options. |
| web_mode | DAIAD@home web application mode. |
| web_config | DAIAD@home web application custom configuration options. |
| utility_mode | DAIAD@utility web application mode. |
| utility_config | DAIAD@utility web application custom configuration options. |
| daily_meter_budget | Daily budget for smart water meter in liters. |
| daily_amphiro_budget | Daily budget for amphiro b1 in liters. |

### 5.2.6. cluster

Stores user clusters that are generated by data analysis algorithms.

| Field | Name |
|---|---|
| id | Unique Id. |
| key | Row key. |
| row_version | Row version used by Spring for implementing optimistic row locking. |
| utility_id | Reference to utility table. |
| name | Unique name. |
| created_on | Row creation date. |

### 5.2.7. group

Represents groups of users. A group may be a cluster segment or a bag of consumers which was manually created by a user.

| Field | Name |
|---|---|
| id | Unique Id. |
| key | Row key. |
| row_version | Row version used by Spring for implementing optimistic row locking. |
| utility_id | Reference to utility table. |
| name | Name |
| created_on | Row creation date. |

| | |
|---|---|
| spatial | Group geometry representation in WGS84. |
| size | Number of group members. |

### 5.2.8. group_member

Table for storing user memberships to groups. A user may belong to more than one groups.

| Field | Name |
|---|---|
| id | Unique Id. |
| group_id | Reference to group table. |
| account_id | Reference to account table. |
| created_on | Row creation date. |

### 5.2.9. device

Represents a generic device registration. A device can be either a smart water meter or an amphiro b1 device.

| Field | Name |
|---|---|
| id | Unique Id. |
| key | Row key. |
| row_version | Row version used by Spring for implementing optimistic row locking. |
| account_id | Reference to account table. |
| registered_on | Device registration date. |
| last_upload_success_on | Last successful data uploading operation timestamp. |
| last_upload_failure_on | Last failed data uploading operation timestamp. |
| transmission_count | Number of data transmissions. |
| transmission_interval_sum | Sum of all data transmission intervals. |
| transmission_interval_max | Maximum interval between two successive data transmissions. |

### 5.2.10. log4j_message

Stores logging messages and allows administrators to easily query logs instead of browsing log data files.

| Field | Name |
|---|---|

| | |
|---|---|
| id | Unique Id. |
| account | Principal name i.e. the name of the authenticated account if any existed when the message was being logged. |
| remote_address | Client remote address. |
| category | Message category e.g. scheduler, data query etc. |
| code | Error specific code e.g. authentication missing credentials error. |
| level | Error level e.g. information, debug, warning, critical etc. |
| logger | Class that logged the message. |
| message | Message. |
| exception | Detailed exception message. |
| timestamp | When message has been logged. |

### 5.2.11. account_alert

Table for account customized alerts. Similar tables exist for recommendations (account_dynamic_recommendation), announcements (account_announcement) and tips (account_static_recommendation).

| Field | Name |
|---|---|
| id | Unique Id. |
| account_id | Reference to account table. |
| alert_id | Reference to alert table. |
| created_on | Row creation date. |
| acknowledged_on | When the user has viewed the alert. |
| receive_acknowledged_on | When the server has been informed that the alert has been viewed by the user. |

### 5.2.12. alert_translation

Table that contains translations for generic alerts. A similar table, dynamic_recommendation_translation exists for recommendations.

| Field | Name |
|---|---|
| id | Unique Id. |
| alert_id | Reference to alert table. |

| | |
|---|---|
| locale | Translation locale. |
| title | Alert title. |
| description | Alert description. |
| link | Optional link to external resources. |

# 6. Annex: Benchmarking Environment

Our dedicated benchmarking environment has been kindly provided by the University of Peloponnese at Tripolis and consists of a Dell PowerEdge M910 Blade Server with the following specifications:

Hardware:

- 4 x Intel® Xeon® CPU E7- 4830 at 2.13 GHz.
    - Each CPU has 8 cores.
    - Supports Hyper-Threading technology.
- 4 x 8 x 2 = 64 threads in total.
- 256 GB (32 x 8GB) RAM.
- 900 GB + 900 GB (mirror) HDD.

Software:

- OS: Linux Debian v3.2.81-1.
- Flink v0.10.2.
- Hadoop v2.7.0.

The pseudo-distributed cluster that we setup is consisted of:

- 1 Flink Job Manager of 8192 MB.
- 4 Flink Task Managers of 32768 MB and 16 slots each.
- Single Node HDFS.
- Single Node YARN.

# 7. Annex: Staging Environment

The staging environment is a mirror of our Production environment (see Section 3) deployed in the same physical hardware infrastructure, where software updates and/or newer cluster configurations are applied and tested before being pushed to production. The difference between staging and production environments is the exact size (i.e., scale) for each node group and, consequently, the replication level at which the cluster operates). So, apart from size, staging closely follows the structure of production environment. In fact, both environments are set-up/updated by the same deployment scripts (Ansible), driven by configuration that only differs in sizing and naming of nodes. Also, both environments follow exactly the same steps to pull application-level software updates (i.e., updates not relevant to cluster's architecture): fetch and fast-forward to code repository's production branch.

The staging environment contains exactly the same node groups acting on the same roles as in production. All nodes follow a similar naming scheme of "something-cX-nYY.stage.daiad.eu":

- Administration node (admin-c1.stage.daiad.eu)
- Data group (data-c1-nYY.stage.daiad.eu)
- Master group (master-c1-nYY.stage.daiad.eu)
- Manager group (manager-c1-nYY.stage.daiad.eu)
- Application group (app-c1-nYY.stage.daiad.eu).
- Relational database group (rdb-c1-nYY.stage.daiad.eu).

All nodes mainly communicate on a private IPv4 network. A certain part of them (application and administration groups) are firewalled and exposed to the public IPv4 network. Additionally, IPv6 connectivity is set-up (global addressing) for all nodes.

By maintaining a staging environment as a "reduced" mirror of production (keeping the so-called parity between stage and production), we facilitate testing & integration and introduce and enable experimentation with architecture-level changes (e.g. new API versions, new configurations that affect the inter-relations of node groups in the whole cluster). In addition, this staging phase (deployment and testing) helps to significantly decrease the possibility of a broken configuration or a bug to appear in the production site, acting as a net that catches a wide range of common mistakes and misconfigurations, before causing any real harm (or even a downtime) on production.

# 8. Annex: Software versions

Ubuntu:

- **Distributor ID**: Ubuntu.
- **Description**: Ubuntu 14.04.4 LTS.
- **Release**: 14.04.
- **Codename**: Trusty.

Java:

- OpenJDK Runtime Environment (IcedTea 2.6.4) (7u95-2.6.4-0ubuntu0.14.04.1).
- OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode).

Apache:

- **Server version**: Apache/2.4.7 (Ubuntu).
- **Server built**: Jan 14 2016 17:45:23.
- Server's Module Magic Number: 20120211:27.
- **Server loaded**: APR 1.5.1-dev, APR-UTIL 1.5.3.
- **Compiled using**: APR 1.5.1-dev, APR-UTIL 1.5.3.
- **Architecture**: 64-bit.
- **Server MPM**: prefork.
    - **threaded**: no.
    - **forked**: yes (variable process count).

Tomcat:

- **Server version**: Apache Tomcat/8.0.33.
- **Server built**: Mar 18 2016 20:31:49 UTC.
- **Server number**: 8.0.33.0.
- **OS Name**: Linux.
- **OS Version**: 3.13.0-44-generic.
- **Architecture**: amd64.
- **JVM Version**: 1.7.0_95-b00.
- **JVM Vendor**: Oracle Corporation.

PostgreSQL:

- PostgreSQL 9.3.11 on x86_64-unknown-linux-gnu, compiled by gcc (Ubuntu 4.8.2-19ubuntu1) 4.8.2, 64-bit.

PostGIS:

- POSTGIS="2.1.2 r12389" GEOS="3.4.2-CAPI-1.8.2 r3921" PROJ="Rel. 4.8.0, 6 March 2012" GDAL="GDAL 1.10.1, released 2013/08/26" LIBXML="2.9.1" LIBJSON="UNKNOWN" TOPOLOGY RASTER

Hadoop:

- Hadoop 2.6.0
- Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r e3496499ecb8d220fba99dc5ed4c99c8f9e33bb1
- Compiled by jenkins on 2014-11-13T21:10Z.
- Compiled with protoc 2.5.0 From source with checksum 18e43357c8f927c0695f1e9522859d6a

HBase:

- HBase 0.98.10.1-hadoop2
- Subversion git://aspire/home/apurtell/src/hbase -r d5014b47660a58485a6bdd0776dea52114c7041e
- Compiled by apurtell on Tue Feb 10 11:34:09 PST 2015.

Flink:

flink-0.8.1-bin-hadoop2-yarn

# 9. Annex: Usage Workloads

amphiro b1

Install

- Unpackage the device and follow the step-by-step instructions included in the box
    - Please report if instructions are unclear at any of their steps
- Install the device in your shower-head using the provided O-rings
    - Please report for any incompatibilities with your shower-head (e.g. you cannot screw it in, water leaks from the connection)
- Turn the water on and wait for the first set of codes to appear for a few seconds
- Turn the water off; you will use the device the next time you take a shower

Shower

- Take a shower as you would normally do and take note of the following
    - Please report if there are any water leaks from the device
    - Please report if you experience any perceivable reduction in water pressure
    - Please report if you can hear the device operating (whining noise)
    - Please report about any change in comfort when using the showerhead (e.g. do you notice the additional weight, is the LCD at your normal field of view)
- During your shower, please confirm the following info is available at the LCD
    - Current water use in liters; the number should increase as you take your shower
    - Current water temperature in C (or F); the temperature should change depending on your preferred mix of hot/cold water
    - Current efficiency class in an A to F scale; the more time/hot water, the lower the rating (will change during the shower)
    - Visualization of efficiency using a polar bear graphic; the more time/hot water you use, the less ice cubicles appear (will change during the shower)
- After your shower, please confirm the following info is available at the LCD
    - The LCD should be ON for at least 30 sec
    - Total water use in liters
    - Average water temperature in in C (or F)
    - Shower efficiency class in an A to F scale
    - Visualization of efficiency using a polar bear graphic

DAIAD@home (mobile)

- Open the mobile app store of your device (Android and iOS are supported)
- Search for the application called 'DAIAD'
- Install the application
- Open the application
- The welcome screen provides four messages rotating every few seconds
    - Wait for a full rotation
    - Swipe left-right to manually rotate the messages
- Select 'Cambiar a Espanol' or 'Switch to English' (depends on your location) and return to your preferred language
- Press Sign Up
- Scroll up/down the screen with location options
- At the bottom of the screen select 'How to Join?'
- Scroll up/down the screen with join options
    - For each one, select it, follow the instruction and then press Back
- Go back to the welcome screen and select 'Already a member? Sign in!'
- Enter a random email and password combination; the application will not allow you to continue
- Go back to the welcome screen
- Exit the app (force close or Home button)

Initialization and Pairing

- Unpackage the b1 and follow the step-by-step instructions included in the box to install it in the showerhead
    - If you are testing in the DAIAD aquarium ignore this step
    - If you have been given multiple devices, install each one at your showers
- Open the DAIAD mobile app
- At the welcome screen press 'Sign Up' and then select 'Other locations'
- Enter your email and select a password (enter twice)
    - Provide a wrong email; the application will present an error message
    - Enter a different password in the second field; the application will present an error message
- Follow the step-by-step wizard to pair your mobile device with the b1
    - Please report for any delays over 10 seconds during pairing
    - Please report if the instructions are not clear or if you have had any trouble following them
    - Please report if the b1 LCD is deactivated after the pairing process
- After the wizard has ended provide any friendly name you want for the shower (e.g. Master Bathroom)

- If you have been given additional b1 devices, please select Yes in the following screen and follow the pairing process for each one of them
- After the pairing process has completed the application will present a message informing you that DAIAD is in learning mode
    - Exit the app (Home button) and open it again; the same message will appear
    - Force close the app and open it again; the same message will appear
    - Uninstall the app, reinstall it, and open it again; the same message will appear
- Follow the sequence you have been given and that appropriate sub-sections:
    - Turning on b1, turning on mobile
    - Turning on mobile, turning on b1

Turning on b1

- Contact Sofia or Nikos, ask them to set 'b1 ON', and wait for their confirmation
- Open the DAIAD app; you will be welcomed by a new message about the b1 display being enabled
- Open the water in the shower having the mobile device near; after 5-15 seconds the b1 LCD will start working
    - Please report for any delays above 15 sec
- Exit the app (Home button) and open it again; the same message will appear
- Force close the app and open it again; the same message will appear
- Uninstall the app, reinstall it, and open it again; the same message will appear

Turning on mobile

- Contact Sofia or Nikos, ask them to set 'mobile ON', and wait for their confirmation
- Open the DAIAD app; you will be welcomed by a new message about the mobile application being enabled
- Press OK and you should have full access to the DAIAD app
    - Exit the app (Home button) and open it again; you should have full access to the DAIAD app
    - Force close the app and open it again; you should have full access to the DAIAD app
    - Uninstall the app, reinstall it, open it, and sign in again; you should have full access to the DAIAD app
    - Sign out and sign in; you should have full access to the DAIAD app

Using the DAIAD mobile app (new user)

- Open the mobile app
- At the Dashboard page
    - Press the menu button (top right 'hamburger')
    - Select each option, going back to the Menu again

- Press each of the 3 complications and turn back to the Dashboard
- Press 10 times the central gauge; the debug pane will appear; press Exit to return to the Dashboard
- Scroll left right to view the 3 Tips under central gauge
- Turn on the water; the central gauge will change after 5-15 seconds and present information for the current shower (water used, water temperature, energy efficiency)
- Wait for 30 seconds and turn of the water; after 5-15 seconds the central gauge will return to its original content
- Press the devices icon at the top right corner; the paired b1 will appear; exit to the Dashboard
- Press each of the icons at the bottom part of the screen waiting 5 seconds at each screen

- Stats
  - Select the Stats icon
  - Select in sequence the available options at the top waiting 5 seconds at each graph change
  - Select in sequence the available options under the chart waiting 5 seconds at each graph change
  - Scroll up/down the Stats screen
  - View the list of events and press on each one; you should have at least two: one from the pairing process and one for the b1 ON process
    - Select the event; the application will provide you info on the specific event
    - Select in sequence the available options at the top waiting 5 seconds at each graph change
    - View the consumption statistics and press all available options waiting 5 seconds at each screen change
    - Press back
- Messages
  - Select the Messages icon
  - Select in sequence the available options at the top waiting 5 seconds at each screen change; you should have access only to 3 Tips
  - Select one Tip, scroll up/down and press the X button; you will return to the Tips pane with the particular Tips marked as read (the small blue bubble on its left will disappear)
  - Select the same Tip and in the following screen press the right icon; the next Tip will appear; press the X button
  - Select any of the read Tips, press the left icon 5 times; press the X button
- Settings
  - Select the Settings icon
  - Scroll up/down
  - Take notice of the status of the progress bar and select the user

    - Add a photo from the user (take one or add from library); press back, the photo will be available and the progress bar will be increased
    - Add a family member (add any name/sex/age you like)
    - Add another family member (add any name/sex/age you like)
        - Go to the Stats page, select any event, and change the user to any of the new family members
        - Go to the Dashboard, then to Stats, select the above event; confirm the new user has been added
- Water Calculator
    - Go to the top left Menu ('hamburger' icon) and select Water Calculator
    - Follow the steps and provide answers for your own household
    - At the results page select all available options at the top
- Shower timer
    - Go to the top left Menu ('hamburger' icon) and select Shower Timer
    - Select a Custom Shower
    - Select all available options at the top
    - Select Volume, set alarm to 6 liters and turn on the water
        - The display will automatically update with the current consumption; audible queues and changing gauge colors will appear as you reach 80% and 100% of your volume limit
        - Turn off the water; the timer will automatically stop and present information about the event
        - Go Back
    - Select Time, set alarm to 2 mins (120 secs) and turn on the water
        - The display will automatically update with the current consumption; audible queues and changing gauge colors will appear as you reach 80% and 100% of your time limit
        - Turn off the water; the timer will automatically stop and present information about the event
        - Go Back
    - Select Energy, set alarm to 150 Watts and turn on the water
        - The display will automatically update with the current consumption; audible queues and changing gauge colors will appear as you reach 80% and 100% of your energy limit
        - Turn off the water; the timer will automatically stop and present information about the event
        - Go Back
    - Go to the Stats page; confirm that these 3 events are available
- Log out and log in; you should have access to all data recorded previously

- Uninstall, reinstall and log in; you should have access to all data recorded previously

Using the DAIAD mobile app (existing user)

- Open the DAIAD mobile app
    - Make sure you have Signed out from any pre-existing users
- Select 'Already a member? Sign in!' and enter the user credentials (email, password) you have been given
    - The user will have one of the following pre-loaded data: b1 only, smart water meter only, smart water meter and b1
    - At the next steps, ignore tests that do not apply in SWMs or b1 according to the user you have been given
- At the Dashboard page
    - Press the menu button (top right 'hamburger')
    - Select each option, going back to the Menu again
    - Press each of the 3 complications and turn back to the Dashboard
    - Press 10 times the central gauge; the debug pane will appear; press Exit to return to the Dashboard
    - Press the top right complication (plus icon)
        - If your user has access to a SWM and a b1 do the following steps for each data source (press the top slider at the right and left accordingly)
        - Select each of the available central gauge options, press Done and return to the Dashboard; confirm the central gauge has changed and press the plus icon again
        - Select each of the available options for the 3 complications, press Done and return to the Dashboard; confirm the complication has changed; select it to view the available shortcut and return to the Dashboard; the plus icon again
    - Scroll left right to view the available messages under the central gauge
    - Press the devices icon at the top right corner; the paired b1 will appear; exit to the Dashboard
    - Press each of the icons at the bottom part of the screen waiting 5 seconds at each screen
- Stats
    - Select the Stats icon
    - Select in sequence the available options at the top waiting 5 seconds at each graph change
        - If your user has access to a SWM and a b1 do the following steps for each data source (select the appropriate data source)
        - View SWM data at the day, week, month, year and All level; press the arrows left/right to view past time periods
        - View b 1 data for the past 20, 50, and All events; press the arrows left/right to view past event collections

- Select in sequence the available options under the chart waiting 5 seconds at each graph change
- Scroll up/down the Stats screen
- View the list of events and press on each one; you should have at least 5 per day; these will be real-time events or historical events ('??' appear instead of time/date for these type of events)
    - Select an event; the application will provide you info on the specific event; select at least one historical and one real-time event
    - For real-time events:
        - Select in sequence the available options at the top waiting 5 seconds at each graph change
        - View the consumption statistics and press all available options waiting 5 seconds at each screen change
        - Change the appointed event for the specific user (pick anyone you like)
        - Press back
    - For historical events:
        - Enter a date; return to the Stats page and confirm the date has been set for the event
        - Change the appointed event for the specific user (pick anyone you like)
        - Press back

- Messages
    - Select the Messages icon
    - Select in sequence the available options at the top waiting 5 seconds at each screen change; you should have access only Tips, Alerts and Recommendations
    - Select one Tip, scroll up/down and press the X button; you will return to the Tips pane with the particular Tips marked as read (the small blue bubble on its left will disappear)
    - Select one Alert, scroll up/down and press the X button; you will return to the Alerts pane with the particular Alert marked as read (the small blue bubble on its left will disappear)
    - Select one Insight, scroll up/down and press the X button; you will return to the Insights pane with the particular Insight marked as read (the small blue bubble on its left will disappear)
    - Select any of the Alerts, Tips and Recommendations, go through all of them using the left and right arrows
- Settings
    - Select the Settings icon
    - Scroll up/down
    - Take notice of the status of the progress bar and select the user
    - Add a photo from the user (take one or add from library); press back, the photo will be available and the progress bar will be increased

    - Add a family member (add any name/sex/age you like)
    - Add another family member (add any name/sex/age you like)
        - Go to the Stats page, select any event, and change the user to any of the new family members
        - Go to the Dashboard, then to Stats, select the above event; confirm the new user has been added

DAIAD@home (web)

- Visit the URL you have been provided with; you can use one of the following browsers: Firefox, Chrome, Edge, Safari
    - Keep track of the browser used (type, version) and any active ad-blockers
    - Make sure you have signed-out from any other DAIAD web application
    - In case of any problems, try first to force reload the page (Ctrl+F5)
- Sign in using the user credentials (email, password) you have been given
    - The user will have one of the following pre-loaded data: b1 only, smart water meter only, smart water meter and b1
    - At the next steps, ignore tests that do not apply in SWMs or b1 according to the user you have been given
    - Do not allow the browser to store the credentials
- Dashboard
    - Select each of the options at the left pane; confirm the application returns to the appropriate sections and return to the Dashboard
    - Log out and log in again using the same credentials
    - For each of the available widgets select all of its options (top right), hover your pointer over the any visualization waiting for the appropriate tooltips to appear, and select More to view its information in detail
    - Add a new widget following the provided wizard; explore the widget and delete; repeat for at least 3 different widgets of your choice
- Stats
    - Select the icon from the top left menu
        - You should the DAIAD mobile app open and signed-in for the specific user
        - Follow the steps below and compare with the corresponding data presented at the DAIAD mobile app; report for any differences or inconsistencies
    - Scroll up/down the Stats screen
    - If your user has access to a SWM and a b1 do the following steps for each data source (select the appropriate data source from the top right corner)
        - For SWM

        - View SWM data at the day, week, month, year and All level; press the arrows left/right to view past time periods
        - Hover over the charts at any location you want and wait for the tooltip to appear
        - Select 'Compare with' and scroll through the graph
    - For b1
        - View b1 data for the past 20, 50, and All events; press the arrows left/right to view past event collections
        - Select in sequence the available options at the left of the chart (Volume, Energy, Temperature, Duration, Efficiency)
  - View the list of events and press on each one; you should have events for at least 2 months (SWM) and/or 100 showers (b1). Repeat the following for each type of data source:
    - Scroll up/down the list of events
    - Change the ordering using all available options
    - Download the events; open the file and confirm all events are available (in different tabs for multiple b1 and/or SWM)
    - Select at least 5 events; the application will provide you info on the specific event in a pop-up page; confirm the same information is provided from the DAIAD mobile app
- Messages
  - Select the Messages from the top left menu
  - Select in sequence the available tabs at the top; you should have access to Tips, Alerts and Recommendations
  - Select one Tip, and it will appear at the left pane; the particular Tip will be marked as read (the small blue bubble on its left will disappear)
  - Select one Alert, and it will appear at the left pane; the particular Alert will be marked as read (the small blue bubble on its left will disappear)
  - Select one Insight, and it will appear at the left pane; the particular Insight will be marked as read (the small blue bubble on its left will disappear)
  - Select any of the Alerts, Tips and Recommendations, go through all of them

DAIAD@commons

- Visit the URL you have been provided with; you can use one of the following browsers: Firefox, Chrome, Edge, Safari
  - Keep track of the browser used (type, version) and any active ad-blockers
  - Make sure you have signed-out from any other DAIAD web application
  - In case of any problems, try first to force reload the page (Ctrl+F5)
- Sign in using the user credentials (email, password) you have been given

- The user will have access to data and analytics only for b1 data (bottom-up deployment) for a number of demo users
- Do not allow the browser to store the credentials

- Dashboard
  - Select each of the widgets at the top and confirm you are being directed in the correct page
  - Explore the time axis (x) in the chart
  - Pan and zoom in/out at the map component
- Analytics
  - Charts
    - Select all combinations of the following criteria:
      - Detail level: hour, week, month, year
      - Metric: Average, Total, Top
      - Time range: this month, last month, last quarter, last year, custom (pick any you like in the last 2 months)
    - The chart will be updated according the provide criteria; confirm it has been generated correctly
  - Maps
    - Select all combinations of the following criteria:
      - Any time interval in the last 2 months (at least 3 different time intervals)
    - The map will automatically be updated with a choropleth of heat-map visualization (depends on the number of users)
      - Hover over multiple areas and confirm its total water use is updated
      - Click on at least 3 consumers
      - Pan and zoom in/out at the map component
      - Press the play icon for the visualization to start; press pause/play at least 3 times (pick randomly)
  - Reports
    - Use a reference time period in the last 2 months (at least 3 different time periods) and Refresh
    - Confirm all charts have been generated correctly

DAIAD@utility

- Visit the URL you have been provided with; you can use one of the following browsers: Firefox, Chrome, Edge, Safari
  - Keep track of the browser used (type, version) and any active ad-blockers
  - Make sure you have signed-out from any other DAIAD web application
  - In case of any problems, try first to force reload the page (Ctrl+F5)

- Sign in using the user credentials (email, password) you have been given
    - The user will have access to data and analytics only for SWM and b1 data (top down deployment) for a number of demo users
    - Do not allow the browser to store the credentials
    - You will be having administrator privileges; do not use any of the functionalities explicitly excluded below

- Dashboard
    - Select each of the widgets at the top and confirm you are being directed in the correct page
    - Explore the time axis (x) in the chart
    - Pan and zoom in/out at the map component
- Analytics
    - Charts
        - Select all combinations of the following criteria:
            - Detail level: hour, week, month, year
            - Metric: Average, Total, Top
            - Data source: SWM and b1
            - Time range: this month, last month, last quarter, last year, custom (pick any you like in the last 2 months)
        - The chart will be updated according the provide criteria; confirm it has been generated correctly
    - Maps
        - Select all combinations of the following criteria:
            - Any time interval in the last 2 months (at least 3 different time intervals)
            - At least 3 users groups (pick randomly)
            - At least 2 data sources
        - The map will automatically be updated with a choropleth of heat-map visualization (depends on the number of users)
            - Hover over multiple areas and confirm its total water use is updated
            - Click on at least 3 consumers
            - Pan and zoom in/out at the map component
            - Press the play icon for the visualization to start; press pause/play at least 3 times (pick randomly)
    - Reports
        - Select SWM and b1, use a reference time period in the last 2 months (at least 3 different time periods) and Refresh
        - Confirm all charts have been generated correctly

    - Favorites
        - Confirm the Favorites you have selected is included in this page
- Forecasting
    - Change the time-scale of the chart and deactivate any combinations of the provided time-series
    - Search for at least 3 users (pick randomly); their actual and forecasted water use will be added in the chart
    - Select the various chart types
- Consumers
    - Users
        - Search for a specific user by name (pick at least 3; full or partial string)
        - Search for a specific user by account (pick at least 3; full or partial string)
        - Search for a specific meter ID (pick at least 3; full or partial string)
        - Explore the different views of the table; confirm the map component presents the same information (at least two views)
        - Add at least 3 users as Favorites (pick randomly); sign-out/sign-in and confirm the Favorites have been maintained
        - Click on the charts Icon for at least 4 users; their water use will appear at the bottom chart
        - Pan and zoom in/out at the map component
        - Create a random geographical area and perform a search based on location; confirm the table on the left contains the same information; delete the geographical area
        - Create a new user group selecting at least 3 users (pick randomly); take note of the name/members as you will use them in the following
    - Groups
        - Filter based on group type
        - Filter based on group name
        - Explore the different views of the table
        - Select at least 3 segments/groups (pick randomly) and view info on their members
        - Select the charts Icon for at least 4 segments/groups; their water consumption for the last 30 will appear on the chart at the bottom of the page; select Total, Average, Min, Max
- Job Management
    - Select at least 3 of the available jobs and press the Play icon; confirm it has appeared on the execution log (bottom of the page)
        - Important: do not execute an ETL job; it will change the testing data you have been provided with
    - Explore the different views of the executions table

    - Filter by job name
    - Filter by exit code
- Engagement
    - Messages
        - Explore the different views of the table
        - Mark a message (pick randomly) as inactive
        - Delete a message (pick randomly)
        - Create a new message (press the plus icon)
        - Save the message; confirm it is added in the table
        - Edit the message and save; confirm it is added in the table
    - Announcements
        - Select a user group and create an announcement; confirm the announcement appears the History section (bottom page)
        - Randomly select at least 5 users and create an announcement; confirm the announcement appears the History section (bottom page)
        - Contact Nikos to confirm the announcement has been pushed in the queue of the individual users
- Trial Management
    - Overview
        - Explore the different views of the table
        - Search for a specific user (pick at least 3)
        - Add a new user and confirm it has been created
        - View the activity for a specific user (pick at least 3);
            - select the SWM and b1 icons, the chart at the bottom of the page will be updated accordingly
            - download the corresponding data for each of the users; open the provided xls file and compare against the information in charts
    - Pilot Reports
        - Select a month (at least for M1, M3) and press Refresh to generate the charts
        - Confirm the charts are generated with the appropriate cluster/segment groups per category
        - Save at least 3 charts (randomly) as images and add them in an MS Word document
- Support
    - Logs
        - Explore the different views of the table
        - Search for a specific account (pick at least 3)
        - Search for a specific filter (use all filters: Debug, Info, Warn, Error)

    - Mode Management
        - Do not test; it is only active for the DAIAD internal group; tested by Sofia and Yannis exclusively
    - Data Management
        - Do not test; it affects bulk data import; tested by Yannis and Mihalis exclusively
- User Preferences
    - Do not test; changes the credentials/info of the admin

# 10. Annex: Mobile devices

Devices owned by the project

- 2 x IPhone 5S (one on latest iOS version, one in tester-selected iOS version)
- 2 x iPad mini 3rd generation (one on latest iOS version, one in tester-selected iOS version)
- 2 x LG Nexus 5 (one in latest Android version, one tester-select Android version)
- 1 x Asus Nexus 7 (2nd generation, latest Android version)

Devices owned by the Consortium

- iPhone 4s (latest iOS version)
- iPhone 5c (latest iOS version)
- iPhone 5s (latest iOS version)
- iPhone 6 (latest iOS version)
- iPad (multiple, 3rd and 4th generation, latest iOS version)
- iPad Air (1st and 2nd generation, latest iOS version)
- iPad Mini (3rd and 4th generation, latest iOS version)
- Samsung Galaxy S4 (latest Android version)
- Samsung Galaxy S4 mini (latest Android version)
- Samsung Galaxy S5 (latest Android version)
- Motorola Moto G (1st, 2nd, 3rd generations, latest Android version)
- Samsung Galaxy Tab S2 (latest Android version)
- Samsung Galaxy Tab E (latest Android version)
- Lenovo A7 (latest Android version)
- Lenovo Yoga 3 (latest Android version)
- LG Nexus 5X (latest Android version)

Devices owned by outside parties

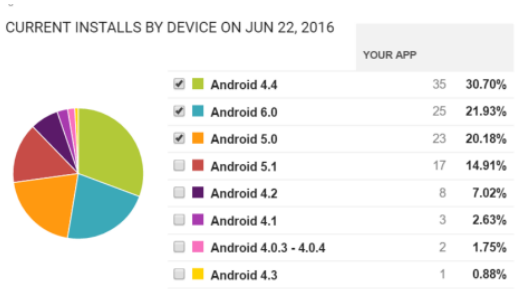
- iPhone 4s (latest iOS version)
- iPhone 5 (latest iOS version)
- iPhone 5c (latest iOS version)
- iPhone 5s (latest iOS version)
- iPhone 6 (latest iOS version)
- iPhone 6s (latest iOS version)

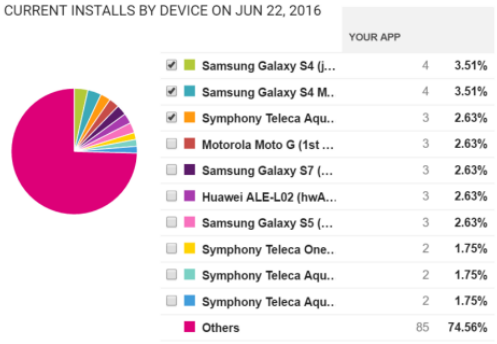
- iPhone 6 Plus (latest iOS version)
- iPad (multiple, 3rd and 4th generation, latest iOS version)
- iPad Air (1st and 2nd generation, latest iOS version)
- iPad Mini (3rd and 4th generation, latest iOS version)
- Samsung Galaxy S4 (latest Android version)
- Samsung Galaxy S4 mini (latest Android version)
- Samsung Galaxy S5 (latest Android version)
- Samsung Galaxy S7 (latest Android version)
- Samsung Galaxy S7 Edge (latest Android version)
- Motorola Moto G (1st, 2nd, 3rd generations, latest Android version)
- Samsung Galaxy Tab S2 (latest Android version)
- Samsung Galaxy Tab A (latest Android version)
- Samsung Galaxy Tab E (latest Android version)
- Samsung Galaxy Tab 4 (latest Android version)
- Lenovo A7 (latest Android version)
- Lenovo A10 (latest Android version)
- Lenovo Yoga 3 (latest Android version)
- Lenovo Tab2 A10 (latest Android version)
- HTC Nexus 9 (latest Android version)
- Motorola Nexus 5 (latest Android version)
- Motorola Nexus 6 (latest Android version)
- LG Nexus 5X (latest Android version)
- Sony Xperia Z5 (latest Android version)
- Sony Xperia M5 (latest Android version)
- Sony Xperia Z3 compact (latest Android version)

Devices owned by Trial participants

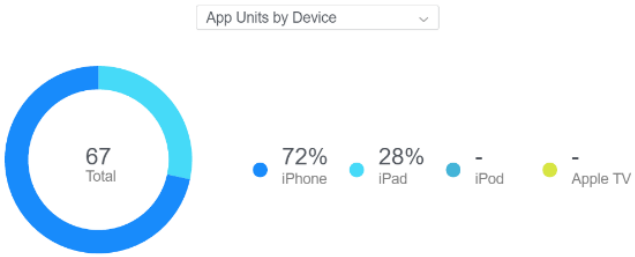
- Symphony Teleca Aquaris E5 HD (latest Android version)
- Symphony Teleca Aquaris OnePlusOne (latest Android version)
- Symphony Teleca Aquaris E10 (latest Android version)
- Symphony Teleca Aquaris E4.5 HD (latest Android version)
- Huawei ALE-L02 (latest Android version)

## Installations breakdown (Android)




## Installations breakdown (iOS)

# 11. Annex: Testing Data

## 11.1. Smart Water Meter Data

The first major category of testing data concerns time-series of water measurement readings generated by AMAEM's deployed smart water meters.

### 11.1.1. Data Format

SWM time-series are provided in plain-text files (CSV) with a well specified format in which every field is separated by a semi-colon (;) character. The default format is presented in Figure 19.

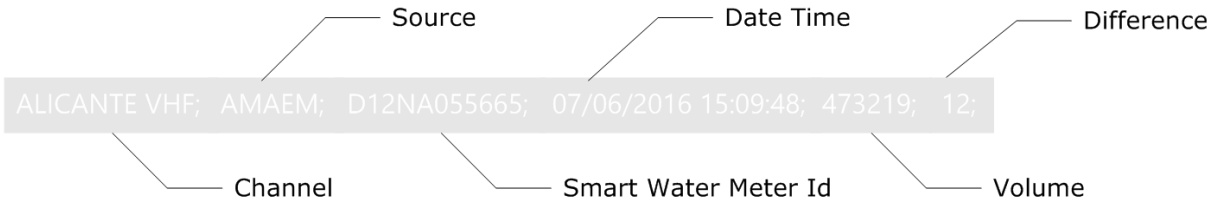



*Figure 19: Smart Water Meter data example*

Every record consists of six fields:

- **Channel**. A string representing the transmission channel used to receive the SWM readings. This is an optional field, as for all SWMs deployed in Alicante it has the same value ('Alicante VHF').
- **Source**. A string representing the source (i.e. utility) the SWM belongs to. This is an optional field, as for all SWMs deployed in Alicante it has the same value ('AMAEM').
- **SWM ID**. An alphanumeric value uniquely identifying each SWM.
- **Reading date**. A timestamp (date, time) of when the reading was taken from the SWM, expressed in local time zone (Europe/Madrid for AMAEM).
- **Volume reading**. The actual value measured by the SWM in liters.
- **Difference**. The difference in liters between two most recent SWM readings. This is an optional field.

### 11.1.2. Historical SWM Data (1K SWMs)

This dataset contains hourly SWM time-series for 1,000 households in Alicante (selected randomly) covering a time period of a whole year, i.e., from 1st of July 2013 to 30th of June 2014 (8.6M records).

This dataset proved extremely valuable for our entire body of work (beyond simply integration and testing) as it shed light to the real-world characteristics and issues of SWM data management and analysis. In particular, we identified the following data quality issues:

- Missing SWM readings for a particular SWM ID. This can be caused by a malfunctioning SWM, SWM RF component (add-on to the mechanical SWM), or RF connectivity (temporary connection issues between the SWM and the RF antenna/aggregator). Missing SWM readings can vary, from 1 missing reading/day to several days.
- Out of order SWM readings. This can be caused by a malfunctioning SWM (internal clock) or the underlying software for SWM data management (external to the project).
- Changing time period between consecutive measurements. This can be caused by a malfunctioning SWM or the underlying software for SWM data management (external to the project). The period between two consecutive measurements for the same SWM is not always the anticipated 1h, but may drift for a number of minutes (e.g., 80 min instead of the expected 60 min).

It is worth highlighting that such data quality issues are considered as an *accepted behavior* of a SWM roll-out (i.e. within normal operation parameters), as the emphasis is given on accurate billing (i.e. difference between two measurements in time for the billing period). Consequently, low data quality was acknowledged early in the project's lifecycle as an *integral characteristic* of production SWM roll-outs that must be gracefully handled by the entire DAIAD system.

A sample of several records for a single SWM follows (notice the optional fields Channel, Source, Difference are missing, as well as the slight change in time period):

> *C12FA151955;08/06/2014 03:07:17;112370;*
>
> *C12FA151955;08/06/2014 02:07:17;112369;*
>
> *C12FA151955;08/06/2014 01:07:17;112369;*
>
> *C12FA151955;07/06/2014 23:46:26;112368;*
>
> *C12FA151955;07/06/2014 22:46:26;112366;*

## 11.2. Amphiro data

Amphiro data is separated into two categories (real-time and historical data), depending on how the extraction event was transmitted from the device to an external data aggregator. In particular:

- Historical data. It comprises shower events stored in the device's internal non-volatile memory and concerns events during which a mobile device was not available to receive the time-series in real-time (b1) or produced with the previous version of the device (a1, no RF capabilities). For each event: (a) the timestamp is not known (only the unique shower ID) since the device does not include an internal clock (due to its energy-autarkic nature), and (b) only aggregate data are available (due to the internal memory constraints, ~200 showers can be stored).
- Real-time data. It comprises shower events captured from a mobile device within range during an actual shower taking place. In this case (a) we receive actual time-series for the shower, but at non-stable periods (transmission frequency depends on energy management), and (b) the timestamp of the time-series is inferred by the mobile device.

### 11.2.1. Amphiro historical (historical extractions)

This data set includes (a) shower extraction events from a1 devices and (b) detailed demographic information, produced in the context of an external study performed by Amphiro on 77 Swiss households (5,795 showers).

The dataset is provided in a plain-text file (CSV) with a well specified format in which every field is separated by a semi-colon (;) character. Its fields are:

- Household ID
- Trial group ID (treatment groups for particular study; ignored)
- Total number of showers per household
- An incremental shower ID (unique per device)
- Duration of a shower in seconds
- Volume of water used during a shower in liters
- Average flow rate of a shower in liters per minute
- Average temperature of a shower in $^{0}$C
- Duration of interruptions during a shower (e.g. while soaping) in seconds
- Number of male residents living in a household
- Number of female residents living in a household
- Number of residents living in a household and aged between 0 and 5 years; 6 and 15 years; 16 and 25 years; 26 and 35 years; 36 and 45 years; 46 and 55 years; 56 and 65 years; 66 and 75 years; over 75 years
- Costs for water consumption included in the rent (YES/NO)
- Costs for heating energy included in the rent (YES/NO)
- Number of household residents using the monitored shower
- Number or household residents having long hair
- Gender, age, nationality, and level of education of the survey taker (one per household)
- Number of adults living in the household
- Number of children living in the household
- Form of housing
- Monthly net income of the household in CHF (Fr)

An example of several records is the following (consecutive commas indicate no value):

> *2640,2,47,47,0,1591,0,304,37.5,11.4645,2,1,1,,,,,,,,2,,0,0,2,0,male,65+,Switzerland,apprenticeship,2,no child, rental apartment,81-115 m2,7000 - 7999 Fr.*
>
> *2640,2,47,34,0,216,0,33,37.5,9.16667,2,1,1,,,,,,,,2,,0,0,2,0,male,65+,Switzerland,apprenticeship,2,no child,rental apartment,81-115 m2,7000 - 7999 Fr.*

*2640,2,47,3,0,31,0,5,37.5,9.67742,2,1,1,,,,,,,,2,,0,0,2,0,male,65+,Switzerland,apprenticeship,2,no child,rental apartment,81-115 m2,7000 - 7999 Fr.*

*2640,2,47,4,0,1406,0,244,37.5,10.4125,2,1,1,,,,,,,,2,,0,0,2,0,male,65+,Switzerland,apprenticeship,2,no child,rental apartment,81-115 m2,7000 - 7999 Fr.*

### 11.2.2. Amphiro data (real-time)

This data set includes real-time shower extractions produced from b1 devices from the Consortium. At the end of the reporting period (M23) it included ~600 shower events, and it is constantly increasing in size.

For each shower extraction two types of data are persisted, namely, shower aggregated data referenced as **session** data and a time series of detailed data referenced as **measurements**.

For each session the following attributes are stored:

- Timestamp: The time of extraction expressed as Unix Time; (Unix Time = the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970, not counting leap seconds)
- History: True if the session is a historical session; Otherwise False
- Updated: True if this real-time session represented a historical one that has been converted to real-time
- Volume: Total volume in liters
- Energy: Total energy in watts
- Duration: Session duration in seconds
- Temperature: Average temperature in Celsius (°C)
- Flow: Average flow in litters per minute

The attributes of a single measurement are:

- Timestamp: The time that the measurement has been recorded
- Volume: The volume difference since the last measurement in liters
- Energy: The energy difference since the last measurement in watts
- Temperature: The current temperature in Celsius (°C)

## 11.3. Synthetic data

### 11.3.1. Synthetic data for scalability testing

Using a Big Data generator (part of the BigDataBench benchmark suite[65]) we generated a synthetic dataset based on the SWM dataset's format after the proper feature extraction. The synthetic dataset consists of 100M records with the following format:

- An alphanumeric identifier
- 9 integer features with 10 decimals, from 1 to 99
- 2 features as target variables
    - One floating point with 10 decimals from 1 to 99
    - One binary

An example of several records is the following:

```
Alanah,13,1,6,10,1,1,5,405,5,75.3193100497,1
Peg,22,3,3,10,3,1,5,380,5,51.2148987257,1
Patch,5,3,6,7,2,1,6,480,7,32.8837968085,1
Quality,7,2,1,7,2,1,9,467,9,31.3787360197,1
Cayla,13,1,4,4,1,1,8,476,7,96.5476215768,1
Oralie,5,1,6,8,1,1,5,822,2,19.4084972708,1
```

Using the same generator, we created a second synthetic dataset based on the shower related features of the amphiro historical dataset (11.2.1). The synthetic dataset consists of 15M records with the following format:

- An incremental identifier representing the household
- An incremental identifier representing the shower
- 5 floating point features of various ranges
- 1 binary feature as a target variable

An example of a several records is the following:

```
2234,33,12.21,62.54,3.57,75.23,0.4,1
2234,34,11.23,32.12,1.66,85.92,0.1,0
2234,35,15.19,34.65,1.32,83.22,0.15,1
2236,1,1.23,3.23,2.12,26.71,0.87,0
```

[65] http://prof.ict.ac.cn

*2236,2,6.18,8.86,5.87,58.21,0.94,0*

### 11.3.2. Synthetic data for integration and usability testing

We have developed a set of Python scripts to generate arbitrary synthetic data for integration and usability testing.

#### 11.3.2.1. SWM

We have developed two scripts; the first generates random values, while the second outputs SWM data using the historical SWM data (see ) as a seed. Both scripts' CSV outputs comply with the format accepted by our backend SWM data import helper (see 11.1.1, 11.3.3).

Random data

This script generates hourly SWM time-series for a single household and arbitrary time period. The parameters accepted are the following:

- device (required): specific SWM ID to assign the generated data to
- startDate (optional: default beginning of year): the timestamp of the beginning of the period for which data is generated
- endDate (optional: default time of execution): the timestamp of the end of the period for which data is generated
- output (optional: default out.csv): the output CSV file

Example call:

./generateMeterData --device C11FA586148 --startDate 1451606400000 --endDate 1467348719781

Seeded data

The script outputs SWM time-series for an arbitrary number of SWM IDs using input SWM data as a seed. The parameters accepted are:

- input (required): the input csv file containing SWM data in the expected format
- deviceNum (optional): number of random SWMs for which to extract data
- devices (optional): specific real SWM serials from the input file to be extracted
- output (optional: default out/): the output folder where the produced CSV files will be saved

Example calls:

./extractSWMData --input SWM1year.csv --deviceNum 4

./extractSWMData --input SWM1year.csv --devices C11FA586148 C11FA586168

#### 11.3.2.2. Amphiro

The script generates a user-selected collection of real-time and/or historical events produce randomly.

- *Real-time*. First, we produce a random integer for the shower duration (4-30min) and then for regular intervals we generate random amounts for water volume, temperature and energy (3 time-series per shower). Finally, we compute the aggregated values for each metric (sum or average depending on the metric).
- *Historical.* In this case we generate an arbitrary number of showers, with each shower metric generated randomly.

The script outputs a JSON file with a random number of showers for the time range provided. The parameters accepted are the following:

- device (required): the amphiro device key to assign the generated data to
- username (required): the username with which the device key is associated
- password (required): the user password
- lastId (optional - default: 0): the last shower id for the device key provided (shower ids must be incremental)
- startDate (optional: default beginning of year): the timestamp of the beginning of the period for which data is generated
- endDate (optional: default time of execution): the timestamp of the end of the period for which data is generated
- output (optional: default out.json): the output JSON file

Example call:

./generateAmphiroData --device 2bd99b5d-3395-4455-b0ae-b716a56d7757 --username user1@daiad.eu --password ******** --lastId 125 --startDate 1451606400000 --endDate 1467348719781

### 11.3.3. Data import

We have developed a data import facility integrated into the DAIAD system which is used to (a) streamline testing with varying datasets, and (b) initialize the production system during the Trials. Through a UI (Figure 20), the administrator simply loads a data file to import multiple SWM readings (existing or new) to an arbitrary number of user (existing or new).

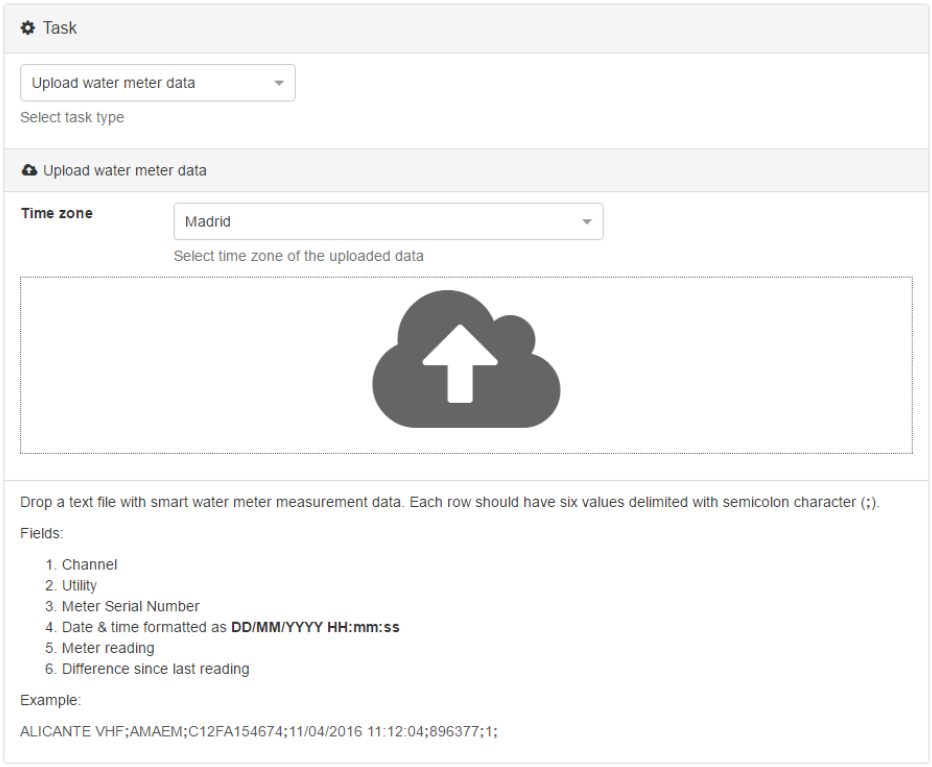



*Figure 20: Data uploading form*

# 12. Annex: Arduino prototype