Project no.:	**610658**
Project full title:	**eWALL for Active Long Living**
Project Acronym:	**eWALL**
Deliverable no.:	**D3.3.1**
Title of the deliverable:	**eWALL configurable metadata streams**

Contractual Date of Delivery to the CEC:	**31.10.2014**
Actual Date of Delivery to the CEC:	**31.10.2014**
Lead contractor for deliverable:	**TUS (P11)**
Author(s):	**Aristodemos Pnevmatikakis (AIT), Krasimir Tonchev (TUS), Nikolay Neshov (TUS), Sofoklis Kyriazakos**
Participants(s):	**TUS (P11), AIT (P07) , AAU (P01), UOZ (P12), AAU (P01)**
Contributing work package:	**WP3**
Nature:	**R**
Version:	**1.0**
Total number of pages:	**27**
Start date of project:	**01.11.2013**
Duration:	**36 months – 31.10.2016**

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 610658

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable is a report on generating the metadata streams from the caring home environment and storage in database.

Keyword list: Metadata, streaming metadata, JSON, databases, SQL, no-SQL

Document History

Version	Date	Author (Unit)	Description
0.1	29/08/14	Aristodemos Pnevmatikakis (AIT)	ToC
0.2	08/10/14	Krasimir Tonchev, Nikolay Neshov (TUS)	INTRODUCTION, DATABASE SELECTION, CONCLUSION & PLANNING
0.21	17/10/14	Aristodemos Pnevmatikakis (AIT)	Updates to chapters 2 & 3
0.3	25/10/14	Krasimir Tonchev, Nikolay Neshov (TUS)	
0.4	25/10/14	Aristodemos Pnevmatikakis (AIT)	Final polishing (section 3.1 and chapter 6) and WP3-leader approval
1.0	27/10/14	Sofoklis Kyriazakos (AAU)	Technical review & approval

Table of Contents

1 Executive Summary

This deliverable describes the generation of the metadata streams and the research on readily available database to be used as a primary mean of storage in the home environment of the eWall system. Three major types of database classes are analysed, namely SQL, non-SQL, and RDF. Based on analysis, a proper selection for the database that will best fit eWall requirements is selected for implementation. The data generated by in-house sensors is stored in JSON format. A detailed description of each sensor JSON format description, integrated in current state of eWall, is presented. Examples of environmental monitoring perceptual component and accelerometer JSON formatting are presented. A conclusive chapter summarizes the current status and the planning for future developments.

2 Introduction

Storage in the context of eWall is implemented in two places – one "in home" and another in the cloud. This document analyses potential databases for "in home" storage, taking into account the requirements and constraints of the project. The task of this storage is to provide a temporary buffer in situations where home – cloud connection failed due to undetermined reason.

The data produced by the sensors, located inside home, is diverse in terms of bits resolutions, frequency and format. Moreover, the Audio-Visual sensor produces huge amount of data which have to be processed and this to be stored in reasonable time limits. This requires storage capable of handling burst of data and support of formats suitable for eWall platform. Even so, there might be excess of data which requires the availability of mechanism for controlling the rate on which it is stored. This rate control should be done in a reasonable fashion according to rules specified by the application. For example, soring person's ECG signals should not drop drastically which will influence the reasoning done automatically or by medical doctors.

Another important aspect is the format in which the data produced by the sensors, or the metadata extracted by processing algorithms, is stored. This format can be various and can be either human readable or not. Most important is that this format should suit the needs of eWall platform in terms of available tools for working with it. One important requirement is that the format must allow extending the data description when is necessary in the future development.

The deliverable is structured as follows:

- Chapter 3 discusses the eWALL configurable metadata streams, resulting to their formal descriptions using JSON messages.
- Chapter 4 offers an in-depth analysis of the options for the storage of the metadata at home. As a result, we propose the use of CouchDB for storing the metadata.
- Chapter 5 concludes the deliverable.

3 eWALL configurable metadata streams

All the eWALL metadata are generated by sensing the users and their environments in the caring homes. To describe this wealth of information, we maintain a metadata dictionary, shown in Section 3.1. As we build perceptual components that interface to the necessary sensors and produce the metadata, we structure formal JSON messages that convey this information to the home database. The messages currently in use are discussed in Section 3.2.

3.1 Metadata dictionary

The metadata dictionary at the time of writing is given in .the following table. The metadata are structured in RDF triples: The subject, predicate and object are given in the first three columns of the table. The implementation status is given in the final column.

Subject	Predicate	Object	Status
person	ID	Integer	JSON message implemented Not integrated with perceptual component
	positionX	Double, cm in room if 3D tracker, pixel in frame if 2D	
	positionY	Double, cm in room if 3D tracker, pixel in frame if 2D	
	positionZ	Double, cm in room	
	positionConf	Double	
	gender	Integer	
	genderConf	Double	
	age	Integer	
	ageConf	Double	
	emotion	Integer	
	emotionConf	Double	
	timestamp	Time string	
activity	ISA	Double	Implemented Metadata is provided to the database
	IMA	Double	
	steps	Integer	
environment	movement	Logical	Implemented Metadata is provided to the database.
	temperature	Double	
	humidity	Double	
	illuminance	Double	
	gas_lpg	Logical	
	gas_ng	Logical	
	gas_co	Logical	
	timestamp	Time string	
sound	micID	Integer	Not implemented Perceptual components are not mature
	level	Double	
	soundType	String (voice, snore, vacuum cleaner, etc.)	
	period	Double	
	speakerID	Integer	
	angleOfArrival	Double	
	timestamp	Time string	

sleep	significantPressure	Integer	Not implemented
	ISA	Double	
	IMA	Double	
	timestamp	Time string	
voiceComm	carrier	Array (phone, skype or similar voice service)	Not implemented
	startTime	Time string	
	endTime	Time string	
	otherParty	String (phone number) or array of string (Skype IDs)	
	incoming	Boolean	
socialStatus	recipient	String	Not implemented
	carrier	String (facebook, twitter, IM service)	
	message	String	
	timestamp	Time string	
entertainment	startTime	Time string	Not implemented
	endTime	Time string	
	ID	String (game or media title or ID)	
	gameScore	Integer	
domotics	socketID	Integer	Not implemented
	powerConsumption	Double	
	tapID	Integer	
	waterFlow	Integer	
	timestamp	Time string	

3.2 Metadata JSON specification

The environmental sensors monitoring a space are usually bundled together into a single board and they send their signals in a single packet. If any of the signals change significantly, then the environmental monitoring perceptual component sends out the following JSON message:

```
{
  "_id": "2014-09-30T12:30:20.004Z",
  "_rev": "1-a456d3f3f0140d0f5c43c5090b6e8cf5",
  "timestamp": "2014-09-30T12:30:20.004Z",
  "temperature": 24.6,
  "humidity": 37.8,
  "illuminance": 770,
  "movement": true,
  "gas": {"LPG" : true, "NG" : false, "CO" : true}
}
```

E.g. see http://141.85.151.162:5984/mote_arduino1/_changes?include_docs=true.

Low level activity of the user is sensed via the wearable accelerometer and is quantified by the equivalent perceptual component into IMA, ISA and number of steps. These are reported synchronously every 10 sec using the following JSON message:

{
"_id":"2014-09-17T10:23:15.861Z",
"_rev":"16-b69e6bd37105318fb8ded9433502b314",
"timestamp":"2014-09-17T10:23:15.861Z",
"activity":{"IMA":0.702577,"ISA":0.467623,"steps":444333}
}

E.g. see http://141.85.151.162:5984/activity/_changes?include_docs=true.

4 Database selection

4.1 Relational Databases – SQL

- **Advantages**
  - Structured data, less storage required
  - Easily accessible, SQL language
  - Very commonly used, easier to learn
  - Provide advanced functionalities for related data, such as cascade deletions and for multi-step transactions
- **Disadvantages**
  - Need to design a schema for the database beforehand, not easy to update and adapt to constantly changing data
  - Overhead when handling bursts of data insertions/retrievals, as the data needs to be converted
  - Difficulty in working with huge amounts of data
- **Relational Databases Description**

The table below briefly describes some of the most popular non-proprietary (GPL-based or similar) and proprietary relational DBs with current (or recent) support: **MySQL, Oracle, Apache Derby, CUBRID, Drizzle, HSQLDB, Ingres, LucidDB, MariaDB, PostgreSQL, SmallSQL, SQLAnywhere, SQLite**

Database	Supported OS	License	Maintainer, Latest version	Features and Properties
MySQL [1]	Windows OSX Linux BSD UNIX AmigaOS Symbian z/OS Android	GPL v2 or Proprietary	Sun Microsystems (Oracle Corp.) 5.6.21 (2014-09-23)	– ACID, Transactions, Unicode, SQL and GUI interface – Unlimited DB size, 256 TB max table size, 64 kB max row size, 4096 max columns per row, 4 GB as max blob/clob size, 64 kB max CHAR size, 64 bit max NUMBER size, 1000 min DATE value, 9999 max DATE value, 64 max column name size – Temporary table – Indices – B-/B+ (some additional exclusions) – Capabilities – Union, Inner joins, Outer joins, Inner selects, Blobs and Clobs – Other objects – Cursor, Trigger, Function, Procedure, External routine – Partitioning – Range, Hash, Composite, List – Access control – Native network encryption, Enterprise directory compatibility, Patch access (partial), Run unprivileged
Oracle [2]	Windows OSX Linux UNIX z/OS	Proprietary	Oracle Corp. 12c Release 1 (Patchset as of July 2014)	– ACID, Referential integrity, Transactions, Unicode, SQL, GUI and API interface – Unlimited DB size, 4 GB max table size, 8 kB max row size, 1000 max columns per row, 128 TB as max blob/clob size, 32 767 B max CHAR size, 126 bit max NUMBER size, -4712 min DATE value, 9999 max DATE value, 30 max column name size

				– Temporary table, Materalized view – Indices – B-/B+, R-/R+ tree, Hash (some exceptions), Expression, Partial, Reverse, Bitmap, Full-text, Spatial – Capabilities – Union, Intersect, Except, Inner joins (via MINUS), Outer joins, Inner selects, Merge joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query – Other objects – Data domain, Cursor, Trigger, Function, Procedure, External routine – Partitioning – Range, Hash, Composite, List – Access control – Native network encryption, Brute-force protection, Enterprise directory compatibility, Password complexity rules, Run unprivileged, Audit, Resource limit, Separation of duties, Security certification
Apache Derby [3]	Windows OSX Linux BSD UNIX z/OS	Apache	Apache 10.11.1.1 (2014-08-27)	– ACID, Referential integrity, Transactions, Unicode, SQL interface – Unlimited DB size, Unlimited max table size, Unlimited max row size, 1012 max columns per row, 2147483647 chars as max blob/clob size, 254 max CHAR size, 64 bit max NUMBER size, 0001-01-01 min DATE value, 9999-12-31 max DATE value, 128 max column name size – Temporary table – Indices – B-/B+ – Capabilities – Union, Intersect, Except, Inner joins, Outer joins, Blobs and Clobs, Common Table Expressions, Windowing Functions – Other objects – Cursor, Trigger, Function, Procedure, External routine
CUBRID [4]	Windows Linux	GPL v2	NHN Corporation 9.3 (2014-05-23)	– ACID, Referential integrity, Transactions, Unicode, SQL and GUI interface – 2 EB DB size, 2EB max table size, Unlimited max row size, 6400 max columns per row, Unlimited as max blob/clob size, 1 GB max CHAR size, 64 bit max NUMBER size, 0001-01-01 min DATE value, 9999-12-31 max DATE value, 254 max column name size – Indices – B-/B+, Expression, Partial, Reverse – Capabilities – Union, Intersect, Except, Inner joins, Outer joins, Inner selects, Merge joins, Blobs and Clobs, Windowing Functions – Other objects – Data domain, Cursor, Trigger, Function, Procedure, External routine – Partitioning – Range, Hash, List
Drizzle [5]	OSX Linux BSD UNIX	GPL v2, v3 Some BSD Components	Brian Aker 7.2.4 (2012-09-23)	– ACID, Referential integrity, Transactions, Unicode, SQL interface – Unlimited DB size, 64 TB max table size, 8kB max row size, 1000 max columns per row, 4GB max blob/clob size, 64 kB max CHAR size, 64 bit max NUMBER size, 0001 min DATE value, 9999 max DATE value, 64 max column name size – Temporary table – Indices – B-/B+ – Capabilities – Union, Inner joins, Outer joins, Inner

				selects, Blobs and Clobs – Other objects – Data domain, Cursor, Trigger, Function, Procedure, External routine
HSQLDB [6]	Windows OSX Linux BSD UNIX z/OS	BSD	HSQL Development Group 2.3.1 (2013-10-08)	– ACID, Referential integrity, Transactions, Unicode, SQL interface – 64 TB DB size, Unlimited max table size, Unlimited max row size, Unlimited columns per row, 64 TB max blob/clob size, Unlimited max CHAR size, Unlimited max NUMBER size, 0001-01-01 min DATE value, 9999-12-31 max DATE value, 128 max column name size – Temporary table – Indices – B-/B+ – Capabilities – Union, Intersect, Except, Inner joins, Outer joins, Inner selects, Merge joins, Blobs and Clobs, Common Table Expressions, Parallel Query – Other objects – Data domain, Trigger, Function, Procedure, External routine – Access control – Native network encryption, Enterprise directory compatibility, Password complexity rules, Patch access, Run unprivileged, Separation of duties
Ingres [7]	Windows OSX Linux BSD UNIX	GPL and Proprietary	Ingres Corp. Ingres Database 10 (2010-10-12)	– ACID, Referential integrity, Transactions, Unicode, SQL and QUEL interface – Unlimited DB size, Unlimited max table size, 256 kB max row size, 1024 max columns per row, 2 GB as max blob/clob size, 32 000 B max CHAR size, 64 bit max NUMBER size, 0001 min DATE value, 9999 max DATE value, 256 max column name size – Temporary table – Indices – B-/B+, R-/R+ tree, Hash, Expression, Bitmap – Capabilities – Union, Inner joins, Outer joins, Inner selects, Merge joins, Blobs and Clobs – Other objects – Data domain, Cursor, Trigger, Function, Procedure, External routine – Partitioning – Range, Hash, Composite, List
LucidDB [8]	Windows OSX Linux	GPL v2	The Eigenbase Project 0.9.3 (2010-06-16)	– ACID, Unicode, SQL interface – Indices – B-/B+, Bitmap – Capabilities – Union, Intersect, Except, Inner joins, Outer joins, Inner selects, Merge joins – Other objects – Cursor, Function, Procedure, External routine
MariaDB [9]	Windows OSX Linux BSD UNIX	GPL v2 (LGPL for client-libraries)	MariaDB Community 10.0.14 (2014-09-26)	– ACID, Transactions, Unicode, SQL interface – Indices – B-/B+ – Access control – Native network encryption (SSL), Enterprise directory compatibility (not on Windows), Patch access (partial), Run unprivileged
Postgre SQL [10]	Windows OSX Linux BSD UNIX Android	PostgreSQL (liberal Open Source license)	PostgreSQL Global Development Group 9.3.5 (2014-07-24)	– ACID, Referential integrity, Transactions, Unicode, SQL, GUI and API interface – Unlimited DB size, 32 TB max table size, 1.6 TB max row size, 250-1600 max columns per row, 1 GB as max blob/clob size, 1 GB max CHAR size, Unlimited max NUMBER size, -4713 min DATE value, 5874897 max

				DATE value, 63 max column name size – Temporary table, Materalized view – Indices – B-/B+, R-/R+ tree, Hash, Expression, Partial, Reverse, Bitmap, GiST, GIN, Full-text, Spatial (some exceptions) – Capabilities – Union, Intersect, Except, Inner joins, Outer joins, Inner selects, Merge joins, Blobs and Clobs, Common Table Expressions, Windowing Functions – Other objects – Data domain, Cursor, Trigger, Function, Procedure, External routine – Partitioning – Range, Hash, Composite, List – Access control – Native network encryption, Brute-force protection, Enterprise directory compatibility, Password complexity rules, Patch access, Run unprivileged, Resource limit, Separation of duties, Security certification
SmallSQL [11]	Windows OSX Linux BSD UNIX z/OS	LGPL	SmallSQL 0.21 (2011-06-22)	– Indices – B-/B+
SQL Anywhere [12]	Windows OSX Linux UNIX Android	Proprietary	Sybase 16.0 (2013-04-18)	– ACID, Referential integrity, Transactions, Unicode, SQL interface – 128 TB DB size, Limited by file size - max table size, Limited by file size - max row size, 32767 max columns per row, 2 GB as max blob/clob size, 2 GB max CHAR size, 64 bit max NUMBER size, No DATE type, Unlimited max column name size – Temporary table, Materialized view – Indices – B-/B+, Full-text – Capabilities – Union, Intersect, Except, Inner joins, Outer joins, Inner selects, Merge joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query – Other objects – Data domain, Cursor, Trigger, Function, Procedure, External routine – Access control – Native network encryption, Enterprise directory compatibility, Password complexity rules, Run unprivileged, Audit, Separation of duties, Security certification
SQLite [13]	Windows OSX Linux BSD UNIX AmigaOS Symbian iOS	Public Domain	D. Richard Hipp 3.8.6 (2014-08-15)	– ACID, Referential integrity, Transactions, Unicode (Optional), SQL and API interface – 104 TB DB size, Limited by file size - max table size, Limited by file size - max row size, 45000 max columns per row, 2 GB as max blob/clob size, 2 GB max CHAR size, 64 bit max NUMBER size, 0001-01-01 min DATE value, 9999-12-31 max DATE value, Uknown max column name size – Temporary table – Indices – B-/B+, R-/R+ tree, Partial, Reverse, Full-text, Spatial (some exceptions) – Capabilities – Union, Intersect, Except, Inner joins, Outer joins (LEFT only), Inner selects, Blobs and Clobs

				– Other objects – Trigger, External routine – Access control – Patch access (partial), Run unprivileged, Audit, Resource limit

Remark: Some features and properties may be missing

- ## Relational Database Selection Criteria

The criteria for selection a particular database management system (DBMS) for a given purpose could be clustered in various groups. Here two main factors are considered: support organization and capabilities – both of which are considered to be of equal importance for a final select decision.

Support organization factors:

- **Software license** – in its base lay some of the most motivating reasons for selection of a DBMS: paid vs. non-paid; usage time restrictions; number of clients/machines to run onto; commercial vs. academic (personal) usage restrictions; separate modules / libraries additional restrictions; incorporating into larger projects (systems) as a complete new product or supporting sub-product; re-licensing admissions; etc.
- **Supported operating system (OS)** – depending on the OS as a base platform for running the higher-level applications which in the most cases is of more fundamental role within a whole project planning, the support of that OS by the DBMS' maintainer may well be crucial factor for selection.
- **Latest version release date (latest stable version)** – reveals how current the development process of the maintainer is.
- **Lifespan of development (first stable version release date)** – indicates the release activity consistency over time – how many generations of the product has been produces taking into account all major technologies' changes during the years.
- **Number of stable versions being released during lifespan and releases' frequency** – shows how often a given set of currently used technologies are updated, how regular (active) is the production process and to what degree it correspond in number of enhancement stages to other maintainers' (developers') products.
- **Maintainer (Developer)** – although not a direct objective factor for selection of a DBMS, the maintainer itself given its current status as for the market rating (financial status), stated future plans for development (support), undertaken incorporating initiatives (merging the product in larger systems, frameworks, etc. including with other companies, forming enterprises, etc.) and other activities may well indicate indirectly important clues for proper choice.

Technical features:

The technical features may be divided into two sub-groups. The first one concerns the fundamental features or native capabilities of the DBMS itself. Based on these features the basic functionalities for interconnection with the "outer" world (the whole framework to be developed at hand) are defined which must cover fully the expected system's functions. The second sub-group consists of all limiting factors inside the DBMS – mainly the minimal and maximal size of a particular database property.

Fundamental features:

- Atomicity, Consistency, Isolation, Durability (ACID)
- Referential integrity
- Transactions
- Unicode support

- Supported Interfaces Type
- Temporary table support
- Materialized view
- Supported indices type – most popular B-/B+ tree, R-/R+ tree, Hash, Expression, Partial, Reverse, Bitmap, GiST, GIN, Full-text, Spatial, FOT
- Supported data types – apart from standard (most common) types, e.g. Integer, Floating point, Decimal, String, Binary, Date/Time, Boolean, interest may represent special types of data supported by some DBMS such as PICTURE, GEOMETRY, SEQUENCE, etc. when processing image/audio/video data in a more complete form (context based).
- Database capabilities – Union, Intersect, Except, Inner joins, Outer joins, Inner selects, Merge joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Queries
- Other supported objects - Data Domain, Cursor, Trigger, Function, Procedure, External routine
- Supported partitioning methods – Range, Hash, Composite (Range+Hash), List, Expression
- Access control functionalities - Native network encryption, Brute-force protection, Enterprise directory compatibility, Password complexity rules, Patch access, Run unprivileged, Audit, Resource limit, Separation of duties (RBAC), Security Certification, Label Based Access Control (LBAC)

Limiting factors:

- Maximal DB size
- Maximal table size
- Maximal row size
- Maximum number columns per row
- Maximal Blob/Clob size
- Maximal CHAR size
- Maximal NUMBER size
- Minimal DATE value
- Maximal DATE value
- Maximal column name size

Given the table comparison of the most popular DBMS a precise balanced decision should be made for selecting the proper one for the current project. Some factors to be considered are:

- Type, speed, supported transfer protocols by client's connection
- Centralized (server/cloud) OS/environment
- Number of simultaneously served clients
- Average growth of users in time
- Type of data to be stored (to be compared with the limiting factors of the candidate DBMS) – numbers, text, images/frames, audiosamples, medical signals (packets) – separation of short- and long-term storage data for temporal analysis/decision of immediate action and prolonged analysis of clients' state (possibly for diagnosis, etc.)
- Clients' and operator's user interfaces – search and visualization capabilities, (re-) ordering of data, cross-connectivity among various users' data (personal/public distinction), etc.
- Network and local security level (type)
- Initial lifespan prediction of the developed system
- Compatibility and interoperability demands (changes) in time (possible OS change, ceased/transfered DBMS support by the maintainer, etc.)

4.2 Non-relational Databases – NoSQL

- **Advantages**
  - Can accept varying data in JSON format, no need for predefining a database schema
  - Can handle high rates for data insertions
  - Scale up to handle huge amounts of data
  - CouchDB provides direct REST interface and easy replication
  - MongoDB : Consistency and Partition Tolerance
  - CouchDB : Availability and Partition Tolerance
- **Disadvantages**
  - No common standard for access
  - Can be complex to program the required functions for data retrieval
  - User management in CouchDB is quite difficult
- **Non-relational Databases Description**

In the table below are briefly described some of the most commonly used Non-relational Databases: **CouchDB, MongoDB, Redis, Cassandra, Riak, Accumulo, HBase, Hypertable, Neo4j, ElasticSearch, Couchbase (ex-Membase), Scalaris, VoltDB**

Database	**Supported OS / Impl. language**	**License**	**Maintainer, Latest version**	**Protocol , Features and Properties, Usage**
CouchDB [14]	Android BSD Linux OS X Solaris Windows / Erlang	Apache	Apache Software Foundation 1.6.1 (2014-09-03)	**Protocol** – HTTP/REST **Features and Properties** – DB consistency, ease of use – Bi-directional replication, – continuous or ad-hoc, – with conflict detection, – thus, master-master replication. – MVCC - write operations do not block reads – Previous versions of documents are available – Crash-only (reliable) design – Needs compacting from time to time – Views: embedded map/reduce – Formatting views: lists & shows – Server-side document validation possible – Authentication possible – Real-time updates via changes – Attachment handling – thus, CouchApps (standalone js apps) **Usage** – For accumulating, occasionally changing data, on which pre-defined queries are to be run. Places where versioning is important. – For example: CRM, CMS systems. Master-master replication is an especially interesting feature, allowing easy multi-site deployments.
MongoDB [15]	Linux OS X	AGPL	MongoDB, Inc	**Protocol** – Custom, binary (BSON)

	Solaris Windows / C++	Drivers Apache	2.6.4 (2014-08-11)	**Features and Properties** – Retains some friendly properties of SQL. (Query, index) – Master/slave replication (auto failover with replica sets) – Sharding built-in – Queries are javascript expressions – Run arbitrary javascript functions server-side – Better update-in-place than CouchDB – Uses memory mapped files for data storage – Performance over features – Journaling (with --journal) is best turned on – On 32bit systems, limited to ~2.5Gb – An empty database takes up 192Mb – GridFS to store big data + metadata (not actually an FS) – Has geospatial indexing – Data center aware **Usage** – If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks. – For example: For most things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back.
Redis [16]	BSD Linux OS X Windows / C	BSD	Salvatore Sanfilippo 2.8.16 (2014-09-16)	**Protocol** – Telnet-like, binary safe **Features and Properties** – Blazing fast – Disk-backed in-memory database, – Dataset size limited to computer RAM (but can span multiple machines' RAM with clustering) – Master-slave replication, automatic failover – Simple values or data structures by keys but complex operations like ZREVRANGEBYSCORE. – INCR & co (good for rate limiting or statistics) – Bit operations (for example to implement bloom filters) – Has sets (also union/diff/inter) – Has lists (also a queue; blocking pop) – Has hashes (objects of multiple fields) – Sorted sets (high score table, good for range queries) – Lua scripting capabilities – Has transactions – Values can be set to expire (as in a cache) – Pub/Sub lets one implement messaging **Usage** – For rapidly changing data with a foreseeable database size (should fit mostly in memory). – For example: To store real-time stock prices. Real-time analytics. Leaderboards. Real-time communication. And wherever you used memcached before.
Cassandra [17]	BSD Linux OS X Windows / Java	Apache	Apache Software Foundation 2.1.0 (2014-09-11)	**Protocol** – CQL3 & Thrift **Features and Properties** – Store huge datasets in "almost" SQL – CQL3 is very similar SQL, but with some limitations that

				come from the scalability (most notably: no JOINs, no aggregate functions.) – CQL3 is now the official interface. Don't look at Thrift, unless you're working on a legacy app. This way, you can live without understanding ColumnFamilies, SuperColumns, etc. – Querying by key, or key range (secondary indices are also available) – Tunable trade-offs for distribution and replication (N, R, W) – Data can have expiration (set on INSERT). – Writes can be much faster than reads (when reads are disk-bound) – Map/reduce possible with Apache Hadoop – All nodes are similar, as opposed to Hadoop/HBase – Very good and reliable cross-datacenter replication – Distributed counter datatype. – You can write triggers in Java. **Usage** – When you need to store data so huge that it doesn't fit on server, but still want a friendly familiar interface to it. – For example: Web analytics, to count hits by hour, by browser, by IP, etc. Transaction logging. Data collection from huge sensor arrays.
Riak [18]	Linux BSD Mac OS X Solaris / Erlang & C, some Java Script	Apache	Basho Technologies 2.0.0 (2014-09-02)	**Protocol** – HTTP/REST or custom binary **Features and Properties** – Fault tolerance – Stores blobs – Tunable trade-offs for distribution and replication – Pre- and post-commit hooks in JavaScript or Erlang, for validation and security. – Map/reduce in JavaScript or Erlang – Links & link walking: use it as a graph database – Secondary indices: but only one at once – Large object support (Luwak) – Comes in "open source" and "enterprise" editions – Full-text search, indexing, querying with Riak Search – In the process of migrating the storing backend from "Bitcask" to Google's "LevelDB" – Masterless multi-site replication replication and SNMP monitoring are commercially licensed **Usage** – If you want something Dynamo-like data storage, but no way you're gonna deal with the bloat and complexity. If you need very good single-site scalability, availability and fault-tolerance, but you're ready to pay for multi-site replication. – For example: Point-of-sales data collection. Factory control systems. Places where even seconds of downtime hurt. Could be used as a well-update-able web server.
Accumulo [19]	Linux OS X Unix	Apache	Apache Software Foundation	**Protocol** – Thrift **Features and Properties**

	Windows / Java and C++		1.6.0 (2014-05-02)	– A BigTable with Cell-level security – Another BigTable clone, also runs of top of Hadoop – Originally from the NSA – Cell-level security – Bigger rows than memory are allowed – Keeps a memory map outside Java, in C++ STL – Map/reduce using Hadoop's facitlities (ZooKeeper & co) – Some server-side programming **Usage** – If you need to restict access on the cell level. – For example: Same as HBase, since it's basically a replacement: Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.
HBase [20]	Linux OS X Unix Windows / Java	Apache	Apache Software Foundation 0.98.4 (2014-07-21)	**Protocol** – HTTP/REST (also Thrift) **Features and Properties** – Billions of rows X millions of columns – Modeled after Google's BigTable – Uses Hadoop's HDFS as storage – Map/reduce with Hadoop – Query predicate push down via server side scan and get filters – Optimizations for real time queries – A high performance Thrift gateway – HTTP supports XML, Protobuf, and binary – Jruby-based (JIRB) shell – Rolling restart for configuration changes and minor upgrades – Random access performance is like MySQL – A cluster consists of several different types of nodes **Usage** – Hadoop is probably still the best way to run Map/Reduce jobs on huge datasets. Best if you use the Hadoop/HDFS stack already. – For example: Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.
Hypertable [21]	Linux Mac OS X / C++	GPL 2.0	Hypertable Inc. 0.9.8.1 (2014-09-14)	**Protocol** – Thrift, C++ library, or HQL shell **Features and Properties** – A faster, smaller HBase – Implements Google's BigTable design – Run on Hadoop's HDFS – Uses its own, "SQL-like" language, HQL – Can search by key, by cell, or for values in column families. – Search can be limited to key/column ranges. – Sponsored by Baidu – Retains the last N historical values – Tables are in namespaces – Map/reduce with Hadoop **Usage**

				– If you need a better HBase. – For example: Same as HBase, since it's basically a replacement: Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.
Neo4j [22]	Linux OS X Unix Windows / Java	GPL, some features AGPL/ commercial	Neo Technology 2.1.5 (2014-09-04)	**Protocol** – HTTP/REST (or embedding in Java) **Features and Properties** – Graph database - connected data – Standalone, or embeddable into Java applications – Full ACID conformity (including durable data) – Both nodes and relationships can have metadata – Integrated pattern-matching-based query language ("Cypher") – Also the "Gremlin" graph traversal language can be used – Indexing of nodes and relationships – Nice self-contained web admin – Advanced path-finding with multiple algorithms – Indexing of keys and relationships – Optimized for reads – Has transactions (in the Java API) – Scriptable in Groovy – Online backup, advanced monitoring and High Availability is AGPL/commercial licensed **Usage** – For graph-style, rich or complex, interconnected data. – For example: For searching routes in social relations, public transport links, road maps, or network topologies.
ElasticSea rch [23]	Linux OS X Unix Windows / Java	Apache	Elasticsearch 1.3.3 (2014-09-29)	**Protocol** – JSON over HTTP (Plugins: Thrift, memcached) **Features and Properties** – Advanced Search – Stores JSON documents – Has versioning – Parent and children documents – Documents can time out – Very versatile and sophisticated querying, scriptable – Write consistency: one, quorum or all – Sorting by score – Geo distance sorting – Fuzzy searches (approximate date, etc) – Asynchronous replication – Atomic, scripted updates (good for counters, etc) – Can maintain automatic "stats groups" (good for debugging) – Still depends very much on only one developer (kimchy). **Usage** – When you have objects with (flexible) fields, and you need "advanced search" functionality. – For example: A dating service that handles age difference, geographic location, tastes and dislikes, etc. Or a leaderboard system that depends on many variables.

Couchbase (ex-Membase) [24]	Linux OS X Unix Windows / Erlang & C++	Apache	Couchbase Inc. 2.5.1 (2014-03-31)	**Protocol** – memcached + extensions **Features and Properties** – Memcache compatible, but with persistence and clustering – Very fast (200k+/sec) access of data by key – Persistence to disk – All nodes are identical (master-master replication) – Provides memcached-style in-memory caching buckets, too – Write de-duplication to reduce IO – Friendly cluster-management web GUI – Connection proxy for connection pooling and multiplexing (Moxi) – Incremental map/reduce – Cross-datacenter replication **Usage** – Any application where low-latency data access, high concurrency support and high availability is a requirement. – For example: Low-latency use-cases like ad targeting or highly-concurrent web apps like online gaming (e.g. Zynga).
Scalaris [25]	Windows Linux/ Erlang	Apache	Scalaris 0.7.1 (2014-09-30)	**Protocol** – Proprietary & JSON-RPC **Features and Properties** – Distributed P2P key-value store – In-memory (disk when using Tokyo Cabinet as a backend) – Uses YAWS as a web server – Has transactions (an adapted Paxos commit) – Consistent, distributed write operations – From CAP, values Consistency over Availability (in case of network partitioning, only the bigger partition works) **Usage** – If you like Erlang and wanted to use Mnesia or DETS or ETS, but you need something that is accessible from more languages (and scales much better than ETS or DETS). – For example: In an Erlang-based system when you want to give access to the DB to Python, Ruby or Java programmers.
VoltDB [26]	Linux Mac OS X/ Java, C++	GPL 3	VoltDB Inc. 4.0 (2014-01-26)	**Protocol** – Proprietary **Features and Properties** – Fast transactions and rapidly changing data – In-memory relational database. – Can export data into Hadoop – Supports ANSI SQL – Stored procedures in Java – Cross-datacenter replication **Usage** – Where you need to act fast on massive amounts of

				incoming data. – For example: Point-of-sales data analysis. Factory control systems.

Remark: Some features and properties may be missing

- **Non-relational Database Selection Criteria**

Here are described several important factors that need to be taken into account to make the right choice of NoSQL database:

Storage Type

- For instance, get, put and delete functions are best supported by Key Value systems.
- Aggregation becomes much easier while using Column oriented systems as against the conventional row oriented databases. They use tables but do not have joins.
- Mapping data becomes easy from object oriented software using a Document oriented NoSQL database such as XML or JSON as they use structure document formats.
- Tabular format is replaced and data is stored in graphical format.

Concurrency Control

Concurrency control is what defines how two users can simultaneously edit the same bit of information. It happens quite often that one of the users is locked out and is unable to edit or perform other actions till the active user has finished editing.

- **Locks** - prevent more than one active user to edit an entity such as a document, row or an object.
- **MVCC** (Multi-Version Concurrency Control) - guarantee a read consistent view of the database, but result in conflicting versions of an entity if multiple users modify it at once. MVCC makes it possible for a transaction to seamlessly go through by maintaining many different versions of the object. That means transaction consistency is maintained even if that shows varying snapshots to different users at any given point in time. Any changes made to the database will be shown to others depending which snapshot are they referring to.
- **None –** Atomicity is missing in some systems thereby not providing the same view of the database to multiple users editing the database.
- **ACID –** For reliable database transactions, ACID or Atomicity, Consistency, Isolation, Durability is a safe bet. It allows for pre-screening transactions to avoid conflicts with no deadlocks.

Replication

Replication ensures that mirror copies are always in sync.

- **Synchronous Mode** – Though it is an expensive approach as there is a dependency on the second server to respond, but it always ensures consistency. After receiving response from the second server, the first server sends back the ACK to the client. This ensures data is placed in multiple nodes at the same time.
- **Asynchronous mode** – In this mode, one database gets updated without waiting for the answer from the other database. Two databases could be not consistent in the range of few milliseconds. This should explain why this cost-effective and synchronous replication method is also dubbed as 'Eventually Consistent.'
- **Implementation Language** – Implementation language helps to determine how fast a database will process. Typically NoSQL databases written in low level languages such as C/C++ and Erlang will be the fastest. On the other hand, those written in higher level languages such as Java make customizations easier.

4.3 RDF Databases

- **Advantages**
  - No need for database schema, storing data in RDF triplets
  - Can perform reasoning on the data
  - Standardised language for access (SPARQL)
  - Federated queries, e.g. queries can retrieve data from multiple databases
  - Sesame provides REST interface
- **Disadvantages**
  - Slow input of data
  - Not easy to cope with huge amounts of data
- **RDF Databases Description**

In the table below are given some details about three of the most popular RDF Databases: **Sesame, Jenna, Virtuoso**

Database	**Supported OS / Impl. language**	**License**	**Maintainer, Latest version**	**Features and Properties**
Jena [27]	Linux OS X Unix Windows/ Java	Apache	Apache Software Foundation 2.12.0 (2014-08-02)	– Appropriate for building semantic web applications. – Can be tied to an existing RDBMS such as MySQL or PostgreSQL. – Providing scalable storage and query of RDF datasets using conventional SQL databases (for use in standalone applications, J2EE and other application frameworks) – Provides access to a reasoned that configured to perform reasoning of various degrees. – Can be used to perform simple RDFS reasoning to the more memory intensive OWL-DL reasoning Java framework for building semantic web applications.
Sesame [28]	Linux OS X Unix Windows/ Java	BSD	Aduna 2.7.13 (2014-08-13)	– Supporting both memory-based and a disk-based storage. – Open source framework for storage, inferencing and querying of RDF data. – Matches the features of Jena with the availability of a connection API, inferencing support, availability of a web server and SPARQL endpoint. – Provides support for multiple backends like MySQL and Postsgre. – Sesame Native is the native triple store offering from Sesame. As compared to be Jena's native triple, TDB, it is less scabable.
Virtuoso [29]	AIX FreeBSD HP-UX Linux OS X Solaris Windows / C	GPL v2 and Proprietary	OpenLink Software 7.1 (2014-02)	– Provides command line loaders, a connection API, support for SPARQL and web server to perform SPARQL queries and uploading of data over HTTP. – Scalable to the region of 1B+ triples. – Provides bridges to be used with Jena and Sesame.

4.4 Database analysis and selection

Based on the requirements in the project and considering all the properties and features of the databases reviewed in the previous section, a CouchDB is selected as a storage database for the in-house environment. Specifically, the easy multi-master replication is attractive regarding the context of the project. The following list represents the advantages of CouchDB:

- ***JSON & JavaScript*** – CouchDB stores and serves JSON documents, and utilizes JavaScript to manipulate them during querying/validation via HTTP. This function is primary advantage since eWall platform utilizes Java as a main implementation language;
- ***Notification mechanism*** – trough "`_changes`" API a streaming mechanism can be provided which is excellent feature for implementing notifications to eWall cloud without the need for polling which is connection demanding;
- ***Schema free*** – each document can define its own validation function which introduces flexibility;
- ***Scalability*** – CouchDB is efficient on one machine and through replication can be scaled out to many machines;
- ***Multi-master asynchronous replication*** – documents can be bi-directionally replicated to many instances and every instance can simultaneously modify all of them;
- ***Optimistic locking*** – in some scenario this feature can be very useful. CouchDB stores "_rev" (revision-version) field in every document and this way provides the optimistic locking;
- ***Replication*** - CouchDB can be driven extremely easy by "_replicator" special-db running as a separate process and replicating data between two instances;

Its internal architecture is fault-tolerant, and failures occur in a controlled environment and are dealt with gracefully. Single problems do not cascade through an entire server system but stay isolated in single requests. It was also intended for accumulating, occasionally changing data, on which pre-defined queries are to be run (such as the signals from the particular sensors).

Every database in CouchDB is a master. If a big part of our application is sync between databases that may go offline and online at any time and may require conflict resolution and merging, CouchDB can handle a lot of the low level work. It is also excellent at replicating between multiple nodes, either on-demand or continuously. Thanks to its replication abilities and RESTful API a horizontally can be handled quite easy using mature tools. (Nginx or Apache for reverse proxying, HTTP load balancers, etc.). A map/reduce functions can be developed in JavaScript to precompute queries. The results are built up incrementally on disk which means they only need to be computed once per signal. In other words, queries can be really fast because it only has to do calculations on the signal data recorded since the last time we ran the query. CouchDB trades disk space for performance or in other word the queries can be comparably fast while conserving disk space.

5 Conclusion and planning

This report discussed the metadata generated in the caring home: their format and local storage in a database. Regarding the format, we first presented the metadata dictionary, as it is currently used or envisioned and then we presented the actual JSON messages generated by the perceptual components and sent to the database. The future versions of this deliverable will include a more mature metadata dictionary and examples from all the JSON messages.

The variety of readily available databases imposes challenges in the selection of suitable one for eWall. This requires careful analysis of each DB in terms of performance, available interfaces, licensing, advantages and disadvantages. Such analysis is presented in this document allowing for clear identification of relevant as well as irrelevant to the project criteria for selection. Taking into account eWall requirements, CouchDB as in-house storage was selected. This selection is based on list of clearly identified advantages with one of the main being the readily available interface for the metadata format of the sensors. This format is in RDF triplets and the description of the metadata is in JSON. The metadata for in-house sensors, available up to the current development of the project is presented in a table. Examples of two types of sensors are presented in JSON. Based on this selection of database and metadata format currently is implemented initial prototype consisting of sensors including sensors for presence, accelerometer, gas sensors, etc., feeding the database with data in JSON format. Furthermore CouchDB streams this data to the Cloud platform which runs on on-line server.

Building module for controlling the rate of data requires analysis of the data itself as well as the load on the database. Such analysis was impossible until now because just recently the sensors started generating data. Next steps in the feasible future will include analysis on data rates and based on that building module for automatic control. The control may be based on expert input or using statistical analysis. The first one is mandatory in cases relevant to user's health, such as ECG signals. The second can be implemented where the data is in bursts and preserving all of the data is not vital, but only the relevant information, as for example audio signals. The module should also allow for supporting of future sensors.

Bibliography

[1] http://www.mysql.com/
[2] https://www.oracle.com/database/index.html
[3] http://db.apache.org/derby/
[4] http://www.cubrid.org/
[5] http://www.drizzle.org/
[6] http://hsqldb.org/
[7] https://www.openhub.net/p/ingres
[8] http://luciddb.sourceforge.net/
[9] https://mariadb.org/
[10] http://www.postgresql.org/
[11] http://www.smallsql.de/
[12] http://www.sap.com/pc/tech/database/software/sybase-sql-anywhere/index.html
[13] http://www.sqlite.org/
[14] http://couchdb.apache.org/
[15] http://www.mongodb.org/
[16] http://redis.io/
[17] http://cassandra.apache.org/
[18] http://basho.com/riak/
[19] https://accumulo.apache.org/
[20] http://hbase.apache.org/
[21] http://hypertable.org/
[22] http://neo4j.com/
[23] http://www.elasticsearch.org/
[24] http://www.couchbase.com/
[25] https://code.google.com/p/scalaris/
[26] http://voltdb.com/
[27] https://jena.apache.org/documentation/sdb/
[28] http://www.sesamedatabase.com/
[29] http://virtuoso.openlinksw.com/

Abbreviations

AGPL - Affero General Public License
API - Application Programming Interface
BSD - Berkeley Software Distribution
DB - Database
DBMS - Database Management System
ECG - Electrocardiography
GPL - General Public License
GUI - Grpahical User Interface
HTTP - Hypertext Transfer Protocol
IO - Input/Output
IMA - Integrated Modulus of Accelerometer output
ISA - Integrated Squared output of Accelerometer
JSON - JavaScript Object Notation
OS - Operating System
RDF - Resource Description Framework
REST - Representational state transfer
SQL - Structured Query Language
XML - Extensible Markup Language