

Health Aware

Technical Design Document

Issue 1

0 Preface 1

0.1 Purpose of this document 1

0.2 Use of this document 1

0.3 Overview 2

0.4 Basis of this Document 2

0.5 A Reference Architecture for the IDA Programme 3

0.6 Specific Design Considerations 3

1 Introduction 5

1.1 Purpose 5

1.2 Scope 5

1.3 Definitions, Acronyms and Abbreviations 5

1.4 References 6

1.5 Overview 6

2 System Overview 7

2.1 System Characteristics 7

2.2 System Architecture 7

2.3 Infrastructure Services 9

3 System Context 10

4 System Design 11

4.1 Design Method and Standards 11

4.2 Documentation Standards 12

4.3 Naming conventions 13

4.4 Programming Standards 13

4.5 Software development tools 13

4.6 Outstanding Issues 14

4.7 Decomposition Description 14

5 Component Description 15

5.1 Component Identifier 16

6 Software Requirements Traceability Matrix 19

Document Control 20

Document Signoff 20

Document Change Record 20

0 Preface

0.1 Purpose of this document

This is the complete documentary guide on the building process of the HealthAware web-application.

0.2 Use of this document

This Preface is addressed to the users of this document and is not meant to be retained in any project‑specific Technical Design Document documents based on it.

The remaining sections constitute the contribution from the group members towards the creation of HealthAware

|  |  |
| --- | --- |
| Teammate | Contribution |
| Daniel Szewczyk | * Project lead * Technologies used * Back-end development |
| Mouhand Sultan | * App design & theming * Front-end development * Documentation layout |
| Parmanand Phaskanram |  |
| Alex Gronowski |  |
| Arman Rawat |  |

# Introduction

The health & fitness industry has been steadily growing since the 1970s. Ever since societies consensus on idolizing a healthy body, the general public have related all sorts of goals to physical well-being: happiness, sexual wellbeing, financial success, and even finding your ‘soulmate’ have all been things that are linked to stating in good shape. But… what is good shape?

Besides the current campaign to encourage body positivity over unrealistic goals that started back in 2017, good shape was being a genetically gifted 6” male specimen, and the expectation for women was even more severe. With the surge in expectation, the fitness industry spiralled with TV ad scams, miracle potions, natural brews, gym membership, magazine spreads, and Hollywood fantasies. With technological growth during the last decade, fitness apps have followed suit. Apps like Fitbit, and My health tracker, and Samsung Health have all taken the scene.

Something all these apps do right, is that they are one step closer to helping others get on the right track. Something all these apps do wrong is that they’re bloated and provide an excessive number of features in a package that is meant for mobile. How HealthAware aims to find its place amongst the scene is that is that is it purposely built to strip away all these excessive features and places the user alone, in control.

## Purpose

The purpose of this document is to detail the technologies used, and to meticulously elaborate on how the technologies mentioned are incorporated into our application.

## Definitions, Acronyms and Abbreviations

1. This section should define all terms, acronyms and abbreviations used in this document.

|  |  |
| --- | --- |
| *Schema* | Describes the structure of a system (database) |
| *Data flow diagram* | Expresses how the flow of data is to be transported between object entities in the system |
| *ERD* | Entity Relationship Diagram: expresses how entities co-exist within the schema |
| *Front End* | Describes software development used to create the user interface of the application |
| *Back End* | Describes software development used to run, maintain, and support the application on the server side of the application |
| *Queries* | Requests, usually in code, written to fetch defined data from the database |
| *Two-factor Authentication* | A security measure that requires users to identify themselves through a password key and another form of unique identification |
| *RAD* | Rapid Application Development: a software development approach that favors prototyping over planning |
| *Version Control* | An organized way to control prototyping |
| *HIPAA* | Health Insurance Portability and Accountability Act of 1996) is United States legislation that provides data privacy and security provisions for safeguarding medical information |
| *CIPA* | Children's Internet Protection Act (“CIPA”). CIPA compliance means that schools and. libraries are filtering their Internet services and have implemented formal Internet Safety Policies (also frequently known as Acceptable Use Policies). |

## References

This section should list all the applicable and reference documents, identified by title, author and date. Each document should be marked as applicable or reference. If appropriate, report number, journal name and publishing organisation should be included.

1. The IDA Architecture Guidelines (incorporating, if developed, the IDA Reference Architecture), which should provide the starting point for the system architecture, should be referenced. The elements of the recommended architecture which are used should be described.
2. There should also be a reference to the document, which states how changes to this document are controlled. It is recommended that one of the references highlight the system development model that is to be followed in developing the system, such as Waterfall, V model or RAD. This may reference an internal document or an industry standard work on software development approaches. If an Object Oriented Design (OOD) approach is to be followed a reference to the guidelines to be adopted on developing the design should be provided.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num. | Title (Applicability & Reference) | Author | Date | Issue |
|  |  |  |  |  |
|  |  |  |  |  |

# System Overview

1. This section should briefly introduce the system context and design, and discuss the background to the project. This section may also summarise the costs and benefits of the selected architecture, and may refer to feasibility studies and prototyping exercises. This section should also describe how the design proposed aligns with the IDA Architecture Guidelines[[1]](#footnote-1) and makes use of the outputs of IDA Horizontal Actions and Measures (HAMs).

## System Characteristics

1. The description of the system should be given in terms of the Architecture of the solution that is being implemented with high level data flows described to set the context of the system, i.e. to look at its external interfaces. This section should also set out to ‘characterise’ the system describing aspects of its operation that indicate if the system has, inter alia:

* to operate in real-time or in bursts, linked to month-end reporting, for example
* the nature of the interface to the users of the system
* a large number of concurrent users
* to be highly resilient or fault tolerant
* to provide security features to protect data
* to be scaleable and easily maintainable in the future
* to have any special back-up facilities to protect important data.

## System Architecture

1. This section should describe the Architecture of the system, based where feasible on the IDA Architecture Guidelines.
2. The system under consideration will perhaps be based upon an n-tier, client-server architecture. It is possible that the architecture will have secure managed interfaces or proxies to isolate systems from illegal access. The Architecture may be a simple client-server system in which web technologies are used to provide forms from a simple server that can be filled in remotely by someone, for example, at a border post. With a simple requirement to collect basic information, which can be entered as it is captured by the user, this two-tier architecture may well be quite sufficient.
3. The access layer (client) may have to cater for a range of terminals that may eventually need to access information. It is not impossible to foresee the day when WAP access to information may be required to allow the citizens of Europe direct access to certain pieces of information stored within the overall data systems operated by the European Union. With enlargement, IDA systems may have to handle a wide range of so-called ‘client’ systems ranging from simple database applications (Access) to information being collected from large-scale mainframe applications.
4. Given this likely wide range of means of access, the current emphasis in client-server architectures of breaking down the architecture into several layers will be factors that will have to be addressed in the course of defining solutions that can be rolled out across the increased European Union. The architecture proposed is a tried approach that, when implemented across the European Union, will provide the flexibility to support ever-changing business (often legislatively driven) requirements. Its key feature is that it separates out business logic, client access technology and centrally held data into discrete layers with standard, open interfaces. Today n-tier architectures can often be divided into several tiers as follows:

client or ‘access’ tier – which facilitates access by both external and internal clients through technologies such as web browsers

presentation tier – the layer that accesses information from the other systems as required, and can contain its own ‘business rules’ for simple processes

business tier – contains the business information layer

persistence tier – the database layer, which provides the facilities to store data.

1. Depending upon the solution proposed for the system in question there may be a balance of functions placed into each of these layers. We believe that in the design process it is vital, however, to have a layered architecture as the basis of the solution to facilitate change and adaptation of the system in the future – in other words to protect the investment that has been made in the system as technologies develop in the future. The degree of isolation afforded to elements of the system through adopting a layered approach is significant and can help immunise the project from having to be replaced in full in years to come – some key elements will survive technological developments.
2. Enlargement, both in the medium and long term, could be a crucial factor in the design of IDA systems. Scalability of the architecture could become an issue. With a layered approach the ability to scale to meet future requirements is relatively easy as additional server power can be deployed into the various layers of the system. It may be, for example, that with enlargement, solutions are devised with ‘regional nodes’ which connect administrations not directly into a central server but into a Regional Centre of expertise in a subject area.
3. Such ideas are already common place in current Community systems such as those connected with EIONET. Here regionally based centres of excellence in the processing of environmental data collect and process information prior to it being forwarded into the European Environmental Agency (EEA).
4. This illustrates a key point in the design of the architecture of the systems. Depending upon where the data is going to be processed and analysed, different physical layouts of servers may be required to provide the levels of service required by the users of the system. It may be that there are local nodes that collect information in each administration, and it is then routed to a specialist node and, from there, onto a central repository which contains the master files where regional factors are considered in order to help establish policy in an effective way.

## Infrastructure Services

1. Infrastructure Services should be provided to all applications with a view to reducing the time, cost and risks of development through re-use. To gain full advantage of infrastructure services the requirements of all current applications and the anticipated requirements from future applications should be analysed. These should be fed into the design of the services. The Infrastructure Services can be built incrementally, implementing the most common requirements first, followed by more specialised services. Topics covered in this section might include:

Security

Audit

Performance monitoring and reporting

Error Handling

Debugging

Logging.

# System Design

Due to the lack of experience with technologies used, our team decided on favouring Rapid Application Development(RAD) over most other software development approaches. We used the bulk of the time allotted to complete this project to research and study the technologies that would be used to execute. The specific requirements coupled with a timeline that was designed to aid the team from day one, we split the project up into 3 sections: Planning, Front-end, and Back-end.

## Planing

1. The design method used should be named and referenced. A brief description may be added to aid readers not familiar with the method. Any deviations and extensions of the method should be explained and justified.
2. In this paper we give an example of how client-server technologies can be developed on the basis of a given design paradigm that enables web technologies to be applied to IDA projects offering ease of maintenance and savings in life cycle costs.
3. The design standard might need to be different if more than one method or programming language is involved: for example, if some Javascript design and programming takes place in an HTML project.
4. IDA projects are vulnerable to legislative and political change. For example, an agreement might be reached to focus upon a specific health issue across Europe – where concerted action may be agreed – but which requires the data being collected to change. A key feature of moving towards an OOD-based approach is that it facilitates system change.
5. A key feature of the design process is to attempt to isolate the various ‘layers’ that make up the overall architecture to ensure that changes in one layer have a minimal effect upon other layers in the system, such as for example in the business layer.
6. In following an OOD-based approach it is vital that the design process is able to discriminate between ‘objects’ and ‘classes’. An ‘object’ is a specific instance, containing a set of data and the methods to process the data. A ‘class’ is a generalised description of a group of objects that have the same data item(s) and method organisation.
7. In designing objects there are four important ‘cardinal points’ to be remembered. These are:

attributes – what the object knows

methods – what the object does

states – the changes that occur due to process flow

events – responses to the outside world.

1. Object definitions should be annotated with these items. This will provide the beginnings of a detailed specification document. In addition, each object should be put into a UML diagram set, as this helps summarise the information into small, tight representations of each object in a standard format – there are nine UML diagrams as referenced in Section 1.3. In the course of the process the diagrams will be transformed into a class diagram that represents the relationship between the objects – in effect a blueprint for the structure of the business objects.
2. One area where careful attention to design can reap rewards in the longer term is in the interface between the client and the so-called presentation layer. From its earliest inception in projects using the Smalltalk language the Model View Controller (MVC) paradigm has been one of the most implemented solutions in client-server computing. In implementations based on web technologies, such as JavaServer Pages (JSP) there is much to be gained from separating out elements of the application server architecture. One approach to this is given in an implementation of the MVC paradigm using Servlets or JavaServer Pages.
3. For example, a Servlet (a Java class that operates on the server side to generate output in response to a client request) could be viewed as a ‘controller’ that is responsible for processing the request from the client – we may assume this request arrives in HTTP or HTTP (S) format. The ‘model’ is the representation of the data stored in the database or persistence store and the ‘view’ is the code that decides what to display next in response to the output from the controller. This is described here to encourage readers to look at the MVC paradigm as an example of a way to break out the components of the client-server architecture into building blocks that can be easily modified and maintained, and gain the benefits from an OOD-based approach to the design of the system.

## Front-End Design

1. For a software implementation, this section should contain the standard module header (if necessary) and contain instructions for its completion. In addition this section should define or reference guidelines on the ration of lines of code to comment statements. It may be that these rules highlight specific areas of code where the commentary should be literally line-by-line, as this is a particularly difficult area. Other areas, which may be less difficult, could be commented on a ratio of five lines of code to one line of commentary.

## Back-end Design

1. This section should explain all naming conventions used, and draw attention to any points a maintenance programmer would not expect. A table of the filetypes and the permitted names or extensions for each is recommended for quick reference.
2. Conventions for naming files**,** programs, modules, and possibly other structures such as variables and messages, should all be documented here.

## Software development tools

1. This section should list the tools chosen to assist software development, including testing. The actual software chosen will be heavily dependent upon the language in which the system will be implemented.
2. The list may include:

an application development too;

a configuration manager / builder;

HTML authoring tools;

a word processor for documentation;

a tool for drawing diagrams;

automated testing tools.

1. Prototyping projects might make use of an interpretative tool, such as an incremental compiler/interpreter/debugger.
2. External interfaces may require some of the modules to be pre-compiled.

## Outstanding Issues

1. Provide details of any design issues that remain unresolved at the date of issue of this document. Explain options, pros and cons, and give an estimate of which option is most likely. Outline impact of each option on the rest of the design.

# Component Description

1. For a software implementation, this and the previous section should provide sufficient information for a programmer to produce the software, and for a maintainer, who may not be the developer, to make subsequent changes. The detailed content will depend upon the software tool to be used. The software may be produced using coding statements written by an application programmer. In contrast, it may be automatically generated by an application development tool, or indeed a mixture of both.

# Software Requirements Traceability Matrix

1. This section should contain a table that summarises how each software requirement has been met in this document. The tabular format permits one-to-one and one-to-many relationships to be shown.

|  |  |  |  |
| --- | --- | --- | --- |
| System Req. Number | System Ref. Item | Component Identifier | Component Item |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Document Control

|  |  |
| --- | --- |
| **Title:** | Technical Design Document |
| **Issue:** | Issue 1 |
| **Date:** | 17 January 2001 |
| **Author:** | Dr Dave Sloggett |
| **Distribution:** | EC DG Enterprise – Gavino Murgia Project Team |
| **Reference:** | IDA-MS-TD |
| **Filename:** | IDA-MS-TD-i1 |
| **Control:** | Reissue as complete document only |
|  |  |

Document Signoff

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nature of Signoff** | **Person** | **Signature** | **Date** | **Role** |
| Authors | Dr Dave Sloggett |  |  | Project Member |
| Reviewers | Mark Pillatt |  |  | Consultant |

1. Note that the design MUST not contradict the IDA Architecture Guidelines [↑](#footnote-ref-1)