Universitatea
Transilvania
din Brașov

HABILITATION THESIS

Title: Distributed Applications for Ambient Assisted Living

Domain: Computers and Information Technology

Author: assoc. prof. dr. eng. KRISTÁLY Dominic Mircea

University: Transilvania University of Brașov

BRAŞOV, 2024

CONTENTS

List of abbreviations

AAL	Ambient Assisted Living
API	Application Programming Interface
CORS	Cross-origin resource sharing
CSS	Cascading Style Sheets
GDPR	GDPR - The EU general data protection regulation
HTTP	Hypertext Markup Language
HTTPS	Hypertext Markup Language Secured
IoT	Internet of Things
IR	Infrared
IT	Information technology
JAX-WS	Java API for XML – Web services
JNDI	Java Naming and Directory Interface
JSON	Javascript Object Notation
MQTT	Message Queuing Telemetry Transport
OAuth	Open Authorization
REST	Representational State Transfer
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSL/TLS	Secure Sockets Layer / Transport Layer Security
UDDI	Universal Description, Discovery, and Integration
UI	User interface
WoS	Web of Science
WSDL	Web Services Description Language
WSN	Wireless Sensor Networks
XML	eXtensible Markup Language

(A) Rezumat

Prezenta lucrare prezintă succint și selectiv cercetările și realizările autorului din domeniul vast al aplicațiilor distribuite. Studiile de caz transpuse în tematica domeniului se limitează la aplicații realizate în contextul sistemelor de tip AAL (*Ambient Assisted Living*).

Ambient Assisted Living este un domeniu inovator care utilizează aplicații distribuite pentru a îmbunătăți independența și calitatea vieții persoanelor în vârstă și cu dizabilități. Prin utilizarea tehnologiilor de tip cloud, a rețelelor de senzori și a dispozitivelor *Internet of Things* (IoT), AAL oferă un suport cuprinzător, care poate fi integrat în activitățile zilnice. Aceste tehnologii permit monitorizarea continuă și prelucrarea datelor în timp real, asigurând asistență eficientă și promovând un mediu de autonomie și siguranță pentru utilizatorii săi.

Capitolul 1 al lucrării prezintă fundamente teoretice ale aplicațiilor distribuite și include scurte descrieri ale proiectelor utilizate ca studii de caz în lucrarea de față. Astfel, sunt prezentate modelele arhitecturale principale utilizate în proiectarea aplicațiilor distribuite și tehnologiile care facilitează dezvoltarea acestora.

Arhitectura aplicației distribuite este foarte importantă pentru implementarea unor sisteme eficiente și scalabile. Printre arhitecturile elementare se numără modelul client-server, care implică interacțiunea mai multor clienți cu un server central, și rețelele peer-to-peer (P2P), unde nodurile funcționează atât ca clienți cât și ca servere, îmbunătățind reziliența și distribuția resurselor. Modelele arhitecturale mai complexe, cum ar fi arhitecturile multi-strat și microservicii, permit separarea preocupărilor pe diferite straturi, facilitând întreținerea mai ușoară și o scalabilitate mai bună.

Aplicațiile distribuite pentru AAL integrează o varietate de tehnologii avansate pentru a gestiona și analiza datele colectate din diverse surse. Platformele de *cloud computing* joacă un rol vital în stocarea și prelucrarea datelor, permițând scalabilitatea și accesibilitatea pe diferite dispozitive.

Capitolul 2 dezvoltă subiectul arhitecturii bazate pe servicii (*Service Oriented Architecture* - SOA), care reprezintă o abordare de proiectare ce facilitează dezvoltarea și integrarea sistemelor prin definirea serviciilor ca funcționalități bine definite, accesibile printr-o rețea. Aceste servicii permit organizațiilor să construiască aplicații flexibile capabile să interacționeze cu alte aplicații în diverse rețele printr-un protocol stabilit. Arhitectura bazată pe servicii subliniază principii precum reutilizabilitatea, modularitatea și interoperabilitatea, care ajută la reducerea costurilor de dezvoltare prin minimizarea redundanței și permițând serviciilor să fie slab cuplate (*loosely-coupled*) pentru modificări și înlocuiri mai ușoare.

Capitolul evidențiază modul în care acest tip de arhitectură a fost folosit în realizarea proiectului FOOD (*Framework for Optimizing the prOcess of FeeDing*). Sunt prezentate serviciile web dezvoltate și structura datelor consumate/generate.

Capitolul 3 tratează arhitectura bazată pe microservicii ce reprezintă o schimbare strategică față de arhitecturile tradiționale de aplicații monolitice, oferind o abordare modulară prin împărțirea aplicațiilor în unități mai mici, care pot fi rulate independent. Această arhitectură îmbunătățește flexibilitatea și ușurința întreținerii, ceea ce este deosebit de avantajos în medii cloud, facilitând o scalabilitate și gestionare mai bune. Structura segmentată a microserviciilor permite actualizări și modificări mai eficiente, deoarece schimbările într-un serviciu nu necesită modificări în altele.

Arhitectura bazată pe microservicii nu doar că susține o disponibilitate ridicată și o gestionare eficientă, dar permite, de asemenea, fiecărei echipe din cadrul unei organizații să lucreze independent pe diferite servicii, sporind productivitatea generală. Fiecare microserviciu poate fi scalat/multiplicat independent în funcție de cerere, și în cazul unei defecțiuni, doar serviciul afectat este întrerupt, lăsând restul sistemului operațional. Acest stil arhitectural permite utilizarea unor tehnologii diverse, îmbunătățind adaptabilitatea și robustețea sistemului în timp. Astfel, microserviciile sunt extrem de eficiente pentru aplicații care necesită actualizări frecvente și scalabilitate, oferind un cadru fiabil care susține integrarea și desfășurarea continuă.

Implementarea microserviciilor în sisteme precum SAVE (*SAfety of elderly people and Vicinity Ensuring*) ilustrează beneficiile practice ale acestei abordări, incluzând capacități de gestionare îmbunătățite și flexibilitate sporită pentru adaptare la cerințe în schimbare. Microserviciile în astfel de contexte comunică prin interfețe standardizate, folosind adesea protocoale HTTP sau HTTPS, cu JSON pentru schimbul de date, asigurând interacțiuni facile între servicii. Această arhitectură nu doar că simplifică dezvoltarea și accelerează desfășurarea de noi funcționalități, dar profită și de avantajele infrastructurii cloud, făcând-o ideală pentru aplicații dinamice, scalabile care necesită soluții robuste, tolerante la erori.

În **Capitolul 4** sunt prezentate modul de utilizare și avantajele utilizării dispozitivelor de tip *Internet of Things* (IoT) în cadrul sistemelor AAL. *Internet of Things* transformă sistemele AAL prin integrarea obiectelor cotidiene cu internetul, permițându-le să trimită și să primească date. Această tehnologie sprijină persoanele în vârstă și pe cele cu dizabilități, îmbunătățindu-le calitatea vieții prin intermediul sistemelor de suport inteligente. IoT în AAL utilizează senzori și actuatoare pentru a colecta și acționa pe baza datelor, aceste componente fiind legate printr-o rețea de comunicații robustă. Această configurație nu doar că asigură un răspuns în timp util, dar facilitează și interacțiuni personalizate, esențiale pentru adaptarea serviciilor la nevoile individuale.

Aplicarea IoT în AAL acoperă mai multe domenii critice. În monitorizarea sănătății, dispozitivele IoT oferă supraveghere continuă a parametrilor vitali, oferind feedback în timp real, crucial pentru gestionarea proactivă a sănătății. Pentru siguranță, aceste tehnologii detectează anomalii precum căderile, declanșând răspunsuri de urgență necesare.

Capitolul include două studii de caz. Primul este preluat din proiectul HELICOPTER (*Healthy Life support through Comprehensive Tracking of individual and Environmental Behaviors*) și prezintă ecosistemul software necesar conectării la o aplicație server, prin internet, a unei rețele de senzori locală (bazată pe protocolul Zigbee) folosind un mini-sistem de calcul de tip Gateway.

Al doilea studiu de caz, preluat din proiectul SAVE, detaliază realizarea hardware și software a unui dispozitiv de preluare a datelor colectate de un sistem de monitorizare și transmiterea lor, prin internet, la serviciile sistemului SAVE.

Capitolul 5 oferă o scurtă introducere în subiectul bazelor de date și a sistemelor de gestiune a bazelor de date.

Bazele de date sunt componente integrale ale sistemelor informatice moderne, servind ca depozite critice de date pentru diverse sectoare, inclusiv afaceri, sănătate și educație. Acestea au evoluat de la modele ierarhice și de rețea în anii 1960 la bazele de date relaționale mai flexibile și, mai recent, la modelele NoSQL și NewSQL pentru a răspunde nevoilor erei internetului și a volumelor de date mari.

Sistemele de gestiune al bazelor de date relaționale se bazează pe un model structurat care organizează datele în tabele. Acest format sprijină manipularea și recuperarea eficientă a datelor, esențiale pentru multe aplicații de afaceri. De la crearea sa de către Edgar F. Codd în 1970, aceste sisteme au crescut pentru a sprijini sisteme distribuite și procesarea analitică online, printre alte funcții avansate. În ciuda provocărilor cu gestionarea volumelor mari de date și a datelor nestructurate, sistemul relațional rămâne crucial în gestionarea datelor, cu o evoluție continuă pentru o adaptabilitate și funcționalitate mai bune.

În aplicații specifice, cum ar fi sistemul FOOD, bazele de date joacă un rol esențial în gestionarea datelor din rețelele de senzori. Acest lucru implică o structură ierarhică în care datele de nivel scăzut de la senzori sunt stocate și gestionate într-o bază de date MySQL, permițând controlul operațional în timp real și analiza datelor istorice. Gestionarea de nivel înalt a acestor date folosește o bază de date relațională sincronizată pentru serviciile web pentru a facilita interacțiunea utilizatorilor și monitorizarea sistemului. Astfel de arhitecturi evidențiază natura dinamică și scalabilă a bazelor de date moderne în sprijinirea sistemelor și aplicațiilor complexe, subliniind integrarea continuă a tehnologiei bazelor de date cu alte inovații pentru a satisface cerințele tehnologice în schimbare.

Capitolul 6 tratează subiectul interfeței om-mașină (interfețe utilizator). Interacțiunea om-calculator în proiectarea interfeței utilizator se concentrează pe optimizarea interacțiunii între utilizatori și dispozitive, având ca scop simplificarea, eficiența și ușurința în utilizare. Un design bun al interfeței utilizator asigură că utilizatorii pot finaliza sarcinile eficient fără complicații inutile, sprijinit de designul grafic pentru a îmbunătăți utilizabilitatea. Procesele de proiectare UI echilibrează funcționalitatea tehnică cu elementele estetice pentru a satisface eficient nevoile utilizatorilor în schimbare.

Pentru utilizatorii în vârstă, considerațiile de proiectare se ajustează pentru a se adapta la declinurile tipice legate de vârstă în ceea ce privește vederea, memoria, atenția și abilitățile motorii. De exemplu, interfețele pot folosi fonturi mai mari, fără corpuri de literă de tip serif și scheme de culori cu contrast ridicat pentru a ajuta la lizibilitate, simplifica sarcinile pentru a minimiza încărcătura cognitivă și proiecta alerte audibile în gamele de frecvențe inferioare pentru a se adapta sensibilităților auditive. Aceste adaptări ajută la menținerea utilizabilității dispozitivelor și aplicațiilor printre adulții mai în vârstă, îmbunătățind experiența lor de interacțiune fără a-i copleși cu funcționalități complexe.

Mai mult, planificarea interfețelor utilizator implică conceptualizarea modului în care serviciile sunt prezentate și accesate prin interfețe, utilizând metode precum hărți conceptuale, planuri detaliate și machete pentru a alinia funcționalitățile tehnice cu nevoile utilizatorilor. De exemplu, în aplicații specializate precum sistemul FOOD, interfața utilizator este adaptat pentru a asista vârstnicii în interacțiunea fără probleme cu diverse funcționalități, cum ar fi ciclurile de gătit pe electrocasnice inteligente, implementate în mai multe limbi pentru a deservi o bază diversă de utilizatori.

În **Capitolul 7** este abordat subiectul *Cloud computing*, care este esențial în îmbunătățirea sistemelor AAL prin oferirea de resurse de calcul scalabile, flexibile și eficiente. Această tehnologie susține diverse aplicații AAL, inclusiv monitorizarea sănătății, răspunsul la urgențe și realizarea casei inteligente. În monitorizarea sănătății, platformele cloud colectează și analizează date de la dispozitive precum ceasurile inteligente, permițând furnizorilor de servicii medicale să intervină în timp util. În caz de urgențe, sistemele cloud notifică prompt îngrijitorii și serviciile de urgență, furnizând informații cruciale despre starea utilizatorului. Pentru casele inteligente, cloud computing facilitează controlul sistemelor casnice, cum ar fi iluminatul și încălzirea, îmbunătățind confortul și siguranța pentru persoanele în vârstă.

Beneficiile utilizării cloud computing în AAL includ capacități sofisticate de gestionare a datelor și analize, scalabilitatea resurselor pentru a satisface cerințele variabile și eficiența costurilor prin minimizarea cheltuielilor pentru întreținerea hardware-ului și software-ului. Aceste avantaje fac tehnologiile AAL mai accesibile și eficiente, sprijinind gestionarea proactivă a sănătății și îmbunătățind calitatea vieții utilizatorilor.

Cu toate acestea exista provocări, cum ar fi asigurarea confidențialității și securității datelor sensibile, menținerea unei conexiuni fiabile și realizarea unei integrări facile între dispozitive și platforme diverse. Aceste provocări subliniază necesitatea unor măsuri de securitate robuste, servicii internet fiabile și protocoale standardizate pentru a asigura implementarea eficientă a sistemelor AAL bazate pe cloud. În plus, unelte precum *Node-RED* oferă programare bazată pe fluxuri pentru a facilita dezvoltarea aplicațiilor AAL, exemplificând astfel integrarea tehnicilor avansate de cloud computing în susținerea și scalarea serviciilor AAL.

În ultima parte a lucrării sunt prezentate realizările academice și de cercetare ale autorului și coordonatele viitoare de cercetare și împlinire profesională.

(A-i) Summary

This paper succinctly and selectively presents the author's research and achievements in the vast field of distributed applications. The case studies framed into the domain's theme are limited to applications made in the context of *Ambient Assisted Living* (AAL) systems.

Ambient Assisted Living is an innovative field that uses distributed applications to enhance the independence and quality of life for elderly and disabled individuals. By leveraging cloud technologies, sensor networks, and *Internet of Things* (IoT) devices, AAL provides comprehensive support that can be integrated into daily activities. These technologies enable continuous monitoring and real-time data processing, ensuring effective assistance and promoting an environment of autonomy and safety for its users.

Chapter 1 of the paper presents the theoretical foundations of distributed applications and includes brief descriptions of the projects used as case studies in this work. Thus, the main architectural models used in designing distributed applications and the technologies that facilitate their development are presented.

The architecture of distributed applications is very important for implementing efficient and scalable systems. Among the basic architectures are the client-server model, which involves the interaction of multiple clients with a central server, and peer-to-peer (P2P) networks, where nodes function both as clients and servers, improving resilience and resource distribution. More complex architectural models, such as multi-tier and microservices architectures, allow for the separation of concerns across different layers, facilitating easier maintenance and better scalability.

Distributed applications for AAL integrate a variety of advanced technologies to manage and analyse data collected from various sources. Cloud computing platforms play a vital role in data storage and processing, allowing scalability and accessibility across different devices.

Chapter 2 develops the topic of *Service-Oriented Architecture* (SOA), which is a design approach that facilitates system development and integration by defining services as well-defined business functionalities accessible through a network. These services allow organizations to build flexible applications capable of interacting with other applications across various networks through a set protocol. Service-Oriented Architecture emphasizes principles such as reusability, modularity, and interoperability, which help in reducing development costs by minimizing redundancy and allowing services to be loosely coupled for easier modifications and replacements.

The chapter highlights how this type of architecture was used in the implementation of the FOOD project (*Framework for Optimizing the prOcess of FeeDing*). The web services developed, and the structure of the consumed/generated data are presented.

Chapter 3 addresses the architecture based on microservices, which represents a strategic shift from traditional monolithic application architectures, offering a modular approach by dividing applications into smaller, independently runnable units. This architecture improves flexibility and ease of maintenance, which is particularly advantageous in cloud environments, facilitating better scalability and management. The segmented structure of microservices allows for more efficient updates and modifications, as changes to one service do not require alterations to others.

Microservices architecture not only supports high availability and efficient management but also allows each team within an organization to work independently on different services, boosting overall productivity. Each microservice can independently scale based on demand, and in the event of a fault, only the affected service is disrupted, leaving the rest of the system operational. This architectural style allows the use of diverse technologies, improving the adaptability and robustness of the system over time. Thus, microservices are highly effective for applications requiring frequent updates and scalability, providing a reliable framework that supports continuous integration and deployment.

Implementing microservices in systems like SAVE (*SAfety of elderly people and Vicinity Ensuring*) illustrates the practical benefits of this approach, including improved management capabilities and increased flexibility to adapt to changing requirements. Microservices in such contexts communicate through standardized interfaces, often using HTTP or HTTPS protocols, with JSON for data exchange, ensuring seamless interactions between services. This architecture not only simplifies development and accelerates the deployment of new functionalities but also leverages cloud infrastructure advantages, making it ideal for dynamic, scalable applications that require robust, fault-tolerant solutions.

In **Chapter 4**, the use and advantages of *Internet of Things* (IoT) devices in AAL systems are presented. The *Internet of Things* transforms AAL systems by integrating everyday objects with the internet, allowing them to send and receive data. This technology supports the elderly and people with disabilities, enhancing their quality of life through intelligent support systems. IoT in AAL employs sensors and actuators to collect and act upon data, these components being connected through a robust communication network. This setup not only ensures a timely response but also facilitates personalized interactions, essential for adapting services to individual needs.

The chapter includes two case studies. The first is taken from the HELICOPTER project (*Healthy Life support through Comprehensive Tracking of individual and Environmental Behaviors*) and showcases the software ecosystem required to connect to a server application, via the internet, a local sensor network (based on the *Zigbee* protocol) using a mini-computer Gateway system.

The second case study, taken from the SAVE project, details the hardware and software implementation of a device for collecting data from a monitoring system and transmitting them, via the internet, to the SAVE system services.

Chapter 5 offers a brief introduction to the topic of databases and database management systems.

Databases are integral components of modern information systems, serving as critical repositories of data for various sectors, including business, health, and education. They have evolved from hierarchical and network models in the 1960s to more flexible relational databases and, more recently, to NoSQL and NewSQL models to meet the needs of the internet and big data era.

Relational database management systems are based on a structured model that organizes data in tables. This format supports efficient data manipulation and retrieval, essential for many business applications. Since its creation by Edgar F. Codd in 1970, these systems have grown to support distributed systems and online analytical processing, among other advanced functions. Despite the challenges with managing large volumes of data and unstructured data, the relational system remains vital in data management, with ongoing evolution expected for better adaptability and functionality.

In specific applications, such as the FOOD system, databases play an essential role in managing data from sensor networks. This involves a hierarchical structure where low-level data from sensors are stored and managed in a MySQL database, allowing for real-time operational control and historical data analysis. High-level management of this data uses a relational database synchronized with web services to facilitate user interaction and system monitoring. Such architectures highlight the dynamic and scalable nature of modern databases in supporting complex systems and applications, underscoring the continuous integration of database technology with other innovations to meet changing technological demands.

Chapter 6 addresses the topic of human-computer interaction (user interfaces). Human-computer interaction in user interface design focuses on optimizing the interaction between users and devices, aiming for simplicity, efficiency, and ease of use. Good user interface (UI) design ensures that users can complete tasks efficiently without unnecessary complications, supported by graphic design to enhance usability. UI design processes balance technical functionality with aesthetic elements to effectively meet changing user needs.

For elderly users, design considerations adjust to accommodate typical age-related declines in vision, memory, attention, and motor skills. For example, interfaces may use larger, sans-serif fonts and high-contrast color schemes to aid readability, simplify tasks to minimize cognitive load, and design audible alerts in lower frequency ranges to accommodate auditory sensitivities. These adaptations help maintain the usability of devices and applications among older adults, enhancing their interaction experience without overwhelming them with complex functionalities.

Moreover, user interface planning involves conceptualizing how services are presented and accessed through interfaces, using methods such as conceptual maps, wireframes, and mock-ups to align technical functionalities with user needs. For example, in specialized applications such as the FOOD system, the user interface is tailored to assist the elderly in seamlessly interacting with various functionalities, such as cooking cycles on smart appliances, implemented in multiple languages to serve a diverse user base.

In **Chapter 7**, the topic of cloud computing is addressed, which is essential in enhancing AAL systems by providing scalable, flexible, and efficient computing resources. This technology supports various AAL applications, including health monitoring, emergency response, and smart home implementation. In health monitoring, cloud platforms collect and analyse data from devices such as smartwatches, allowing healthcare providers to intervene in a timely manner. In emergencies, cloud systems promptly notify caregivers and emergency services, providing important information about the user's condition. For smart homes, cloud computing facilitates the control of home systems, such as lighting and heating, enhancing comfort and safety for the elderly.

The benefits of using cloud computing in AAL include sophisticated data management and analysis capabilities, scalability of resources to meet varying demands, and cost efficiency by minimizing expenses for hardware and software maintenance. These advantages make AAL technologies more accessible and effective, supporting proactive health management and improving the quality of life for users.

However, there are challenges, such as ensuring the confidentiality and security of sensitive data, maintaining reliable connectivity, and achieving easy integration between diverse devices and platforms. These challenges underscore the need for robust security measures, reliable internet services, and standardized protocols to ensure the effective implementation of cloud-based AAL systems. Additionally, tools such as *Node-RED* offer flow-based programming to facilitate the development of AAL applications, exemplifying the integration of advanced cloud computing techniques in supporting and scaling AAL services.

In the final part of the paper, the author's academic and research achievements and future research and professional fulfilment coordinates are presented.

(B) Scientific and professional achievements and the evolution and development

(B-i) Scientific and professional achievements

1. Introduction to distributed applications for Ambient Assisted Living

Ambient Assisted Living (*AAL*) is an emerging field that leverages technology to assist people, typically the elderly or those with disabilities, in their daily activities, enhancing their quality of life and promoting independent living.

Distributed applications, or distributed systems, refer to software systems whose components are located on multiple networked computers which communicate and coordinate their actions by passing messages to one another. The components interact with each other to achieve a common goal. This architectural style allows for system scalability, reliability, and fault tolerance. Through such systems, resources and workloads can be distributed across multiple machines, which can help enhance performance and provide redundancy in case of system failures.

Distributed applications in AAL involve a wide range of subjects and technologies to create comprehensive systems that support users in a seamless and integrated manner. The key subjects linked to distributed applications in the AAL context are:

- *Sensor Networks*: Involves the deployment of various sensors (motion, temperature, health monitors) to gather data about the user's environment and health status. Sensor data is used to monitor activities, detect emergencies, and even predict potential health issues.
- *Internet of Things* (IoT): IoT technologies enable the connectivity of everyday objects to the internet, allowing them to send and receive data. In AAL, IoT devices can include smart home devices, wearable technologies, and medical monitoring equipment.
- *Data Analytics*: Utilizes machine learning and data mining techniques to analyse the data collected from sensors and devices to understand patterns, make predictions, and provide actionable insights.
- *Cloud Computing*: Provides the infrastructure for data storage and computing power. Cloud platforms can host applications and manage data from multiple sources, ensuring that information is available and scalable across different devices and locations.
- *Mobile Computing*: Involves the use of smartphones, tablets, and other portable devices to access, monitor, and control AAL services. Mobile applications can provide interfaces for users, caregivers, and healthcare providers to interact with the AAL system.

- *Security and Privacy*: Covers the methods and technologies used to protect data and preserve the privacy of individuals. This includes encryption, secure data transmission, and compliance with regulations like GDPR.
- *Human-Computer Interaction* (*HCI*): Focuses on designing user interfaces that are accessible and easy to use for elderly or disabled users. HCI in AAL might involve adaptive interfaces, voice recognition, and touchless interaction technologies.
- *Ubiquitous and Pervasive Computing*: Involves creating a computing environment where AAL services are seamlessly integrated into everyday objects and activities. This subject area explores the development of systems that are always available but unobtrusive.
- *Robotics*: Includes the use of robotic systems for physical assistance, such as mobility aids, automated delivery systems, or robots that can perform tasks like cleaning or medication management.
- *Telemedicine and E-health*: Involves the use of telecommunications technology to provide remote health care. It includes the management of chronic conditions, remote consultations, and remote monitoring.
- *Network Communications*: Studies the protocols and technologies for data transmission over networks, ensuring reliable and timely communication between devices in an AAL system.
- *Software Engineering*: Focuses on the development methodologies, tools, and practices necessary to design, implement, and maintain reliable AAL applications.

1.1. Fundamentals of distributed applications

Distributed application (also known as distributed systems) are complex networks of interconnected computers that collaborate to achieve a common objective. These systems are fundamental to modern computing and are essential for running large-scale applications across multiple physical locations. The study of distributed systems involves understanding the principles and challenges associated with designing, building, and maintaining systems that are both efficient and reliable.

The foundational theory of distributed systems revolves around data consistency, fault tolerance, and scalability. Achieving consistency in distributed systems means ensuring that all nodes, or computers, in the system agree on the data state at any given time. This is challenging because each node operates independently and may not have immediate access to the state changes made by others. Techniques such as consensus algorithms are pivotal in maintaining consistency. These algorithms help systems agree on a single value in the presence of failures, ensuring that the system continues to operate correctly even when some components fail. [1] [2]

Fault tolerance is another critical aspect of distributed systems, ensuring the system continues to function in the face of hardware or software failures. Redundancy is a common approach, where critical components are duplicated so that if one fails, another can take over. Another approach involves implementing failover mechanisms to automatically transfer control to operational systems without user intervention. [3]

Scalability in distributed systems can be achieved through load balancing and resource allocation strategies. Systems must be designed to accommodate growth without performance degradation. Techniques include horizontal scaling (adding more nodes to the system) and vertical scaling (adding more resources to existing nodes). [4]

Moreover, distributed systems must address network issues such as latency and bandwidth limitations. Latency can significantly affect system performance, especially in real-time applications. Solutions involve optimizing communication protocols and data compression techniques to reduce the amount of data transferred and improve response times. [5]

Finally, security concerns in distributed systems are paramount, especially as these systems often handle sensitive data over potentially insecure networks. Security mechanisms must ensure data integrity, confidentiality, and availability. Common practices include implementing cryptographic protocols for secure data transmission and designing robust authentication and authorization techniques. [6]

1.2. Architectural models for distributed systems

Architectural models for distributed systems delineate the structural layout that guides the organization, interaction, and communication among system components distributed across multiple hardware resources. These models are important for developing efficient, robust, and scalable distributed systems.

1.2.1. Client-Server model

The client-server model is one of the most traditional and widely used architectures in distributed systems. In this model, multiple clients request services from a server, and the server processes these requests. This architecture is inherently asymmetrical, with servers providing resources and services to clients that initiate communication sessions. The client-server model is scalable in terms of adding more servers, but it can become a bottleneck if a single server must handle all requests. [5]

1.2.2. Peer-to-Peer model

In contrast to the client-server model, the peer-to-peer (P2P) architecture is characterized by the distribution of tasks and services among multiple nodes that equally share the responsibilities of data processing and service provision. Nodes in a P2P network both provide and consume resources, acting simultaneously as clients and servers. This model enhances fault tolerance and reduces risks of bottlenecking. However, managing data consistency and security can be challenging due to the decentralized nature of the architecture. [4]

1.2.3. Multi-tier architecture

Multi-tier architectures, often seen in enterprise applications, involve dividing tasks into layers, each responsible for a specific aspect of the application. Typically, these include a presentation layer, an application logic layer, and a data management layer. This separation allows for independent scaling and maintenance of each layer, enhancing the system's responsiveness and flexibility. This model is particularly well-suited for web applications and services. [3]

1.2.4. Service-oriented architecture

Service-Oriented Architecture (SOA) is based on the concept of service providers and consumers interacting through well-defined interfaces and contracts. Services are loosely coupled to ensure that changes in one service do not require changes in others. SOA promotes interoperability and modularity, enabling developers to build applications that combine services from various sources over a network. This model supports reusability and can dynamically respond to changing business requirements. [7]

1.2.5. Microservices architecture

Emerging from SOA, the microservices architecture structures an application as a collection of small, autonomous services modelled around a business domain. Each microservice runs in its own process and communicates with other services through lightweight mechanisms, often an HTTP-based application programming interface (API). Microservices allow for continuous delivery and deployment of large, complex applications. However, they introduce challenges in coordinating services and managing data consistency. [8]

1.2.6. Space-based architecture

Space-Based Architecture (SBA) addresses scalability issues in traditional architectures by eliminating the central database and instead spreading data across the network, in memory, to improve performance. It integrates the processing and storage into the same node, avoiding data access bottlenecks. SBA is particularly useful for high-performance and real-time applications where traditional data storage models might be a limiting factor. [9]

1.3. Main technologies supporting distributed applications

Distributed applications leverage a network of computers to achieve enhanced performance, reliability, and scalability. These applications depend on a variety of underlying technologies that enable efficient communication, data consistency, fault tolerance, and resource management across distributed networks. This introduction explores the critical technologies that support the operation and development of distributed applications, highlighting their importance and implementation in contemporary systems.

1.3.1. Communication middleware

Communication middleware provides essential services and capabilities to handle data transmission between distributed components. This includes message queuing systems, publish/subscribe systems, and remote procedure calls (RPCs). Technologies such as *Apache Kafka* and *RabbitMQ* facilitate robust message queuing mechanisms that support complex messaging patterns with high throughput and fault tolerance. [10]

1.3.2. Relational databases in cloud environments

Relational databases in cloud environments represent a significant evolution in the architecture of distributed applications, offering scalability, availability, and flexibility that are crucial for modern software development. In distributed settings, these databases leverage cloud-specific features such as automatic replication, load balancing, and on-demand scalability to ensure consistent performance across geographically dispersed data centres. The integration of relational databases in the cloud facilitates a more robust approach to data management, supporting complex transactions and maintaining strong consistency models despite the inherent challenges of latency and partial failures typical in distributed systems [11].

1.3.3. Distributed databases

Distributed databases manage data across multiple computing nodes to ensure high availability, fault tolerance, and quick response times. NoSQL databases like Cassandra and MongoDB offer scalable solutions for managing large volumes of data with eventual consistency models that are suitable for distributed environments. [12]

1.3.4. Consensus protocols

Consensus protocols are critical for maintaining consistency across distributed processes. These protocols ensure that all nodes in a distributed system agree on a single data value or a sequence of operations, even in the presence of failures.

1.3.5. Load balancing and scalability tools

Load balancers distribute workloads across multiple computing resources to optimize resource use, maximize throughput, and minimize response time. Technologies such as *NGINX* and hardware-based load balancers play a crucial role in managing traffic and services in distributed architectures [13]. Scalability is further enhanced through containerization and orchestration platforms like *Docker* and *Kubernetes*, which manage containerized applications in various environments [14].

1.3.6. Virtualization and cloud services

Virtualization technology, including hypervisors and virtual machines, along with cloud services, forms the backbone of modern distributed applications by providing flexible, scalable, and efficient resources. Cloud service providers such as *AWS*, *Azure*, and *Google Cloud* offer a range of services that support distributed application deployment and management.

1.3.7. Security mechanisms

Security in distributed applications is enforced through cryptographic techniques, secure communication protocols, and identity and access management systems. Technologies like SSL/TLS for secure data transmission and *OAuth* for authorization are fundamental in protecting data and ensuring trusted interactions among distributed components [15].

1.3.8. Monitoring and management tools

Monitoring and management tools are essential for maintaining the health and performance of distributed systems. Tools such as *Prometheus* for monitoring and *Ansible* for configuration management help administrators oversee and control distributed infrastructures effectively [16].

1.4. AAL projects for case studies

1.4.1. FOOD Project

The aim of FOOD project (*Framework for Optimizing the prOcess of FeeDing*) is the development of specific AAL (Ambient Assisted Living) services, dedicated to the kitchen environment, to support elderly people in carrying out food-related daily living activities and interacting with home appliances in a much simpler, safer, and rewarding way. The proposal addresses elderly people with a sufficient level of autonomy for independent life, if suitably supported. It aims at preserving and enhancing independence of elderly people in all aspects of daily life (addressing activities at home, security, health care control), to guarantee them the possibility of taking active part in the "self-serve" society (ability to access information and negotiate and or be supported for getting necessary items if mobility out of the house is a problem), and to secure social contacts and/or support, when necessary.

The devised solution consists of a home-based system that enables elderly people to deal with feeding and food-related tasks in a safe, effective, and rewarding way. It is based on the seamless integration of sensors, intelligent appliances able to offer functionalities in the house and Internet based services and applications, able to give access, through a natural interface, to information and communication in different social environments. Its innovation lies in the integration and cooperation of Internet of things, Semantic Web, and Web 2.0. The availability of relevant data from sensors on people and their environment and the cooperation of artificial and human intelligence through the network will contribute to support independence of people. Moreover, it is supposed that the quality of the end-users' everyday life will improve not only due to the support in crucial activities in the house, but also for the possibility of interaction with the outside world both for practical purposes (e.g. ecommerce, e-government, etc.) and for socializing.

The FOOD system relies on a technical infrastructure, made of sensors, smart kitchen appliances and user's interaction tools (interfaces), thus building a kitchen networked environment. Communication among kitchen devices exploits a wireless network, compliant with the IEEE 802.15.4-ZigBee standard. On top of such network, a supervising system and a web server are built, which enable user's application, and manage exchanging of information through the internet. The kitchen is therefore connected to external physical and digital networks (i.e., neighbourhood community, shops and to the web), enabling service aimed at increasing safety, at providing help and guidance in food preparation and at fostering exploitation of inherent social and cultural implication of feeding. End-users (which include elderly people as well as their supporting network) are involved in system and service design since its earlier phases, exploiting participatory design tools. [17]

1.4.2. NOAH Project

The NOAH system exploits an "Internet of Things" approach, with dedicated home sensors suitable for capturing expressive features of daily living activities in a non-intrusive fashion. Specific attention is paid at usability concerns, with emphasis on ease of deployment and low costs. Sensors connects straightforwardly to the home Wi-Fi network, avoiding the need of dedicated sensor networking and requiring no aggregator node. At the back end, a commercial cloud infrastructure is exploited to gather data, allowing for scalability and lowing home installation costs. On the cloud, machine learning techniques are exploited to transform raw data continuously flowing from sensors into meaningful information: trends, anomalies, alerts. The system is inherently adaptive, not involving predefined thresholds or ranges. By following a user-centric approach, specific apps are designed for the end-user and caregivers. A control dashboard is made available to care systems, to allow for integration of NOAH services into current care practices.

The NOAH project aims at developing innovative continuous monitoring techniques, based on non-intrusive home sensors and on advanced data analytics techniques.

By monitoring behavioural features in daily living activities, user-specific activity profiles are worked out. Changes (either abrupt or slowly developing), possibly related to health or wellness issues, can thus be detected in a fully automatic fashion, relieving the caregiver from the sensor-data interpretation burden. Differentiated feedback are given to the end-user, the caregiver, and the care system. [18]

The NOAH system is built on the client-server architecture, making use of the cloud technologies, and complying with the new trends in cloud application development. The system overview is represented in Figure 1 and depicts its main components.

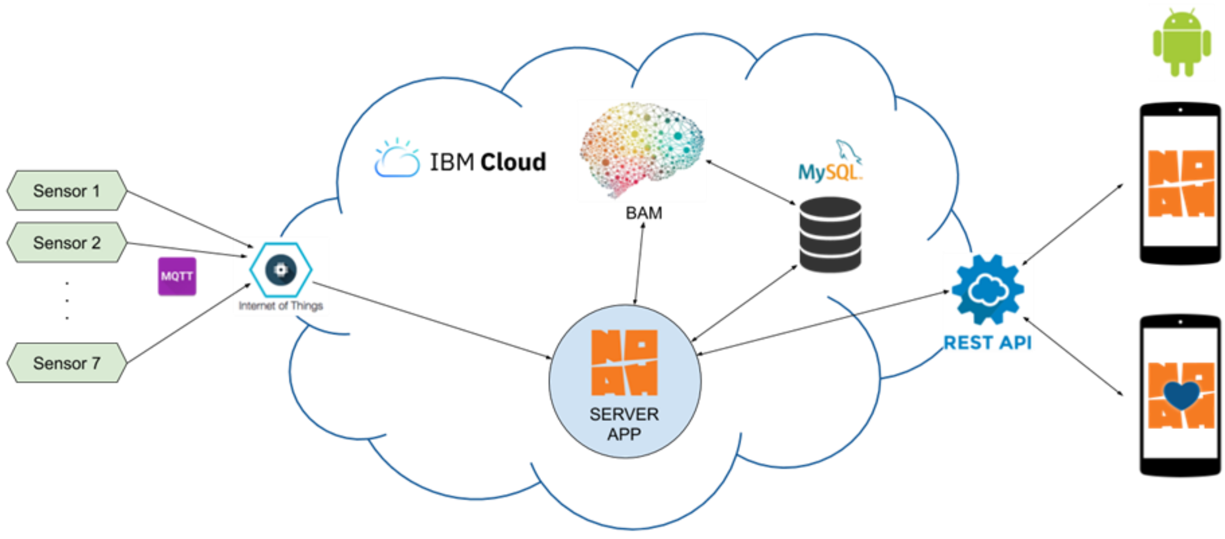


Figure 1 - NOAH System architecture overview [19]

The system's server side is developed and hosted on the *IBM Cloud* (details in [20]). This implies a continuously running cloud foundry application which uses two services: *Internet of Things Service* and *Compose for MySQL*.

The *Internet of Things Service* communicates with the registered devices (sensors) and has the role to collect data from the sensors and relay them to the server application, using the MQTT protocol, secured by SSL/TLS.

Compose for MySQL provides the persistence for the sensor data and for all the other details that the system requires to run. To improve the speed of access, the sensor data is partitioned based on groups of sensors ("kits"), and is not directly linked to the end-user, keeping their anonymity. Also, the users' details and system configuration are stored by the RDMS (MySQL).

The *Behavioural Analysis Module* (BAM) processes the sensors data to detect behaviour patterns that can indicate the well-being state of the monitored person and sends the results to the server application, which generates notifications for the system's users.

The system's client side is represented by two applications: one for the caregivers and one for the end-users. These are developed in native Android and can be used on a wide variety of devices, representing the user interfaces for the NOAH system.

The *NOAHCare* application was developed to be used by the caregivers to monitor the daily activity of the elderly persons they take care of. Through the application, they can view the state of the sensors connected in the end-user home, they receive alerts and notifications regarding sensors and changes in the behaviour of the end-user, and, also, they can view statistics on different time periods related to the collected data.

The *NOAH application* was developed for the elderly people, to receive a series of alerts (e.g. when a door is open). The elderly can register feedback about how they are feeling in a certain moment (a "mood sensor"), so this information can be linked with the interpretations given by the BAM to the elderly's dataset. Also, the end-user can set two contact persons on speed-dial.

The communication between the application's components, client, respectively the server side, is done using REST APIs, over the HTTP protocol.

The server application provides the coordination of all other components, and it is a gateway to the NOAH system's features. The cloud application is built using the *Node-RED* developing environment, which runs over a *Node.js* server. The server application is responsible with collecting data from the *Internet of Things Service*, persisting it into a relational database, generating alerts and notifications, and serving the caregiver and end-user application through a REST API.

A series of optimizations were made to the server application to ensure a high availability of the system and a low latency, by using several memory buffers and hash tables.

By running on a cloud environment, the NOAH System inherits all its advantages: dependability, accessibility, availability, and scalability.

1.4.3. HELICOPTER Project

The HELICOPTER project (*Healthy Life support through Comprehensive Tracking of individual and Environmental Behaviors*) exploits ambient-assisted living techniques to provide older adults and their informal caregivers with support, motivation, and guidance in pursuing a healthy and safe lifestyle. The project is targets 65+ adults, not suffering from major chronic diseases or severe disabilities, yet possibly being affected by (or being at risk of) metabolic or circulatory malfunctioning (e.g., hypertension, mild diabetes) or by mild cognitive deficits. Behavioural analysis is exploited to make health monitoring more effective and less invasive. [21]

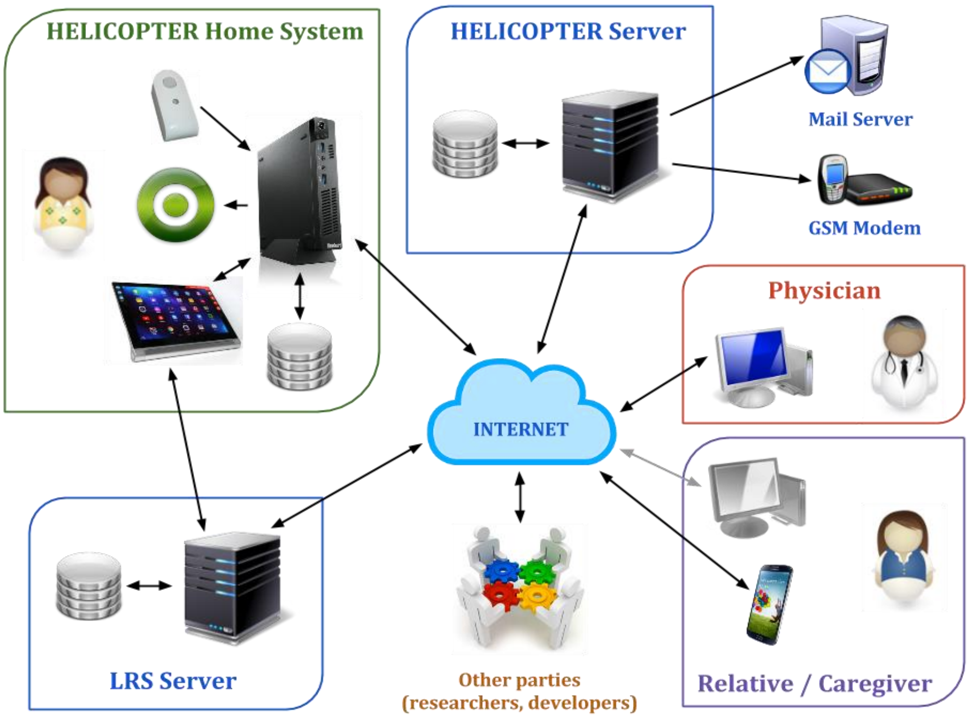


Figure 2 - Structure of the HELICOPTER system

A heterogeneous sensor network was designed and implemented, including environmental, wearable, and clinical sensors (Figure 2). A global monitoring framework was developed, in which all data converge toward a common database feeding a data analysis engine (based on a Bayesian network), capable of inferring from logged data current activity and behavioural information.

For monitoring users and their behaviour, environmental sensors and three clinical measurement devices (a scale, glucometer, and blood pressure monitor) are used, all located in their homes. Data from the sensors and measurement devices are collected by a computing system (mini-PC), also located inside the users' homes, equipped with dedicated software.

Data collection is performed in a relational database within the local system; this database is replicated at the central server level, so that data can be processed in a centralized manner. Techniques based on Bayesian probabilistic networks are used to identify aspects of user behaviour that may indicate certain conditions. The necessary calculations are carried out by the *HeliBrain* component (Figure 3), which evaluates several diagnostics daily based on an analysis of data collected from sensors.

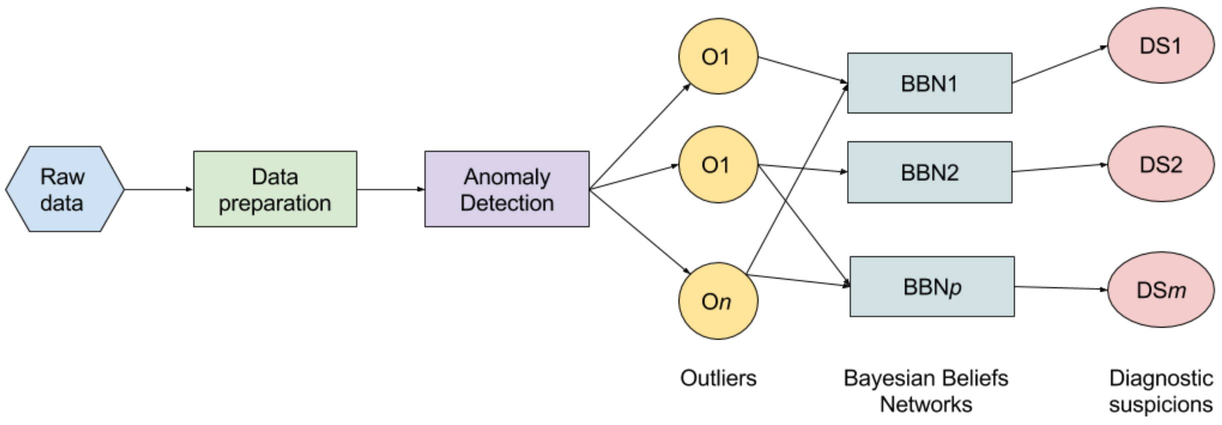


Figure 3 - Simplified diagram of the *HeliBrain* component

Data collected from sensors are filtered and processed by the *Data preparation* module to become inputs for the *Anomaly detection* module, which detects changes in the behaviour of the monitored subject. The anomalies detected are translated into truth values of hypotheses (outliers). These are used as inputs for the Bayesian causal networks, which calculate the probabilities for the diagnostics targeted by the Helicopter system.

By "poisoning" a dataset from a subject who did not present any of the diagnostics, the system's ability to calculate a correct diagnosis was tested, this being intended to validate the results obtained.

1.4.4. SAVE Project

The SAVE (*SAfety of elderly people and Vicinity Ensuring*) project is dedicated to the elderly persons suffering (or at risk) of age-related chronic illnesses and/or mild cognitive issues/disabilities. For these, and a range of "not-so-fit" persons, SAVE aimed to avoid psychosocial exclusion by "restoring the referential". The unique goal of orientation in a supportive environment – in terms of position-location and/or in terms of safety (in sensor-equipped intelligent houses) is also approached in a broader cognitive and behavioural sense, task-oriented in the "personal cloud" of relatives, friends, and caregivers. [22]

The SAVE system is a comprehensive and integrated solution dedicated to enhancing the well-being of elderly individuals by facilitating their ability to remain in their familiar environments for an extended period. The primary objective of SAVE is to ensure their safety and provide optimal care throughout their aging journey. Additionally, this innovative system aims to support informal caregivers, such as relatives, in offering the best possible care to their loved ones while effectively managing their personal and professional commitments.

One of the core features of SAVE is its capability to empower professional caregivers in creating well-structured support plans tailored to the specific needs of each elderly individual. By involving volunteering associations, the system fosters a collaborative approach to caregiving, leveraging collective efforts to maximize the quality of care provided.

The foundation of the SAVE solution lies in cutting-edge technologies, specifically designed to operate efficiently within cloud environments, particularly in the context of Infrastructure as a Service (IAAS). Utilizing the power of containerization, the system minimizes overhead, ensuring increased portability and consistent operation across diverse platforms. This not only enhances scalability and speed but also accelerates application development, allowing for timely updates and improved responsiveness.

Through its innovative approach and utilization of the latest technologies, the SAVE system addresses the evolving needs of elderly individuals, informal caregivers, and professional caregiving organizations alike. By promoting a harmonious balance between optimal care, safety, and the preservation of familiar surroundings, SAVE strives to create a compassionate and sustainable caregiving ecosystem for the elderly.

An overview of the SAVE solution is depicted synthetically in Figure 4. The sensors included in the SAVE kit (wearable and ambient) provide raw data about the elderly (end-user) (his/her well-being, activity, environment, location) by connecting through Internet to dedicated services of the SAVE cloud application. The users of the SAVE solution have dedicated user interfaces, in form of responsive web applications and mobile applications, for accessing its features:

- For the end-users: the *SAVE smartwatch face and application* and the SAVE web application.
- For the caregivers: the *SAVE web application*.
- For the SAVE solution maintenance staff and for the SAVE researchers: the S*AVE Admin Centre* web application.

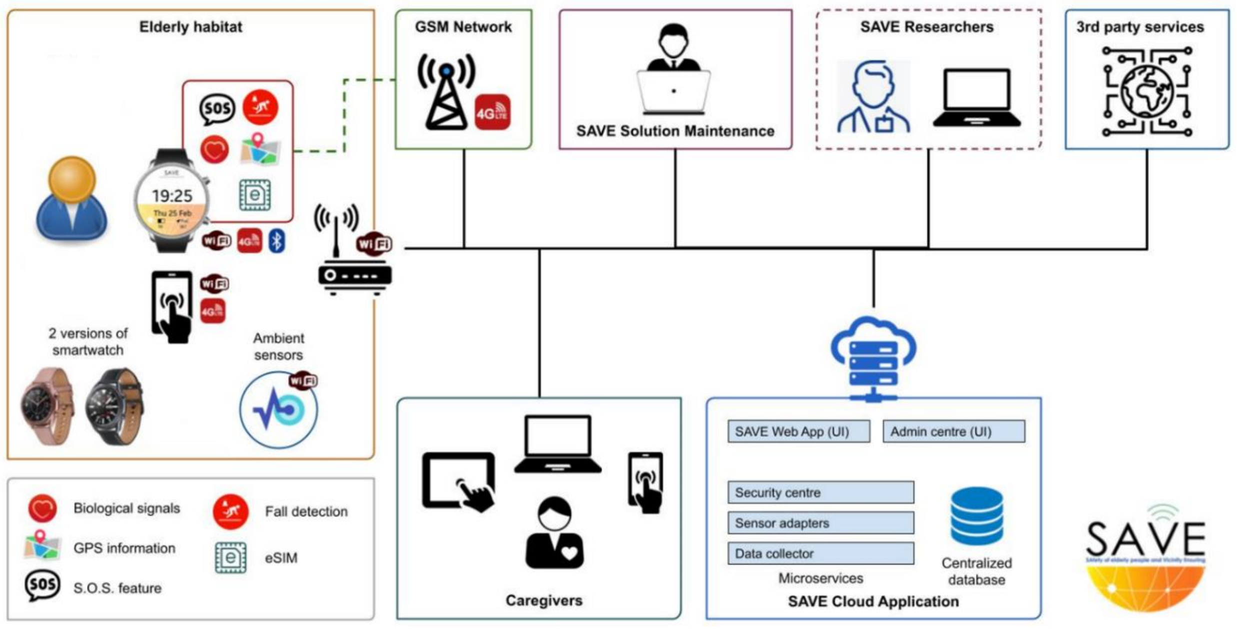


Figure 4 - SAVE solution overview

A Manager user profile has been introduced, whose role is to associate Caregiver users with primary institutionalized users. In addition, the Manager user can view all data of primary users associated with the Caregiver users they are responsible for.

All web applications are designed with responsiveness in mind, ensuring they can adapt and be accessed seamlessly from a wide array of devices, including desktops, laptops, tablets, and smartphones.

The user interfaces (the frontend) of the web applications are developed using the Angular framework and the data is retrieved using the REST APIs of the backend (developed using the Spring Boot framework).

The smartwatch face application and the smartwatch application are Tizen web application, developed in HTML, CSS and JavaScript.

2. Service-oriented architecture

Service-Oriented Architecture is a flexible set of design principles used during the stages of systems development and integration. Such architecture deploys services, which are well-defined business functionalities, accessible over a network. This allows organizations to build applications that offer services to other applications via a communication protocol over a network. The basic principles of SOA include reusability, granularity, modularity, composability, componentization, and interoperability.

SOA is not a new concept; it has evolved significantly with the adoption of the Internet and enterprise computing. SOA promotes the idea of service reuse, which can lead to reduced costs in development by minimizing redundant coding efforts. Services in an SOA environment are often loosely coupled to allow for greater flexibility in changing or replacing them with minimal impact on the consumer of those services [7].

The architecture typically involves making software components network-accessible in a standardized way. Key components of SOA include:

- **Service Provider**: Creates and provides access to services.
- **Service Consumer**: Uses or consumes services provided by the service provider.
- **Service Registry**: A directory where services are listed and described, and through which they can be discovered by potential users.
- **Service Contract**: Defines the interface between the provider and consumer, specifying the service's functionalities and the terms of use.

A variety of technologies support the deployment of SOA:

- **Web Services**: These are the predominant way of implementing SOA, using standards such as SOAP, WSDL, and REST.
- **Middleware**: Technologies like *Enterprise Service Bus* facilitate the communication and management of data between services.
- **XML and JSON**: These are used for data interchange between services.

The adoption of SOA can lead to enhanced agility, where businesses can respond more quickly to changing requirements. SOA also supports integration across different platforms and languages, which facilitates legacy system integration and new service deployment [23].

While SOA offers numerous benefits, there are challenges in its implementation:

- **Complexity**: Managing a large number of services and their interactions can become complex.
- **Security**: Ensuring secure service interactions is critical, especially as services can be exposed over public networks.
- **Governance**: Effective governance practices are necessary to ensure that services adhere to organizational standards and policies.

2.1. Web services

Web services, within the context of Service-Oriented Architecture, play a pivotal role in enabling modular and interoperable software applications across diverse platforms and networks. As distinct components of SOA, web services allow different applications to communicate with each other without requiring user intervention or changes to the underlying software infrastructure. This capability is fundamental to achieving the flexibility and scalability that SOA aims to provide.

Web services in SOA are built around key standards that ensure interoperability and accessibility, like:

- **SOAP**: A protocol that defines a uniform way of passing XML-encoded data.
- **WSDL**: An XML-based language used to describe the services a business offers and to provide a way for individuals and other businesses to access those services electronically.
- **UDDI**: A platform-independent framework for describing services, discovering businesses, and integrating business services using the Internet.
- **REST**: An architectural style that uses existing protocols, typically HTTP, and treats data and functionality as resources that can be read or manipulated using a stateless set of operations.

The implementation of web services in SOA frameworks provides several benefits:

- **Interoperability**: By adhering to standard protocols, web services ensure that different systems can work together, regardless of the platform or programming language used [24].
- **Reusability**: Services can be designed for reuse across multiple applications, reducing the need for duplication of effort.
- **Scalability**: Services can be scaled up or down based on demand without affecting the end user or requiring major changes to the architecture.
- **Flexibility**: Organizations can add or modify services without disrupting existing systems, allowing for rapid adaptation to changing business needs.

Web services are used in a variety of applications, from simple information requests to complex business processes. They are particularly valuable in integrating disparate systems within and across organizational boundaries. However, the adoption of web services also presents challenges:

- **Security**: Ensuring data security and safe transactions over the internet is crucial and complex.
- **Performance**: Handling high volumes of web service requests can lead to performance bottlenecks.
- **Complexity in management**: As the number of web services within an organization grows, managing them becomes more complex.

2.2. FOOD system architecture

The *Service Oriented Architecture* is an approach to organizing IT resources in which data, logic and infrastructure resources are accessed by routing messages between networked devices.

Basic values of SOA in the FOOD project [25]:

- **Reusability of services**: developed services can be combined in countless ways to obtain new facilities (achievable using different user interfaces - UIs).
- **Distributed architecture**: services can be located anywhere.
- **Openness**:
  - new services can be built using already developed facilities through standardized interfaces.
  - use of open standards and protocols (XML, SOAP etc.).
- **Incremental development / implementation**: at any point in time, we can have a functional version of a service.

The FOOD system is built according to the principles of service-oriented architecture (SOA). Figure 5 below contains an overview of the architecture elements.

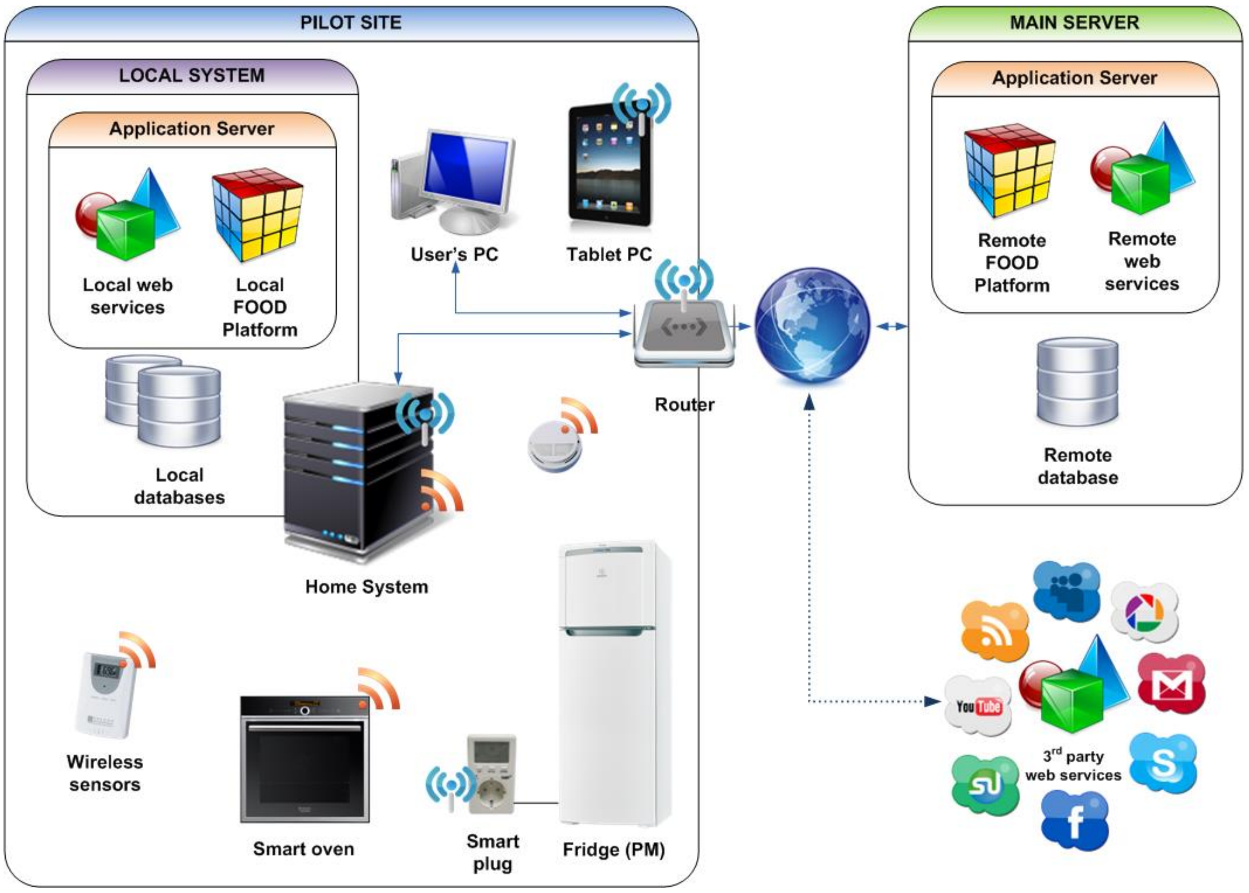


Figure 5 - FOOD system architecture [25]

The technology stack on the server-side employed to implement the services are:

- Operating system: *Ubuntu 11.10 64b*
- Runtime environment: *Oracle Java 1.6*

- Database management system: *MySQL 5.1*
- Application server: *Glassfish 3.1*
- Webservice technologies: *JAX-WS*
- *XML* for formalizing data

The software components are located on the *Local system* and the *Main server*. Both systems run an application server (*Glassfish*) that offers webservices.

From the point of view of the communication resources involved in providing services for the system's users, there are:

- *Local services*: are accessible only inside the *Home system*; they don't need outside communication (e.g. temperature readings).
- *Remote/External services*: are served by an outside system; are accessed directly from the Internet, without the intervention of the Home system.
- *Mixed services*: are provided by the Home system, but they require additional data from the Main server (e.g. internal messaging, updates).

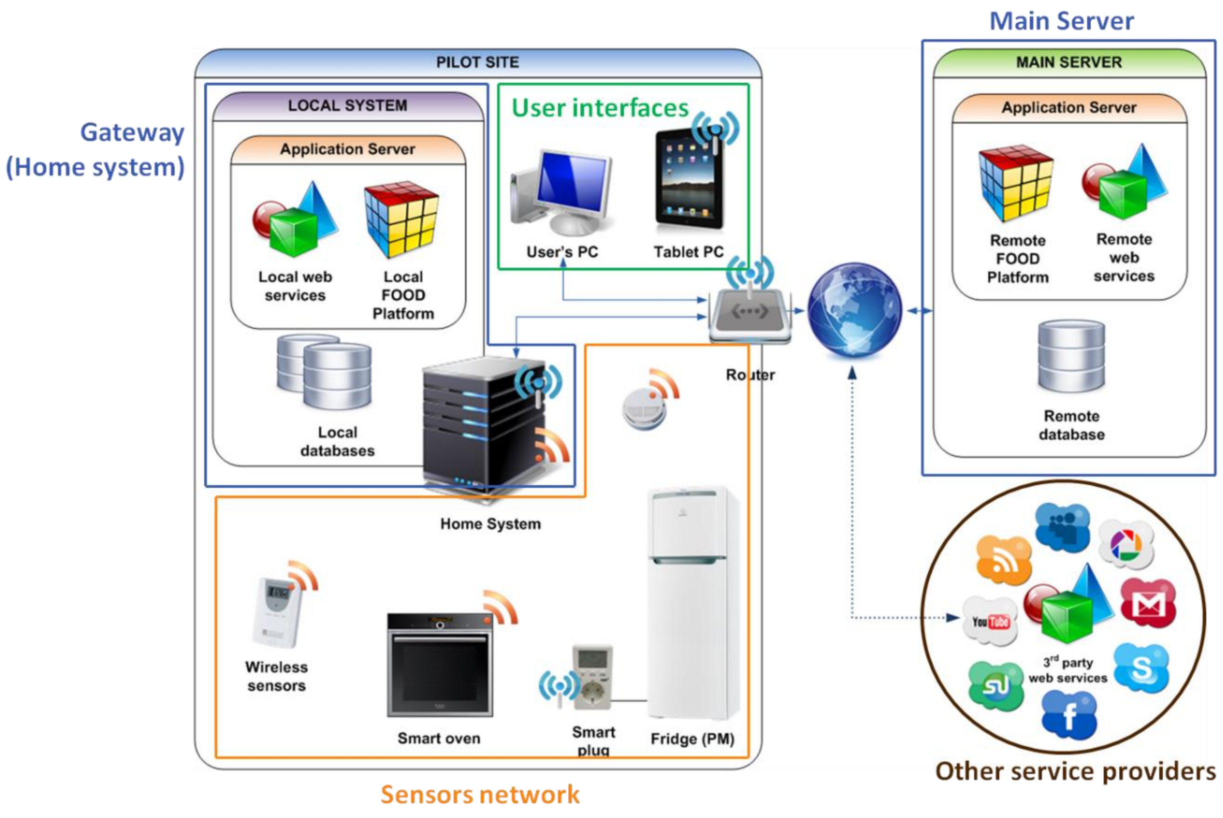


Figure 6 - FOOD system zones

There can be identified 5 zones on the general architecture diagram (Figure 6):

- *Sensor network*: comprised in all the wireless sensors (including appliances).
- *Gateway*: a heterogeneous software component that runs on the home system machine; it contains the databases, the application server and the local webservices.
- *User interfaces*: contains the software used for human-machine communication.
- *Main server*: offers external and mixed services.
- *Other service providers*: offers external services (e.g. *Skype*).

Webservices are one possible way of realizing the technical aspects of SOA. The FOOD services are implemented as web services. The technology used is JAX-WS; the web services reside in a *Glassfish* server.

The communication between client (e.g. web application) and webservice is encoded as SOAP messages. The transfer of SOAP messages is achieved through the HTTP protocol.

2.2.1. Data flow and web services in the FOOD system

Figure 7 shows how the data from sensors are collected and employed by the FOOD system.

The elements of this architecture fulfil the following [25]:

- *Sensor Controller* collects the data from the smart devices using over-the-air technologies and protocols.
- *Current Status table* maintains the current status (the state) of the sensors.
- *Commands table* keeps temporarily the commands to be executed; when a command is executed, it is eliminated from the list.
- *Log values* table stores the historical data, needed for statistical analysis.
- *Smart objects table* and *Smart objects properties table* stores the information about the smart devices.
- *Measure types table* keeps the information about the measurement units used by the properties of the devices (e.g. temperature - degC).
- *Measure data table* stores the historical data.
- *Smart object controller adapter* copies all the data from the current status into the historical data table and stores in the Java Directory (inside the volatile memory) the current status: reduces the load of the database and reduces access time to the current status.
- *DataWS* (*SensorsWS*) offers a communication interface to retrieve data from the current status.
- *HistoryWS* offers a communication interface to retrieve historical data under a graphical form.

- *CommandsWS* offers a communication interface that allows sending of commands to smart devices.

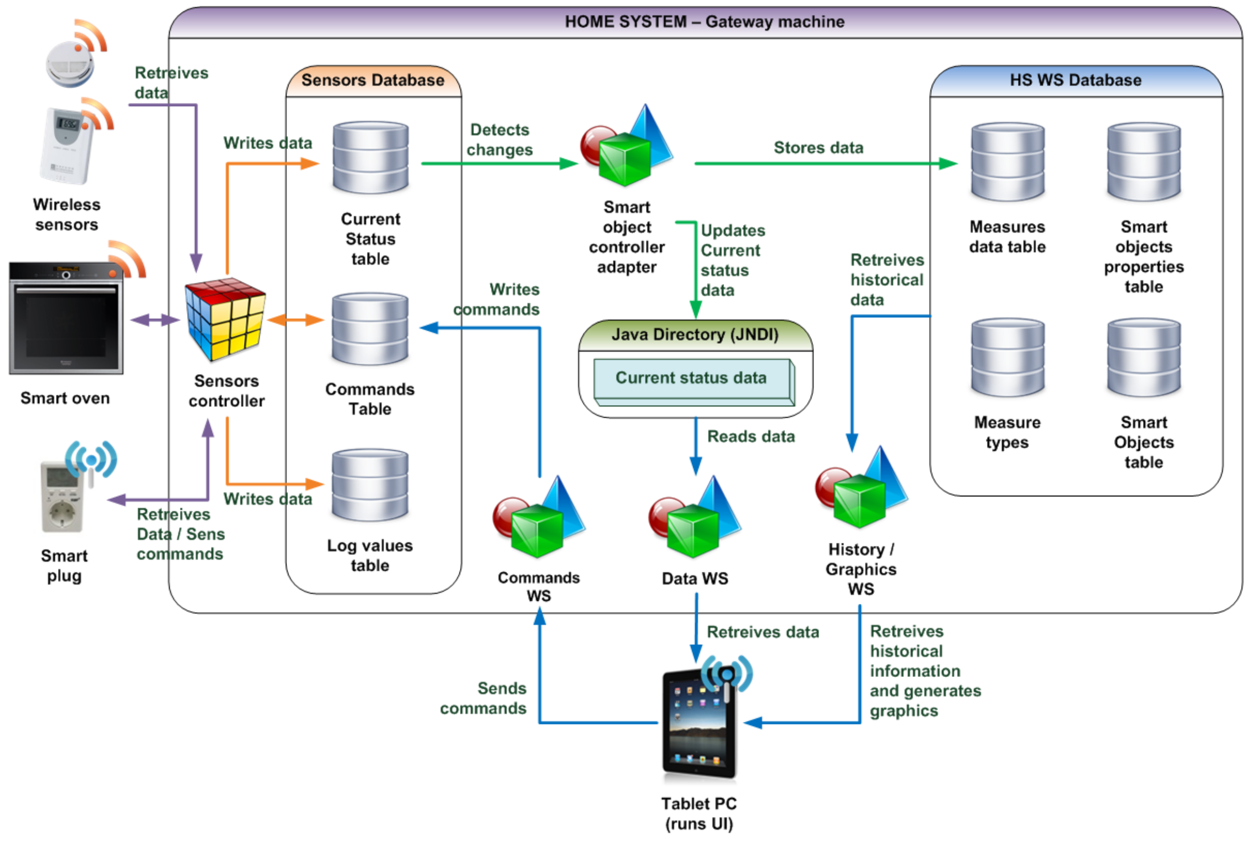


Figure 7 - Information flow in the Home system [25]

At the lowest level, the *Sensors controller* developed by Parma University (partner in the project) communicates directly with the sensors using various wireless protocols. The data is stored in the tables previously described.

The *Smart object controller adapter* acts as a middleware between the sensor network and the *Data/Sensors webservice* and the webservices' database. This is developed as a module inside the *Glassfish* application server. The main purpose of this adapter is to detect changes in the database of sensors and to write them inside the webservices' database. Also, for improved performance of the system, the current status of sensors is stored in the JNDI space, so the requests for it are resolved using the data found in memory and not querying the MySQL database.

At the highest level, the webservices provide an abstract interface (based on XML) for the user applications, which can be developed using a large variety of technologies. There are 3 main webservices for working with the sensors' data, each of them providing a specialized service.

Because webservices only offer abstract interfaces, they are not suitable for direct use. To offer services to the end-user the system must have define a user interface that will interact with the webservices and present the result in a user-friendly, meaningful way.

SensorsWS

This webservice provides operation for working with real-time data coming from different sensors. The data is stored using the setData operation and retrieve data with getData or getAllDataForDevice operations.

The web service is implemented using JAX-WS technology and resides in the Glassfish application server.

Main operations are:

Name:	**getData**
Purpose:	Retrieves the current value of a specific property of a device.
Parameters:	String deviceId: a unique code that identifies each device String devicePropertyId: a unique code that identifies each device property

Request example:	`<?xml version="1.0" encoding="UTF-8"?>`
	`<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">`
	`    <S:Header/>`
	`    <S:Body>`
	`        <ns2:getData`
	`xmlns:ns2="http://webservice.sensors.ws.vision.ro/">`
	`            <deviceId>1000</deviceId>`
	`            <devicePropertyId>1000</devicePropertyId>`
	`        </ns2:getData>`
	`    </S:Body>`
	`</S:Envelope>`

Results example:	`<?xml version="1.0" encoding="UTF-8"?>`
	`<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">`
	`    <S:Body>`
	`        <ns2:getDataResponse`
	`xmlns:ns2="http://webservice.sensors.ws.vision.ro/">`
	`            <return>`
	`                <dataValue/>`
	`                <deviceId/>`
	`                <devicePropertyId/>`
	`                <id>0</id>`
	`                <measurementId/>`
	`                <tstamp/>`
	`            </return>`
	`            <return>`
	`                <dataValue>35</dataValue>`
	`                <deviceId>1000</deviceId>`
	`                <devicePropertyId>1000</devicePropertyId>`

	`<id>`**0**`</id>`
	`<measurementId>`**2013/02/14 8:14:37.000000**
	`</measurementId>`
	`<tstamp>`**2013/02/14 08:14:37.000000**`</tstamp>`
	`</return>`
	`</ns2:getDataResponse>`
	`</S:Body>`
	`</S:Envelope>`

	First *return* node contains the result of the operation; on success the *id* node is 0; on error, this value is -1. The following nodes, if they exists, contain the requested data.

Name:	**setData**
Purpose:	Stores data in the database and in the current status object in the JNDI. Typically is called by the smart object adapter.
Parameters:	String deviceId: a unique code that identifies each device String devicePropertyId: a unique code that identifies each device property String tstamp: (timestamp) date and time of day String dataValue: the value of the property String measurementId: a unique code that identifies each type of measurement

Request example:	`<?xml version="1.0" encoding="UTF-8"?>`
	`<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">`
	`<S:Header/>`
	`<S:Body>`
	`<ns2:setData`
	`xmlns:ns2="http://webservice.sensors.ws.vision.ro/">`
	`<deviceId>`**1005**`</deviceId>`
	`<devicePropertyId>`**1000**`</devicePropertyId>`
	`<tstamp>`**2013-04-23 16:55:10**`</tstamp>`
	`<dataValue>`**25**`</dataValue>`
	`<measurementId>`**2013-04-23 16:55:10** `</measurementId>`
	`</ns2:setData>`
	`</S:Body>`
	`</S:Envelope>`

Results example:	`<?xml version="1.0" encoding="UTF-8"?>`
	`<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">`
	`<S:Body>`
	`<ns2:setDataResponse`
	`xmlns:ns2="http://webservice.sensors.ws.vision.ro/">`
	`<return>`**1234**`</return>`
	`</ns2:setDataResponse>`
	`</S:Body>`
	`</S:Envelope>`

	The *return* node contains the result of the operation; on success the node contains the unique id of the row in the database (the value of the primary key field); on error, this value is -1.

Name:	**getAllDataForDevice**
Purpose:	Retrieves the current values of a all properties of a device.
Parameters:	String deviceId: a unique code that identifies each device

Request example:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:getAllDataForDevice
xmlns:ns2="http://webservice.sensors.ws.vision.ro/">
            <deviceId>1000</deviceId>
        </ns2:getAllDataForDevice>
    </S:Body>
</S:Envelope>
```

Results example:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:getAllDataForDeviceResponse
xmlns:ns2="http://webservice.sensors.ws.vision.ro/">
            <return>
                <dataValue/>
                <deviceId/>
                <devicePropertyId/>
                <id>0</id>
                <measurementId/>
                <tstamp/>
            </return>
            <return>
                <dataValue>1</dataValue>
                <deviceId>1000</deviceId>
                <devicePropertyId>0</devicePropertyId>
                <id>0</id>
                <measurementId>2013/02/20              13:46:13.000000
</measurementId>
                <tstamp>2013/02/20 13:46:13.000000</tstamp>
            </return>
            <return>
                <dataValue>35</dataValue>
                <deviceId>1000</deviceId>
                <devicePropertyId>1000</devicePropertyId>
                <id>0</id>
                <measurementId>2013/02/14              08:14:37.000000
</measurementId>
                <tstamp>2013/02/14 08:14:37.000000</tstamp>
            </return>
            <return>
                <dataValue>1</dataValue>
                <deviceId>1000</deviceId>
                <devicePropertyId>1001</devicePropertyId>
                <id>0</id>
                <measurementId>2013/02/14              08:14:37.000000
</measurementId>
                <tstamp>2013/02/14 08:14:37.000000</tstamp>
            </return>
        </ns2:getAllDataForDeviceResponse>
    </S:Body>
```

	</S:Envelope>
	First *return* node contains the result of the operation; on success the *id* node is 0; on error, this value is -1. The following nodes, if they exists, contain the requested data.

HistoryWS

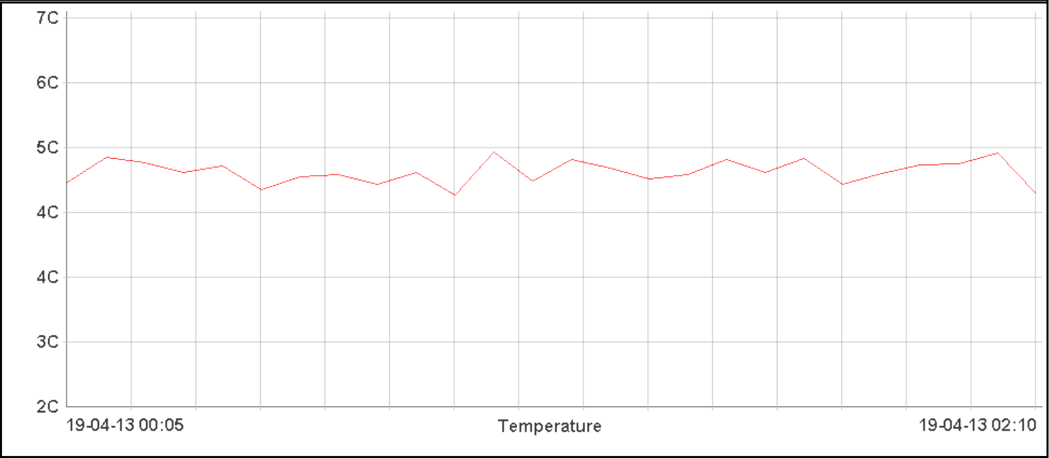
This web service provides a history of data read from the sensors in the form of a graph. A special feature of this chart is that autoscale itself according to the number of values read and according to their value.

The web service is implemented using JAX-WS technology and resides in the Glassfish application server.

Main operations are:

Name:	**getHistoryMeasuresData**
Purpose:	
Parameters:	String deviceId: a unique code that identifies each device String devicePropertyId: a unique code that identifies each device property String timeStart: begin date of the interval for measurements String timeStop: end date of the interval for measurements String yUnit: unit of measurement on the y axis String title: title of measurement

Request example:	<?xml version="1.0" encoding="UTF-8"?>
	<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
	    <S:Header/>
	    <S:Body>
	        <ns2:getHistoryMeasuresData xmlns:ns2="http://webservice.history.ws.vision.ro/">
	            <deviceId>**1005**</deviceId>
	            <devicePropertyId>**1000**</devicePropertyId>
	            <timeStart>**2013-04-19 00:05:00**</timeStart>
	            <timeStop>**2013-04-19 02:10:00**</timeStop>
	            <yUnit>**C**</yUnit>
	            <title>**Temperature**</title>
	        </ns2:getHistoryMeasuresData>
	    </S:Body>
	</S:Envelope>

Result example:	
	The result is encoded as a Base64 image resource that can be displayed in HTML using an *img* element (tag).

CommandsWS

Send commands to the devices control subsystem. If the return value is greater than 0 that means the command was successfully sent.

The web service is implemented using JAX-WS technology and resides in the Glassfish application server.

Main operations are:

Name:	setCommand
Purpose:	
Parameters:	int objId: a unique code that identifies each device int varId: a unique code that identifies each monitored device parameter short flag: specifies the type of the value (integer, float, string) String value: the values sent to the command

Request example:	<?xml version="1.0" encoding="UTF-8"?>
	<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
	    <S:Header/>
	    <S:Body>
	        <ns2:setCommand xmlns:ns2="http://webservice.commands.ws.vision.ro/">
	            <objId>**1001**</objId>
	            <varId>**10000**</varId>
	            <flag>**1**</flag>
	            <value>**25**</value>
	        </ns2:setCommand>
	    </S:Body>
	</S:Envelope>

Result example:	`<?xml version="1.0" encoding="UTF-8"?>`
	`<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">`
	`    <S:Body>`
	`        <ns2:setCommandResponse`
	`xmlns:ns2="http://webservice.commands.ws.vision.ro/">`
	`            <return>`**`14`**`</return>`
	`        </ns2:setCommandResponse>`
	`    </S:Body>`
	`</S:Envelope>`
	Initial value of the "*retVal*" is 0;
	If the returned value is wrong then "*retVal*" is -1;

CookingCyclesWS

The smart oven allows the user to upload new cooking cycles. This webservice facilitates this operation. The cooking cycles are described by the sensors subsystem through a XML file. The webservice offers interfaces for fetching the current cooking cycles and creating new ones. The communication between the upper and lower level of the system is made through the XML files.

The web service is implemented using JAX-WS technology and resides in the Glassfish application server.

Main operations are:

Name:	**getAllCookingCycles**
Purpose:	Returns all the cooking cycles in a settings XML file (extension ".eep")
Parameters:	String inputStringVal: the name of the current CookingCycle file
Request example:	`<?xml version="1.0" encoding="UTF-8"?><S:Envelope`
	`xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">`
	`    <S:Header/>`
	`    <S:Body>`
	`        <ns2:getAllCookingCycles`
	`xmlns:ns2="http://webservice.cookingcycles.ws.vision.ro/">`
	`            <settingFile>`**`Father.eep`**`</settingFile>`
	`        </ns2:getAllCookingCycles>`
	`    </S:Body>`
	`</S:Envelope>`
Result example:	`<?xml version="1.0" encoding="UTF-8"?><S:Envelope`
	`xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">`
	`    <S:Body>`
	`        <ns2:getAllCookingCyclesResponse`
	`xmlns:ns2="http://webservice.cookingcycles.ws.vision.ro/">`
	`            <return>`
	`<File>`
	`<Name>Father.eep</Name>`
	`<Cycles>`

	<Cycle><SelectorLabel>Multilevel</SelectorLabel> **<SelectorId>1</SelectorId></Cycle>** **<Cycle><SelectorLabel>Barbecue</SelectorLabel>** **<SelectorId>2</SelectorId></Cycle>** **<Cycle><SelectorLabel>Gratin</SelectorLabel>** **<SelectorId>3</SelectorId></Cycle>** **<Cycle><SelectorLabel>Rotisserie</SelectorLabel>** **<SelectorId>4</SelectorId></Cycle>** **<Cycle><SelectorLabel>Fish</SelectorLabel>** **<SelectorId>5</SelectorId></Cycle>** **<Cycle><SelectorLabel>Pizza</SelectorLabel>** **<SelectorId>6</SelectorId></Cycle>** **</Cycles>** **</File>**</return> </ns2:getAllCookingCyclesResponse> </S:Body> </S:Envelope>
	The node *name* contains the name of the XML file on the harddrive.

3. Microservices architecture

Microservices serve as a strategic alternative to monolithic application development. In contrast to monolithic applications, where all components are tightly interconnected, microservices offer greater flexibility and ease of maintenance. By breaking down the application into smaller, independent units, modifications and updates can be implemented more efficiently. Additionally, the microservices architecture facilitates migration to cloud environments, enabling better scalability and management (Figure 8).

Advantages of using microservices for applications:

- **Implementation** – there is more agility in delivering new versions of services due to shorter times required for building and testing processes, and consequently shorter application delivery times.
- **Availability** – most often, delivering a new version of an application that uses the monolithic system requires restarting the entire system, whereas a microservices-based system requires very little downtime.
- **Efficient management** – because microservices act as small portions of a whole, software developers on the team will have the ability to work independently and with higher productivity, with the team also being divided into smaller parts.
- **Capability for modification and adaptation** – greater flexibility in using different frameworks, data sources, and libraries will allow the application to be adapted as it develops.
- **Reliability** – in the event of a fault, only a single module is affected. Conversely, faults in a monolithic system can cause the entire system to stop.

- **Scalability** – microservices allow the independent scaling of each component.

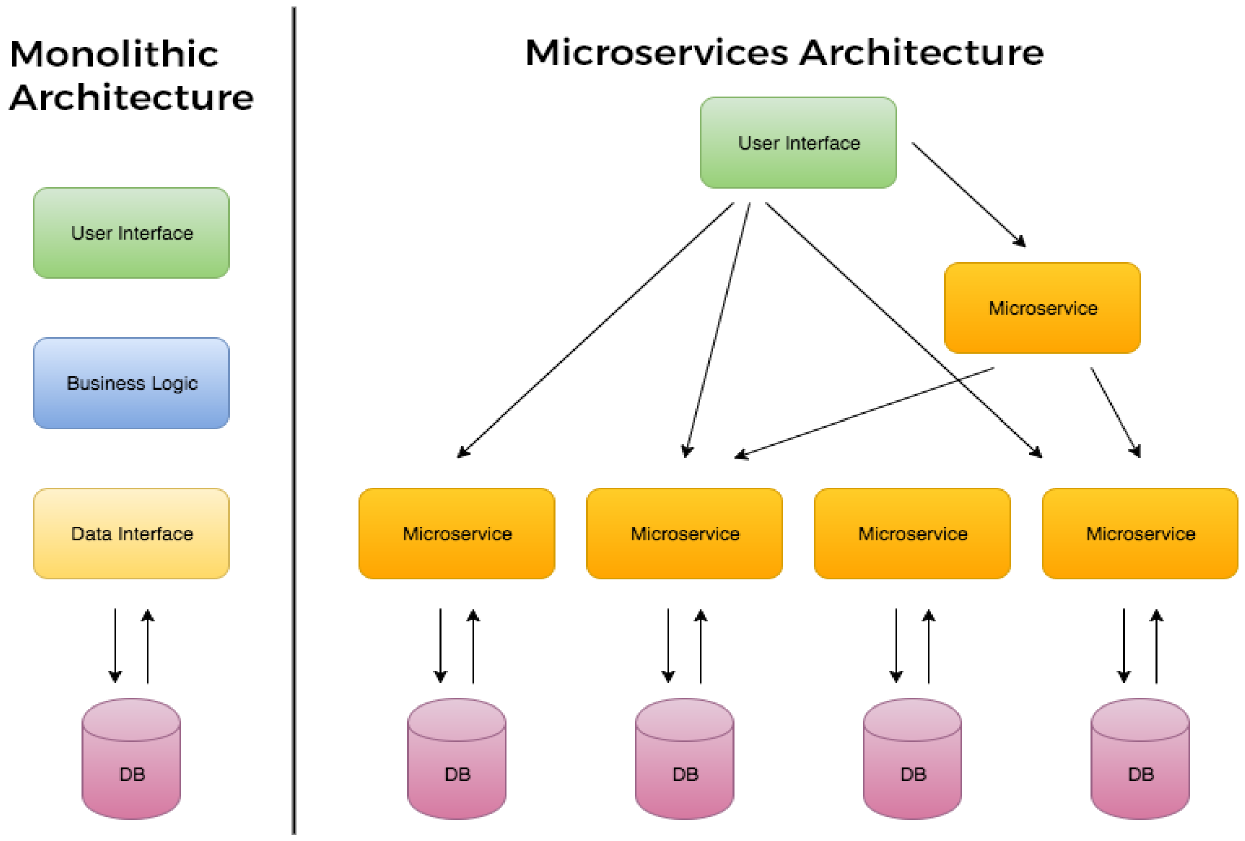


Figure 8 - Monolithic architectures vs. microservices [26]

3.1. SAVE System architecture

From an architectural standpoint, the SAVE solution adopts a service-based architecture, employing microservices as independent and smaller software components that fulfil specific tasks. These microservices communicate with each other through a standardized interface, using HTTP or HTTPS protocols for message-based interactions. Typically, JSON format is used for the messages exchanged between services, providing flexibility and interoperability.

The adoption of a microservice-based architecture for the SAVE system offers a multitude of technical and implementation advantages, streamlining application development in a cloud environment. This approach considers crucial factors such as implementation efficiency, availability, effective management, adaptability, reliability, and scalability.

The microservices approach also facilitates rapid development of new features and functionalities. By breaking down the application into smaller, self-contained units, each microservice can be developed and updated independently. This accelerates the development process, as teams can focus on specific functionalities without disrupting other parts of the system. Consequently, new facilities can be rolled out more quickly, enhancing the system's capabilities and responsiveness to user needs.

One of the primary benefits of employing a microservices-based architecture in the SAVE system is its compatibility with cloud infrastructure. This allows the system to leverage the scalability and flexibility offered by cloud environments, enabling efficient resource allocation and distribution of workloads. As a result, the system can seamlessly accommodate growing demands and handle increased user traffic, ensuring optimal performance.

The SAVE system's architecture is designed to harness the power of cloud hosting, allowing various components to be hosted in the cloud. Each of these components is dedicated to specific tasks and functionalities, ensuring a streamlined and efficient system. Moreover, the microservices-based structure permits seamless integration with external systems provided by third parties. These external systems may include sensors, mobile devices, or web applications that contribute essential data to the SAVE system. The ability to interact with and process data from these external sources enhances the system's overall functionality and empowers it to deliver comprehensive monitoring and analysis capabilities.

The general architecture of the SAVE system (as depicted in Figure 9) is structured around several independent components, which collaborate and communicate with each other to fulfil specific functionalities.

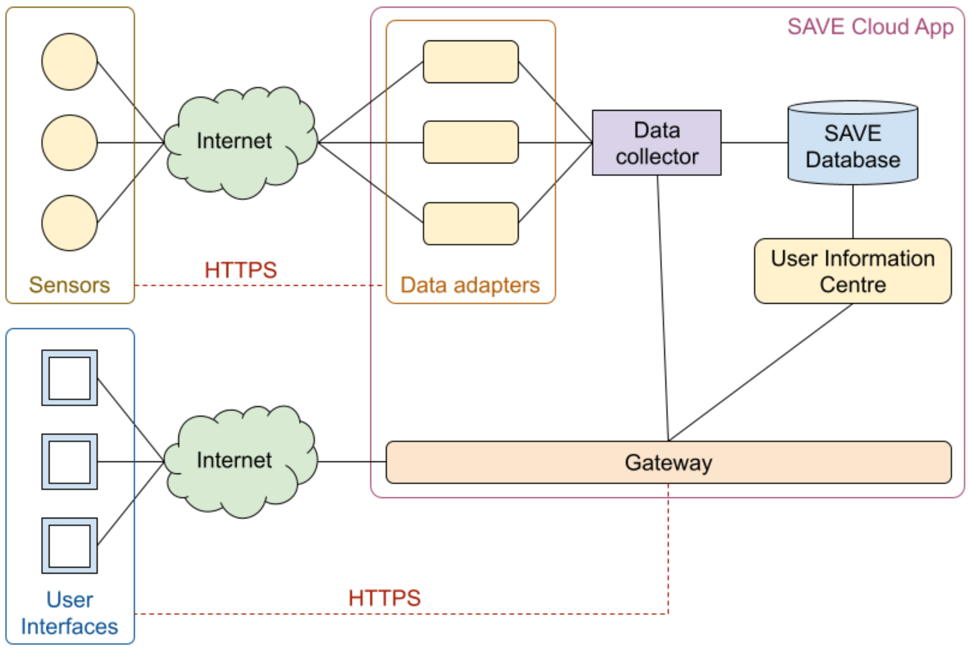


Figure 9 - General architecture of the SAVE system

The SAVE cloud application employs a modular approach, dividing its functionalities into distinct functional units (microservices). Each of these microservices is responsible for handling specific aspects of the business logic:

- **Data Collector**: This microservice gathers data from various sensors and efficiently stores it in the database. It also provides a communication interface through a REST API, enabling easy retrieval of the stored data.
- **Sensor Adapters**: These arrays of microservices establish direct connections with sensing devices and relay the collected data to the Data Collector in a standardized format, ensuring smooth and consistent data exchange.
- **User Information Centre**: This microservice is dedicated to handling authentication and authorization operations, offering REST APIs for seamless access control.
- **SAVE Web App**: Serving as the web-based user interface, this microservice caters to end-users and caregivers, providing a user-friendly experience for managing and monitoring the SAVE system.
- **SAVE Admin Centre**: As another web-based user interface, this microservice is designed for maintenance staff and researchers, facilitating administrative tasks and research-related activities.

For data storage, the SAVE solution utilizes a relational database management system, specifically Oracle's *MySQL*. To ensure rapid access to the data, a partitioning algorithm has been implemented at the cloud application level, leveraging the kit identifier to optimize data retrieval.

The SAVE system is intentionally designed to be inclusive, enabling the incorporation of other sensor kits developed by third-party providers, as long as they don't require permanent maintenance and configuration. Successful piloting has been conducted with the *Xiaomi Aqara Smart Home Set*, demonstrating compatibility and integration with good results. Also, the SAVE solution included a smartwatch as a wearable sensor for monitoring the activity and location of the end-users; a smartphone application connects to the cloud app and feeds the database. Additionally, the *Data Collector* service offers an inclusive REST API, facilitating easy connection with any other sensor systems to be integrated into the SAVE solution. Such an example is the link with the *Technological Club* device (eHealth and CRT – Choice Reaction Time).

Figure 10 shows the communication connections between microservices, sensors and user interfaces for the SAVE solutions.

The implementation of these components was done using the *Java* programming language and the *Spring Boot* framework. This combination offers numerous advantages for developing microservices applications, ensuring their smooth operation and straightforward maintenance over time.

Additionally, the Spring Boot framework simplifies the process of updating individual microservices independently without causing any disruptions to other components, enhancing overall system flexibility and efficiency.

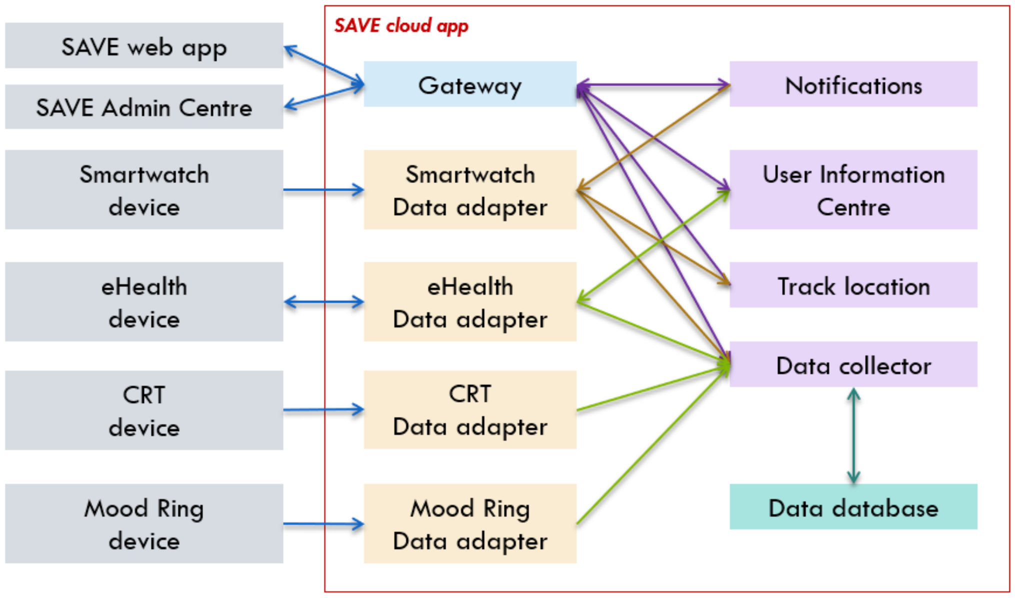


Figure 10 - Communication connections between microservices

For intercommunication between microservices, REST communication interfaces are utilized, employing the HTTP or HTTPS protocols. Standardizing the messages exchanged between microservices using the JSON format ensures seamless and straightforward communication, both among internal components and with external systems. This setup simplifies the integration of various services and external devices into the SAVE system, promoting easy data exchange and interaction.

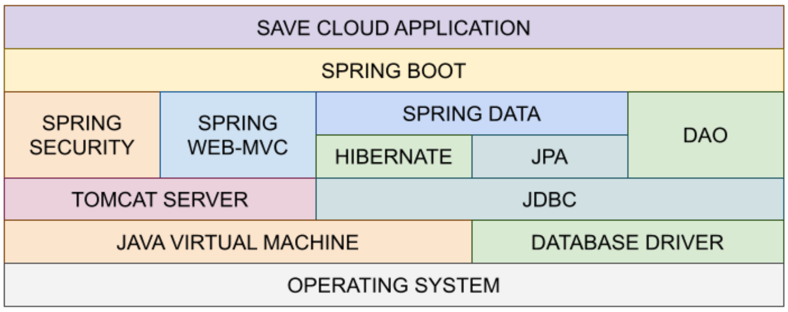


Figure 11 - SAVE microservices technology stack [27]

The simplified technology stack for the microservices included in the SAVE cloud application is depicted in Figure 11.

3.1.1. The SAVE Data collecting system – the Data collector microservice

One of the most important components of the SAVE system is the data collection service.

This service is implemented as a microservice, to be easily scalable with the increase of number of sensors.

The communication interface is a REST API (over HTTPS), that accepts JSON formatted data coming from devices that are registered inside SAVE's database.

The minimal structure of the JSON message accepted by the data collecting API is:

```
{
"kdDevId": <integer_value>,
"kdValue": <string_value>
}
```

When the sensors cannot provide the data in the accepted format, a data adapter must be implemented, so it will wrap the sensor's native format (binary, text, JSON, XML, etc) into the accepted structure. The eHealth, Mood rings and CRT devices have a data adapter running inside the SAVE cloud application that intermediates between the devices and the *Data collecting system* (*Data collector*).

As illustrated in Figure 12, the devices connect via the HTTPS protocol to a device type-specific adapter interface that transmits the data to the collection system, which will persist it in the database.

The data adapter interface is a software module that is developed specifically for each type of device recognized by the system. Its role is to expose a communication interface with external devices and to convert the format of the received data into a standard format recognized by the data collection component and to send it to the latter.

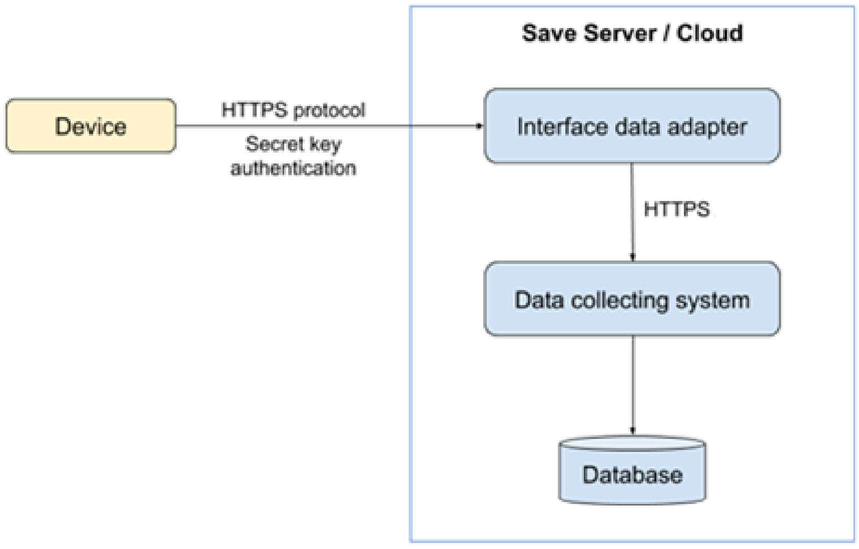


Figure 12 - SAVE data collecting system - communication model [27]

The adapter streamlines communication between devices and system, abstracting the data format and providing it to be persisted. In this sense, the adapter is specific to the type of device with which it interacts and can be developed using any technology; it must, however, maintain the standard of communicating with the central system.

The adapter streamlines communication between devices and system, abstracting the data format and providing it to be persisted. In this sense, the adapter is specific to the type of device with which it interacts and can be developed using any technology; it must, however, maintain the standard of communicating with the central system.

The communication between the adapter and the *Data collector* is done via the HTTPS protocol, doubled by an authentication based on a secret key (API Key) [27].

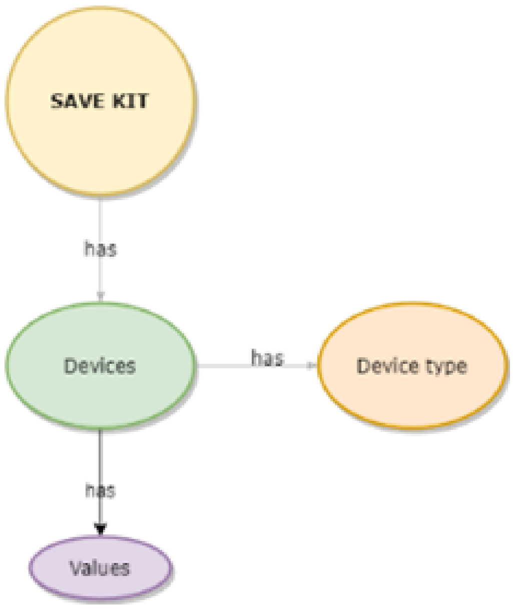


Figure 13 - Data organization in the SAVE system [27]

The adapters are implemented as microservices, using the Java language and the *Spring Boot* framework.

Once in the system and transposed into a standard format, data from the devices are organized (Figure 13) and stored.

A device must be registered in advance; it is associated with a device type (which defines the corresponding adaptation interface). This device will receive a name so that users can easily identify it.

The collected data are stored in tables in the database. To optimize the speed of data access, kits partition them, so that the data from a kit is stored in the same table. The data storage mode is a general one, allowing the saving of any data structure.

3.1.2. Deployment environment

The SAVE cloud application, designed on the microservices architecture and built using the *Spring Boot* framework, can be deployed on a vast number of infrastructures. The microservices can run independently of a servlet engine/web server (e.g. Tomcat) or can be deployed as a classic Java web application. Also, the microservices can be deployed using the *Docker* containerization engine, *Kubernetes* being employed for the orchestration [27].

4. Internet of Things and its implications for AAL

The *Internet of Things* (IoT) represents a paradigm in which the internet extends into the physical world, encompassing a wide array of devices and objects that communicate and interact with each other. In the context of Ambient Assisted Living, IoT technologies offer promising opportunities to enhance the quality of life for the elderly and individuals with disabilities by providing intelligent support systems within home environments.

IoT in AAL relies on several key components: sensors, actuators, communication networks, and data analytics platforms. Sensors collect data about the environment or the user (e.g., motion sensors, wearables monitoring vital signs), while actuators perform actions based on processed data (e.g., automated lighting, emergency alerts). These components are interconnected through a communication network that supports data transfer and integration [28].

The architecture of IoT in AAL systems must support interoperability, scalability, and security. Common frameworks include cloud-based models, where data is processed and stored on remote servers, and edge computing models, where data processing occurs closer to the source of data [29]. These frameworks help manage the vast amount of data generated by IoT devices, ensuring timely responses and personalized interactions.

The main fields of applications of IoT in AAL are:

- **Health monitoring**: IoT devices can continuously monitor health parameters such as heart rate, blood pressure, and glucose levels, providing real-time feedback to both users and healthcare providers. This enables proactive management of health conditions, potentially reducing hospital visits and improving medical outcomes [30].
- **Safety and security**: IoT systems enhance the safety of the elderly by detecting anomalies such as falls or unusual inactivity, and by providing systems for emergency communication. Technologies like smart floors and wearable devices are pivotal in fall detection and prevention [31].

- **Lifestyle enhancement**: Smart home technologies contribute to a more comfortable and convenient lifestyle, enabling control of lighting, heating, and entertainment systems through simple voice commands or mobile applications, thus aiding those with limited mobility or cognitive impairments [32].

Some of the benefits of IoT in AAL:

- **Enhanced independence**: IoT technologies allow individuals to live independently for longer, reducing the need for constant human assistance and thereby improving their quality of life and self-esteem [25].
- **Cost reduction**: By reducing the frequency of in-person healthcare visits and enabling early detection of health issues, IoT can significantly decrease healthcare costs associated with elderly care [26].

4.1. Sensor types

Ambient Assisted Living (AAL) systems represent a technological advancement in enhancing the quality of life for the elderly and individuals with disabilities. These systems integrate a variety of sensors and devices to provide support and improve safety in home environments. A core component of AAL systems' effectiveness is their ability to gather and process data from distributed sensor networks seamlessly.

The sensors employed by AAL systems can be grouped in 3 categories:

- **Wearable sensors**: are integrated into items such as wristbands, watches, and clothing. These sensors monitor physiological parameters like heart rate, blood pressure, and body temperature. Such devices are crucial for continuous health monitoring and emergency detection in AAL environments [33].
- **Environmental sensors**: are deployed within living spaces to monitor conditions such as temperature, light, and air quality. These sensors also include motion detectors and fall detection systems, which are essential for ensuring the safety and well-being of AAL users [34].
- **Multimedia sensors**: including cameras and microphones, are used for more complex monitoring needs such as activity recognition and anomaly detection. They provide audio-visual streams that can be analysed to detect emergencies or changes in routine behaviour [35].

Effective deployment of sensor networks in AAL requires consideration of user privacy, sensor coverage, energy consumption, and data reliability. Strategic placement of sensors is vital to ensure comprehensive monitoring without intruding on personal space or privacy [36].

4.2. Sensor networks

Sensor networks play a pivotal role in the infrastructure of Ambient Assisted Living (AAL) systems, enabling the collection of data necessary for monitoring, assistance, and health management of individuals, particularly the elderly.

Sensor networks in AAL must be scalable to adapt to varying user needs and flexible enough to integrate new technologies. Scalability involves not only physical network expansion but also the capability to handle increasing data volumes and processing demands [20].

Choosing the right network topology - star, tree, or mesh - impacts the network's efficiency and reliability. Mesh networks are particularly valued in AAL applications for their robustness and fault tolerance, as they provide multiple paths for data transmission, reducing the risk of system failure [36].

Ensuring interoperability among different sensor networks and integration platforms is critical. Standards such as Zigbee, Bluetooth LE, and Z-Wave facilitate communication among devices from different manufacturers [39].

4.2.1. Wireless Sensor Networks

WSNs are used in AAL environments for collecting data from various sensors placed around the living environment or worn by users. Technologies like *Zigbee*, which supports low-power mesh networking, are commonly used due to their reliability and extendibility [37].

4.2.2. Challenges in sensor network deployment

Privacy and Security

Protecting sensitive data collected by sensor networks is paramount. Robust encryption and strict access controls are necessary to safeguard user privacy and prevent unauthorized data access [38].

Ethical implications

The dependence on technology for critical care and daily activities poses ethical questions concerning autonomy, consent, and the potential for over-surveillance [39].

Interoperability

The diversity of devices and protocols in IoT can lead to issues with interoperability. Developing universal standards and protocols is essential for the seamless integration of technologies from different manufacturers [40].

Reliability and maintenance

Sensor networks must be reliable to ensure continuous monitoring. Regular maintenance, self-healing mechanisms, and redundancy are strategies used to enhance reliability [41].

Data overload and management

Handling the vast amount of data generated by sensor networks poses significant challenges. Efficient data management strategies are essential to prevent overload and ensure the system remains responsive and effective [42].

4.3. Future directions

Integrating artificial intelligence with IoT can enhance decision-making processes, providing more adaptive and responsive AAL systems that can predict individual needs and adjust services accordingly [43].

Developing intuitive user interfaces that cater to the elderly, particularly those with sensory or cognitive impairments, is critical for the widespread adoption of IoT in AAL [44].

4.4. HELICOPTER Home system

The Helicopter system is composed from two parts:

- Helicopter Home Systems: a mini-PC placed inside users' homes
- Helicopter Central Server

The HELICOPTER Home Systems structure contains (Figure 14):

- Sensors:
  - Musa sensor (for person identification)
  - Fridge_box with ID (light sensor, to detect when the fridge light is on)
  - Magnetic contact with ID
  - Chair sensor with ID
  - Bed sensor (no ID)

  - Cardea_ID for clinical sensor ID
  - Toilet sensor with ID
- Blood Pressure Monitor
- Precision Health Scale
- Blood Glucose Monitoring System
- Micro-server (mini-PC)
- Mobile device (tablet)
- Router

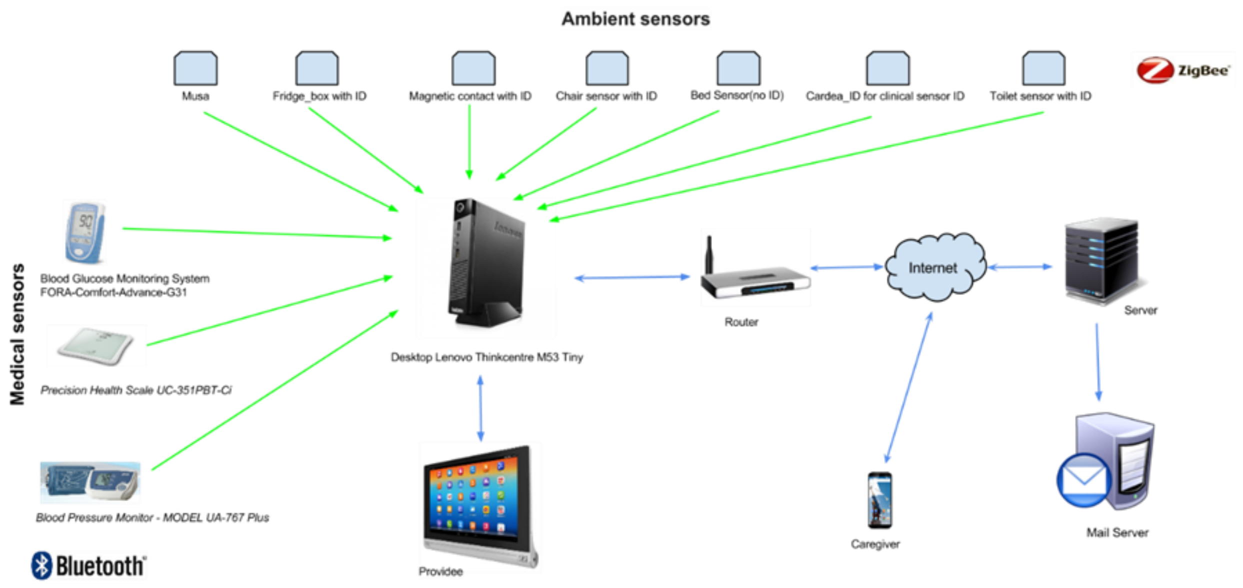

Ambient sensors
Musa
Fridge_box with ID
Magnetic contact with ID
Chair sensor with ID
Bed Sensor(no ID)
Cardea_ID for clinical sensor ID
Toilet sensor with ID
ZigBee
Medical sensors
Blood Glucose Monitoring System FORA-Comfort-Advance-G31
Precision Health Scale UC-351PBT-Ci
Blood Pressure Monitor - MODEL UA-767 Plus
Bluetooth
Desktop Lenovo Thinkcentre M53 Tiny
Router
Internet
Server
Providee
Caregiver
Mail Server

Figure 14 - HELICOPTER system structure

These sensors communicate with the micro-server through the ZigBee protocol. The Blood glucose monitoring system, Precision Health Scale and the Blood Pressure Monitor communicates with the tablet through Bluetooth.

For the HELICOPTER Home Systems software applications were created to ensure the wellbeing of the system (Figure 15):

- System self-check
- Dynamic DNS Service
- System initialization
- Automatic updates
- Data collecting and processing
- Client Interface

4.4.1. System self-check

This feature is responsible for the good functionality of the systems. It has to verify if the sensors work and transmit data to the desktop, verify if the batteries of the sensors are running low and transmit mails to specific technical persons with the sensors' status.

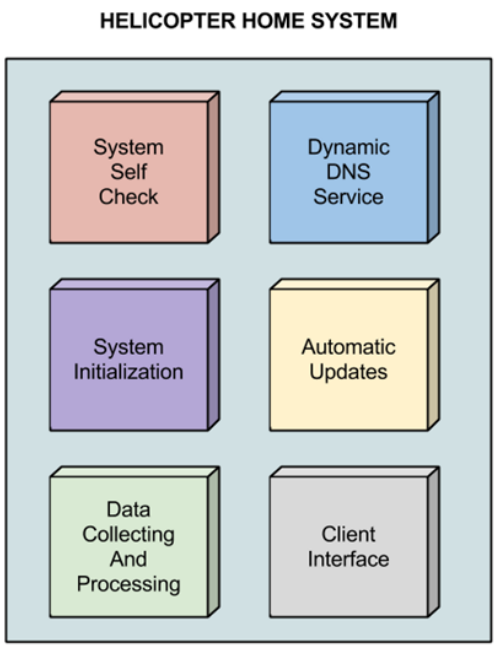


Figure 15 - HELICOPTER Home System software

4.4.2. System Initialization

System Initialization is the process of locating and using the defined values for variable data that is used by a computer program/system. In this case the System Initialization does the next steps when the mini-PC is started (as show also in the figure below):

- Checks for updated information about the network configuration (if any) on USB stick (if present)
- Clears the helicopter Glassfish domain
- Prepares initialization files and scripts
- Redeploys web-based applications
- Starts the application server (Glassfish)

4.4.3. Automatic Updates

Considering that in the Helicopter project the system software will be constantly modified for both the home and central server systems in order to improve and supplement the system's features and capabilities, an automatic update system was devised. Update packages containing, the instructions and software modules, can be installed on the mini-PCs either by using an USB stick or uploading it in a pre-determined directory on the Main Server. The automatic update software component will check periodically for these packages and, if found, will copy them to the mini-PC's storage and by restarting the machine will install them.

4.4.4. Data collecting and processing

A ZigBee connection between the sensors and the home system is open all the time. Each sensor has its own rate of transmitting data; some of the sensors transmit data only when they are used (e.g. Chair sensor with ID). The communication with the other devices (Blood Pressure Monitor, Precision Health Scale, Blood Glucose Monitoring System) is made using Bluetooth. These devices transmit the data after every use. All the data is stored in a database on the mini-PC.

Periodically, all new data will be transferred to the Main Server, where it is processed to reveal behavioral patterns or anomalies.

4.4.5. Client interface

The users are able to interact with the system using a tablet. The most important information that the users receive from the system, using the user interface, is about their wellbeing and the activities that they must do more/less. The user is also able to check the status of the sensors and sensors' batteries.

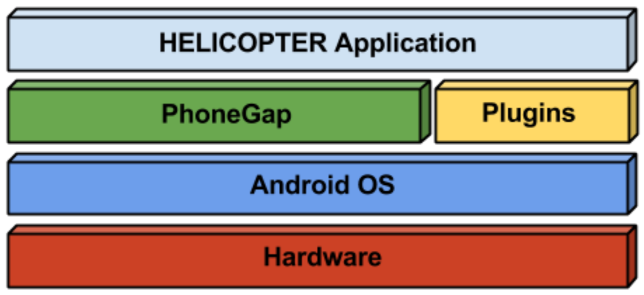


Figure 16 - HELICOPTER tablet client interface architecture

The tablets are using *Android* OS. The application is being created using *PhoneGap*, HTML5, CSS3 and JavaScript and different plugins (for interacting directly with the Android OS).

4.4.6. Home system start-up

The Home System (also named Gateway) life-cycle is presented in Figure 17.

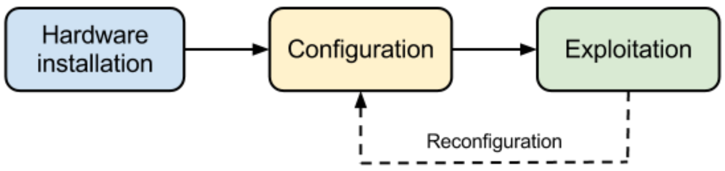


Figure 17 - Home system life-cycle [25]

The configuration consists of a file containing identification data, network data (e.g. IP address), interface information and customer information. This file is obtained from the Helicopter Server (through a dedicated service). The configuration file is read from USB stick at system startup.

The root folder (directory) of the gateway software is */opt/gwrepository*. This is the folder that contains all other files and sub-folders relevant to the project.

The listing of all files and sub-folders along with their description is presented below:

- */backups/db/* - contains the zipped dump files generated by *DumpApp*.
- */backups/db/old/* - contains the zipped dump files generated by *DumpApp* after *TransferBackupAppClient* successfully transferred them on the HELICOPTER Server.
- */logs/* - contains raw log files (in serialized object in odb format used by the *ObjectDB* database).
- */gwupdate/* - contains Gateway updates received from the HELICOPTER Server.
- */gwtmp/upd/* - contains temporary files used in the Gateway update process (files used by *GWUpdater* and *GWUSBUpdater*).
- */wars/* - contains the web archives of the HELCOPTER applications.
- *config.properties*
- *dumpapp.properties*
- *gw.properties*
- *DumpApp.jar* - dumps the database specified in *dumpapp.properties* (if there are any table names specified in the same file, only those tables will be dumped).
- *GWUpdater.jar* - brings update packages from the server (for on-line updates).
- *GWUSBUpdater.jar* - brings update packages from connected USB drives.
- *StartDeployer.jar* - deploys the HELICOPTER services on the Gateway.
- *TransferBackupAppClient.jar* - initiates communication with the HELICOPTER Server. This pair of applications has the role of exchanging dump and update files between the Gateway and the Server
- *vslogger_ 1.1.jar* - offers a server application for collecting logs from all HELICOPTER software.

The boot sequence of the Home system software is illustrated in Figure 18.

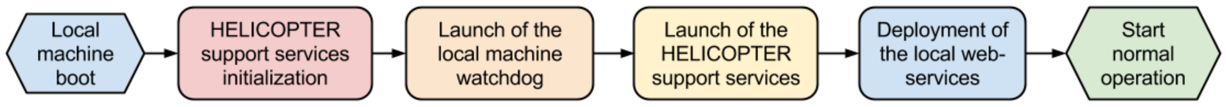

Local machine boot
HELICOPTER support services initialization
Launch of the local machine watchdog
Launch of the HELICOPTER support services
Deployment of the local web-services
Start normal operation

Figure 18 - Booting process of Home System software applications

Figure 19 shows the stages of the gateway start-up.

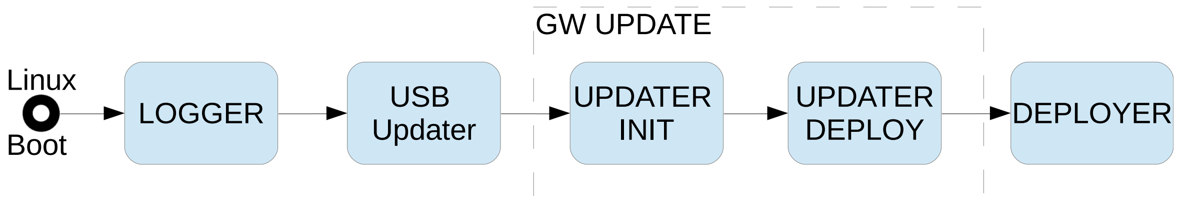


Figure 19 - Gateway start-up block diagram [25]

4.4.7. LOGGER utility

The *LOGGER* is in charge of collecting logging information from the rest of the start-up components (*USBUpdater*, *GWUpdate* and *Deployer*) and sending them to the Server.

This utility is designed as a server application that listens on port 4000 for requests from other applications. The requests are serialized Java objects and are stored in an *ObjectDB* database (more details on DB4object in [45]).

The LOGGER uses database partition, so every month is stored in its own database. The database files are stored inside */etc/opt/gwrepository/logs/db<YYYY><MM>.odb*.

The events are transferred periodically, if an external connection is available, to the Server, by calling the dedicated *LogService* webservice. This service stores the events into a MySQL table of the Server's database, attaching to them the id of the Gateway to which they belong [25].

The application resides in: */etc/opt/gwrepository/vslogger-1.1.jar*.

4.4.8. USB Updater utility

The *USB Updater* application first checks if any USB media device is connected to the computer. If so, it further seeks for a directory called GWCONFIG in the root directory of the USB media. Inside this directory, a ZIP archive file should be found and, if so, copied into the local UPDTMP (*etc/opt/gwrepository/gwtmp/upd*) directory [25].

Figure 20 presents the flowchart for this utility.

The application resides in: */etc/opt/gwrepository/GWUSBUpdater.jar*.

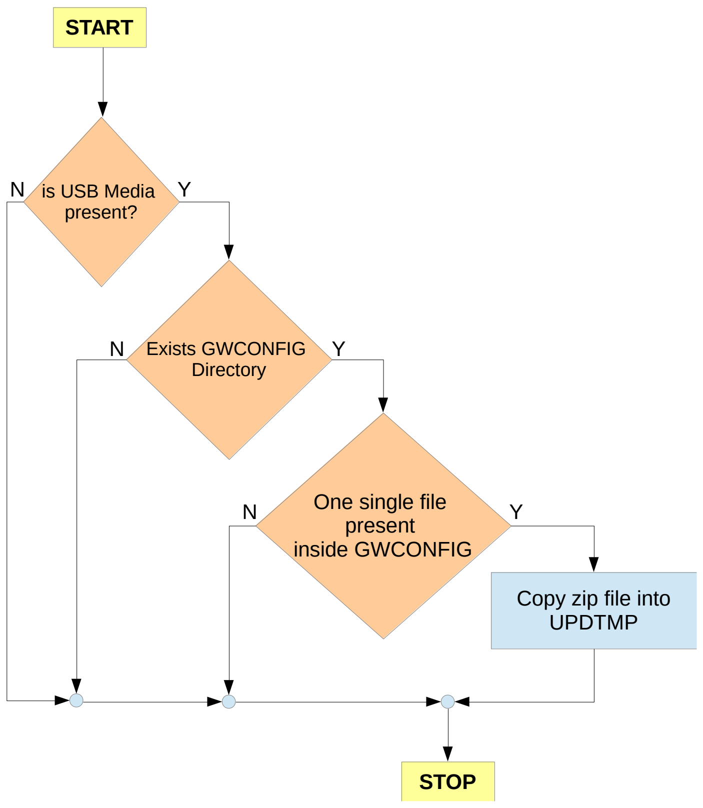

START
is USB Media present?
N
Y
Exists GWCONFIG Directory
N
Y
One single file present inside GWCONFIG
N
Y
Copy zip file into UPDTMP
STOP

Figure 20 - Flowchart for USB Updater utility

4.4.9. GW Updater utility

This utility can be split in two sub-utilities [25]:

- **Updater Init**: checks if any archive is available in the UPDTMP folder. If so, it unpacks the content of the archive in the UPDATE folder. Finally, the UPDTMP directory is emptied.
- **Updater deploy**: first, the application checks if a file named *config.properties* is available in the UPDATE (*/etc/opt/gwrepository/gwupdate/*) directory. If present, using the information provided by this file, it updates the *gw.properties* file. Secondly, it checks for any ".war" files located in the UPDATE directory. The ".war" files found are copied into the WARS (*/etc/opt/gwrepository/wars/*) directory. Then, the application verifies if the *gwconfig.xml* file is present in the UPDATE directory. The *gwconfig.xml* file is copied inside the WARS directory. Next, the available ".sql" files from the UPDATE directory are imported into the local MySQL server. Finally, the UPDATE directory is emptied.

Figure 21 presents the flowchart of this utility.

The GW Updater application resides in: */etc/opt/gwrepository/GWUpdater.jar.*

4.4.10. Deployer utility

The Deployer application's purpose is to start the Glassfish server (or restart it, if it is already started) and to deploy some specified connection pools and applications on the server.

The Deployer application reads from the *gwconfig.xml* file the information about the server and it writes the *asadmin.txt* file, containing security related information about the Glassfish server.

Next, the connection pool information is read. Afterwards, the server's *domain.xml* file is modified as follows:

1) all application nodes and other application references from the file are deleted (all applications are undeployed).
2) all jdbc-resources and resource references are also deleted (all jdbc resources are undeployed).
3) all jdb-connection-pool are deleted from the file (all jdbc connections are undeployed).
4) the connection pools information is written to the file (the specified connection pools - previously read - are deployed).

Another task of the application consists of starting/restarting the server.

Next, the *.war* files, located in the WARS directory, are deployed on the server (in command line, using the *asadmin.txt* password file).

Finally, the *asadmin.txt* file is deleted.

The flowchart for this application is presented in Figure 22 [25].

The application resides in: */etc/opt/gwrepository/StartDeployer.jar* .

4.4.11. DumpApp

This application has the role of creating backups of the database (dumps) for later transfer on the Server. The application is set to run every 24 hours using the standard Linux task scheduler (*cron*).

The output of the application is a ZIP archive containing the dump file. The ZIP archive is named after the following pattern: *<GWId>_<yyyy-MM-dd>_<HH-mm-ss>.zip* and is stored in */opt/gwrepository/backups/db/.*

Information regarding execution status is logged on the Server using the VSLogger client [25].

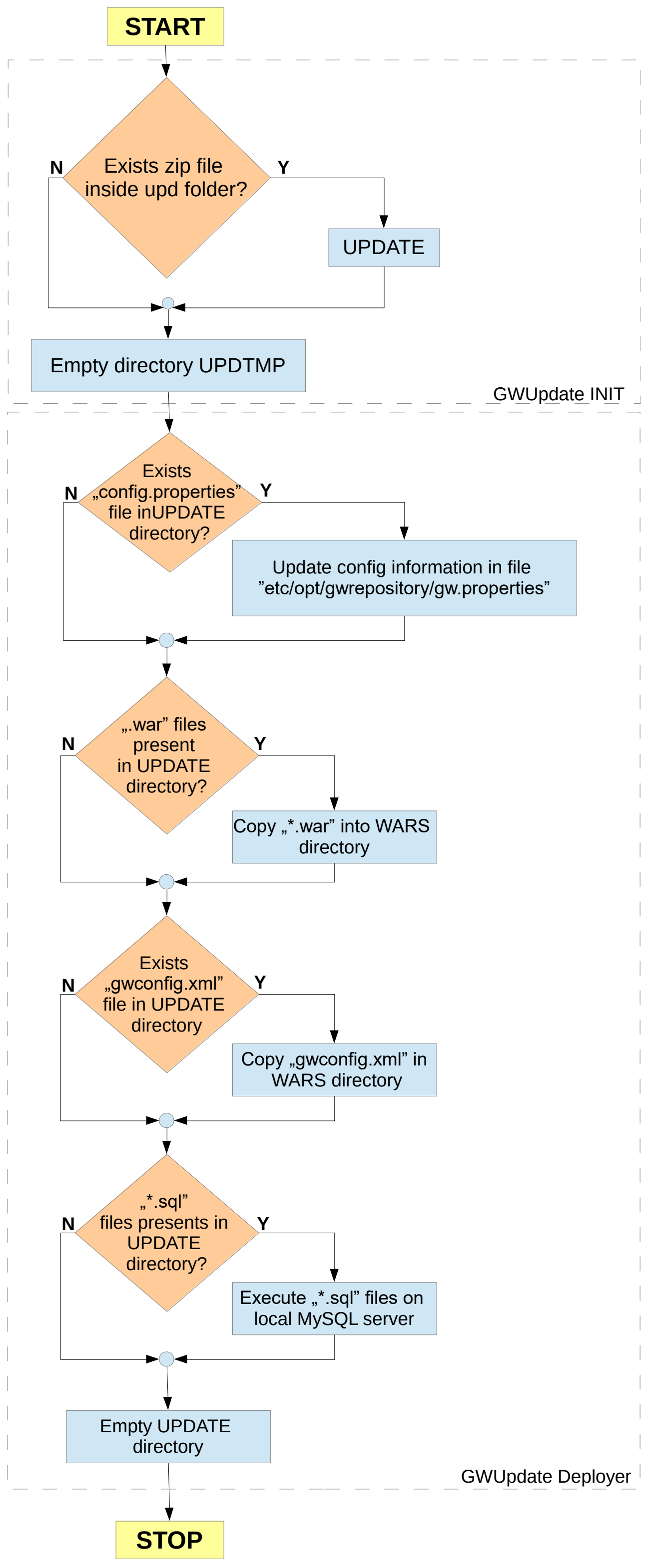

START
Exists zip file inside upd folder?
N
Y
UPDATE
Empty directory UPDTMP
GWUpdate INIT
Exists „config.properties" file inUPDATE directory?
Update config information in file "etc/opt/gwrepository/gw.properties"
„.war" files present in UPDATE directory?
Copy „*.war" into WARS directory
Exists „gwconfig.xml" file in UPDATE directory
Copy „gwconfig.xml" in WARS directory
„*.sql" files presents in UPDATE directory?
Execute „*.sql" files on local MySQL server
Empty UPDATE directory
GWUpdate Deployer
STOP

Figure 21 - Flowchart for GW Updater utility

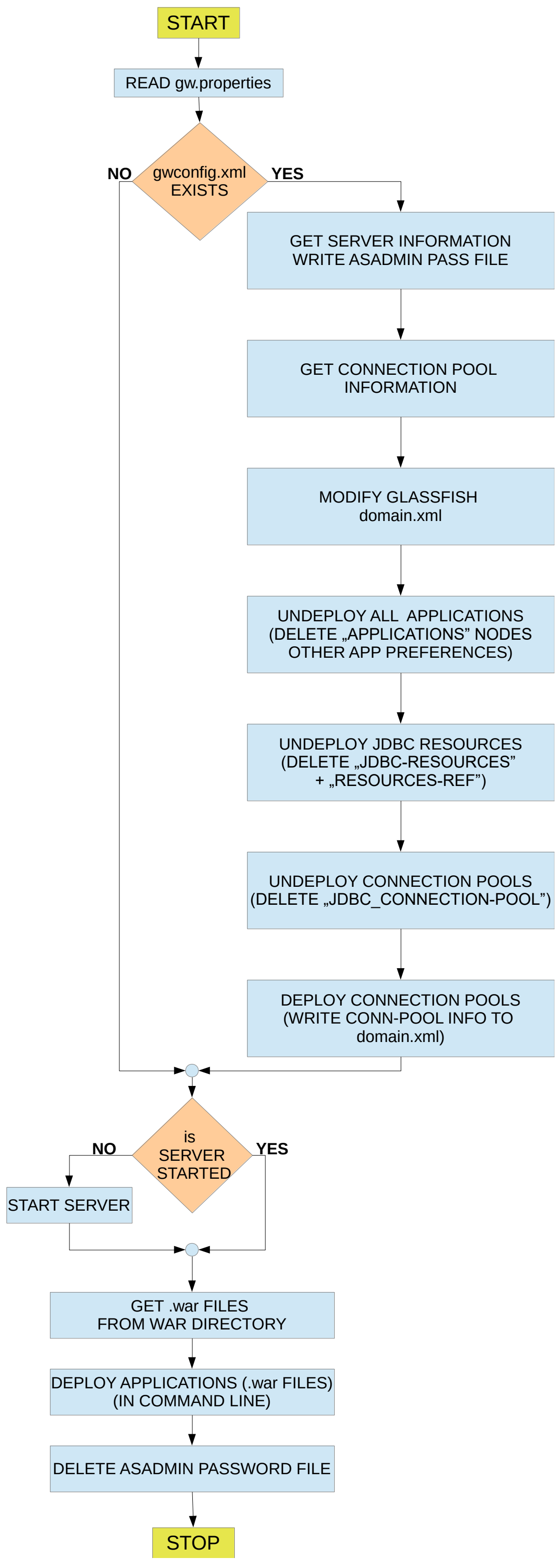

START
READ gw.properties
gwconfig.xml EXISTS
NO
YES
GET SERVER INFORMATION WRITE ASADMIN PASS FILE
GET CONNECTION POOL INFORMATION
MODIFY GLASSFISH domain.xml
UNDEPLOY ALL APPLICATIONS (DELETE „APPLICATIONS" NODES OTHER APP PREFERENCES)
UNDEPLOY JDBC RESOURCES (DELETE „JDBC-RESOURCES" + „RESOURCES-REF")
UNDEPLOY CONNECTION POOLS (DELETE „JDBC_CONNECTION-POOL")
DEPLOY CONNECTION POOLS (WRITE CONN-POOL INFO TO domain.xml)
is SERVER STARTED
START SERVER
GET .war FILES FROM WAR DIRECTORY
DEPLOY APPLICATIONS (.war FILES) (IN COMMAND LINE)
DELETE ASADMIN PASSWORD FILE
STOP

Figure 22 - Flowchart for Depoyer utility

4.4.12. TransferBackupAppClient and TransferBackupAppServer

These two applications work in pair and have the role of exchanging files (dumps and updates) between the gateway and the main server. The client side is responsible for initiating communication with the server and the server side is responsible of responding to client requests. The server can handle multiple client connections simultaneously.

The *TransferBackupAppClient* is set to run every 6 hours using the standard Linux task scheduler (*cron*), whilst *TransferBackupAppServer* is running permanently.

In order to function properly, *TransferBackupAppClient* retrieves the Server IP address from the "**config.properties**" file. Both *TransferBackupAppClient* and *TransferBackupAppServer* listens port 95.

The correctness of the file transfer is checked in both directions by comparing the file sizes and an MD5 checksums.

Information regarding execution status is logged on the Server using the *VSLogger* client [25].

4.5. The SAVE Sensor Adapter

The solution adopted within the SAVE project for collecting and analysing user data consists of the following components:

- Smartwatch
- Data capturing sensors
- Web-based graphical interface for displaying and analysing collected data

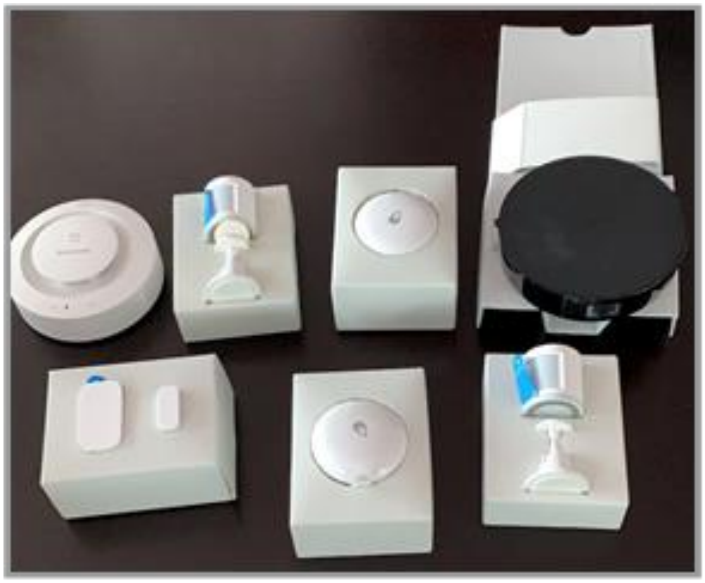


Figure 23 - Sensors included in the SAVE solution

The sensors included into the SAVE solution are (Figure 23):

- Flood detection sensor (2 pieces - for the kitchen, bathroom)
- Presence (human) sensor (2 pieces)

- Contact sensor (1 piece - for the entrance door)
- Centralization device (1 piece)

To collect data from sensors for the purpose of connecting them to the acquired devices, a "sensor adapter" was designed and developed.

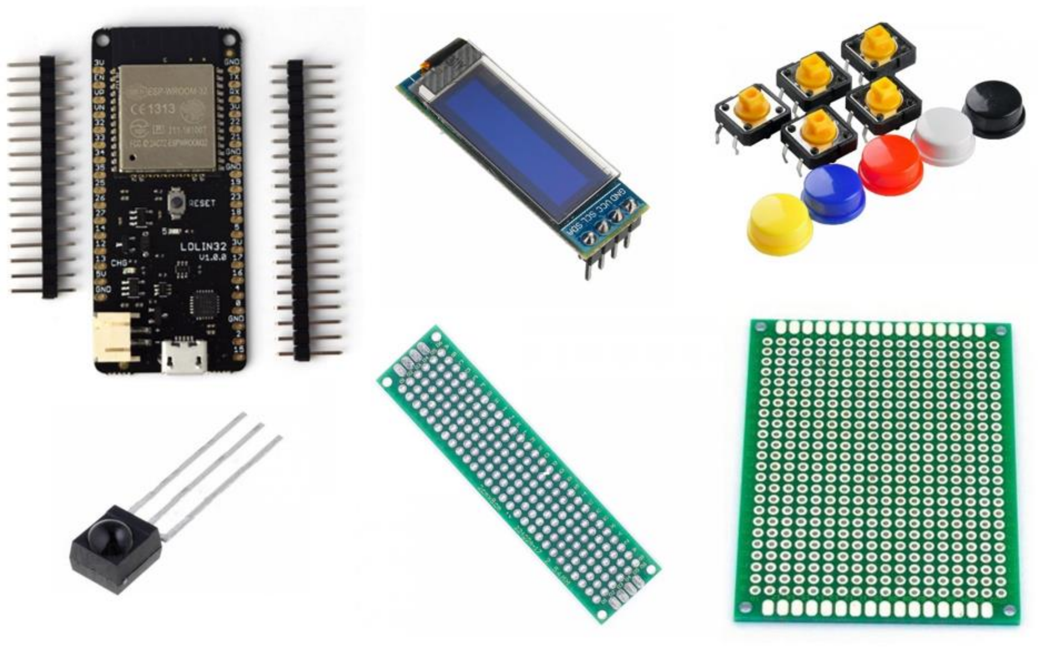


Figure 24 - SAVE Sensor Adapter components

The purpose of the *SAVE Sensors Adapter* is to read, indirectly, the status of the sensors included in the provided sensor kit. The *Aqara Hub 2* is capable of relaying events through IR (infrared) codes. The *SAVE Sensors Adapter* reads the IR codes emitted by the *Aqara Hub* and sends them to the *SAVE cloud application*, through Wi-Fi. The configuration of the Wi-Fi link is done using WPS[1] or through a direct USB connection.

4.5.1. Components

The SAVE Sensors Adapter is built around an 2-cores ESP32 development board, readily available on the market (the *NodeMCU 32S* and *LOLIN32 variants*) (). It also includes:

- a monochrome OLED graphic display, with I2C communication
- 2 push-buttons are used for the user interface
- the VS1838B IR sensor is employed to read the IR codes

[1] *WPS - Wireless Protected Setup*

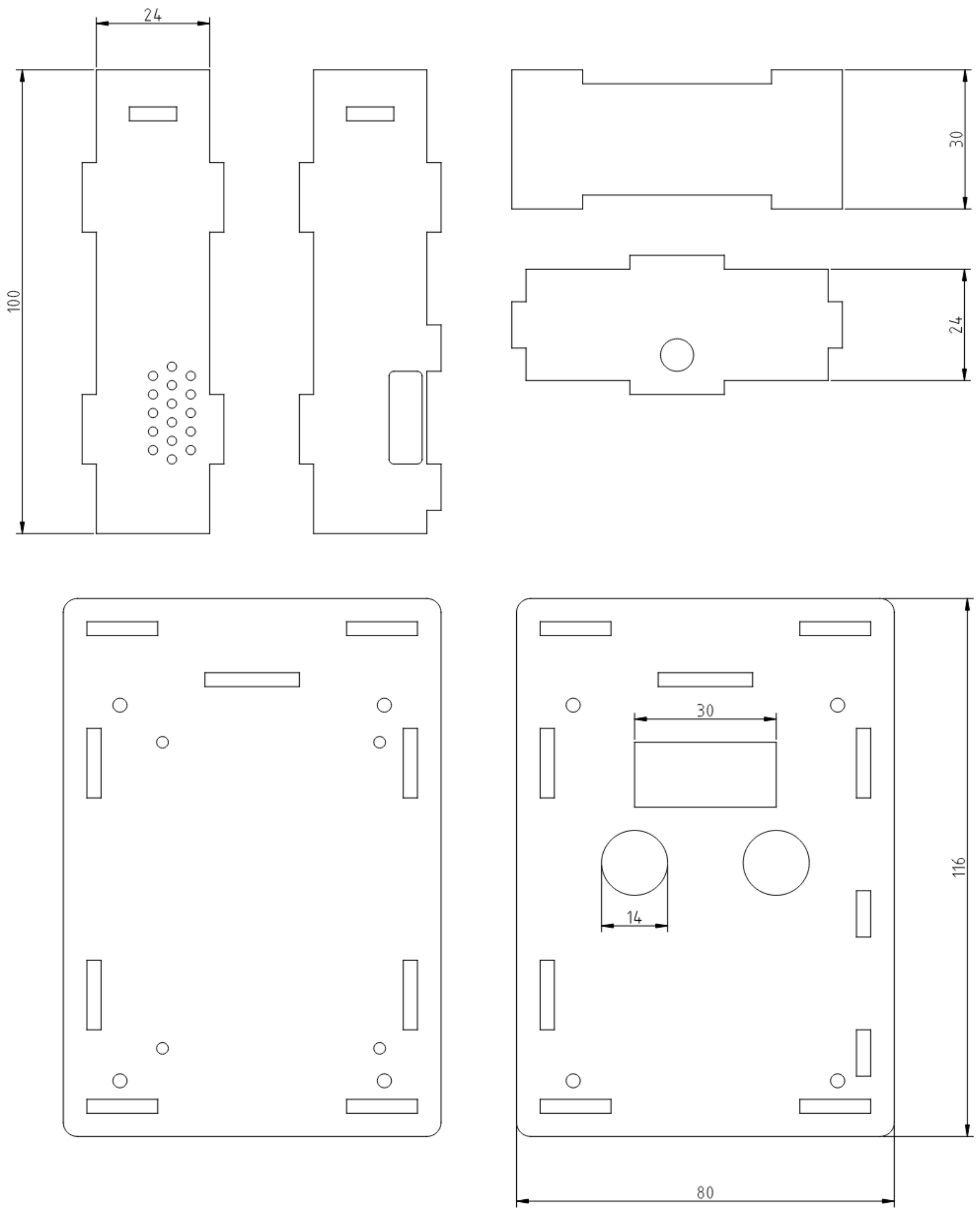


Figure 25 - SAVE Sensor Adapter case - main dimension of the case

4.5.2. The case

The case is built from black and transparent plexiglass sheets, 3mm thick, for maximum sturdiness (Figure 25). The case is held together by M3 screws.

Figure 26 shows the physical aspect of the case parts.

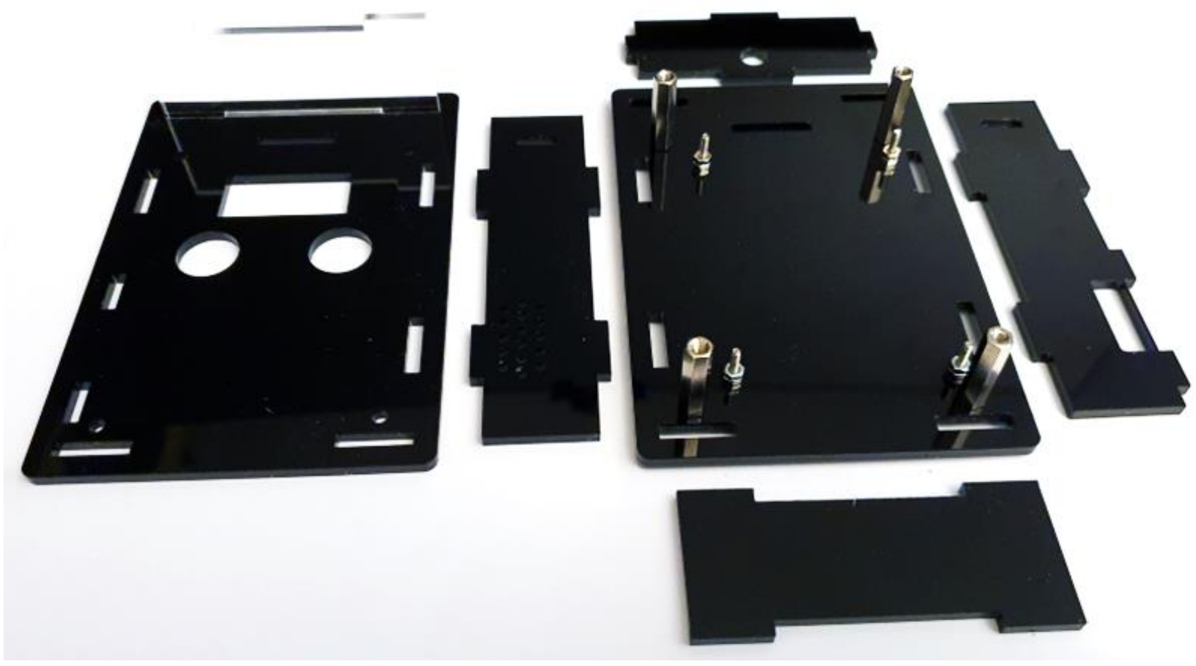


Figure 26 - SAVE Sensor Adapter case parts

4.5.3. Electrical schematic

Figure 27shows the electrical schematic:

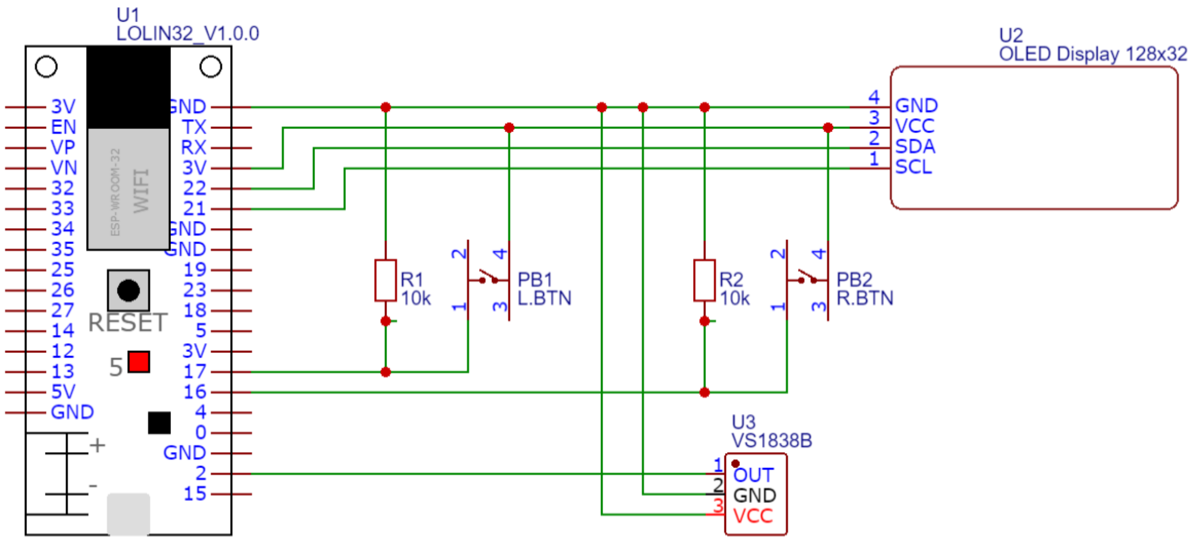


Figure 27 - SAVE Sensor Adapter schematic

The *SAVE Sensor Adapter* is powered from the USB port of the *Aqara Hub 2*.

Figure 28 presents the electronics of the device. The construction is modular, so if any part of the device malfunctions, it can quickly be replaced.

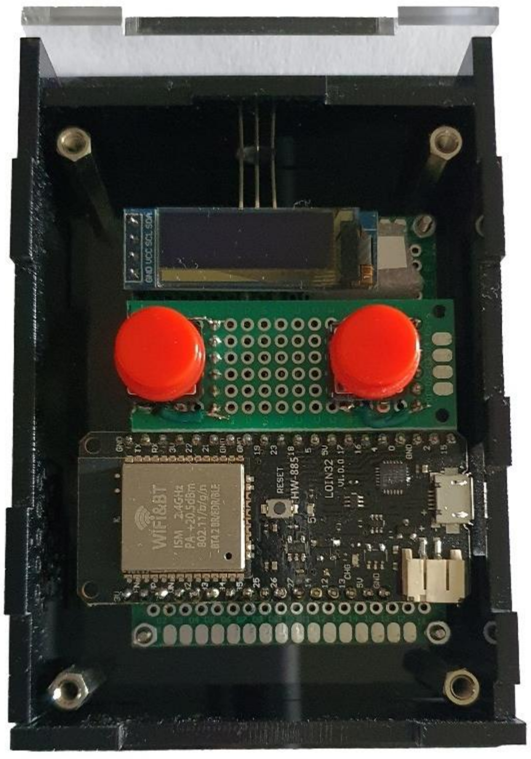


Figure 28 - The electronic of the SAVE Sensors Adapter

4.5.4. Software

The SAVE Sensor Adapter is programmed using Arduino and runs over Free RTOS OS. The software application makes use of the dual-core feature of the ESP32 microcontroller, separating the collection of data (through IR) from the user interface (Figure 29).

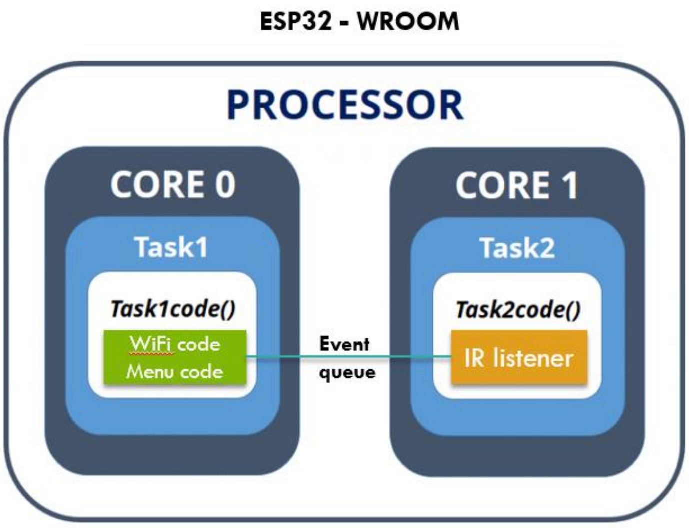


Figure 29 - Microcontroller software organization

4.5.5. The menu

The *SAVE Sensor Adapter* offers a user interface based on 2 buttons (left and right). The left button chooses an option, and the right button executes or confirms.

Figure 30 presents the structure of the user interface, as a menu. The blue screens only report information, the yellow ones allow the user to change the settings of the device.

The *Native ID* is the unique code by which the device is recognized by the SAVE solution.

The WiFi credentials can be set by WPS or by using an USB connection to a computer running the SAVE SWS (*Serial Wireless Setup*) software, developed as a Windows application especially for this device (Figure 31).

The *Reset* option erases all configuration data of the device (i.e. WiFi credentials).

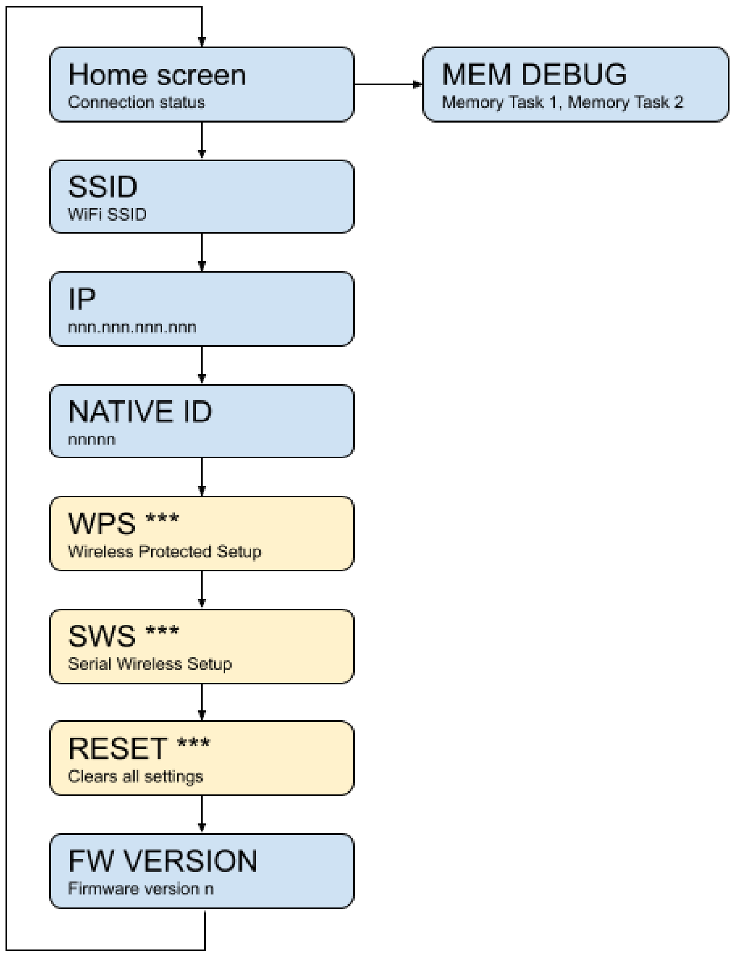


Figure 30 - Menu structure for the user interface

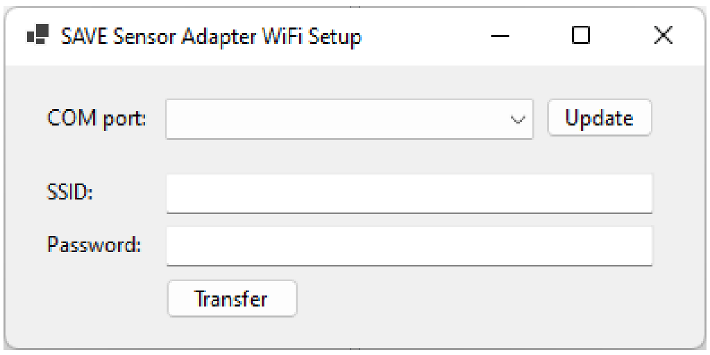


Figure 31 - SAVE Sensors Adapter Configuration application [46]

4.5.6. Final version of the device

Figure 32 shows the *SAVE Sensor Adapter* final device.

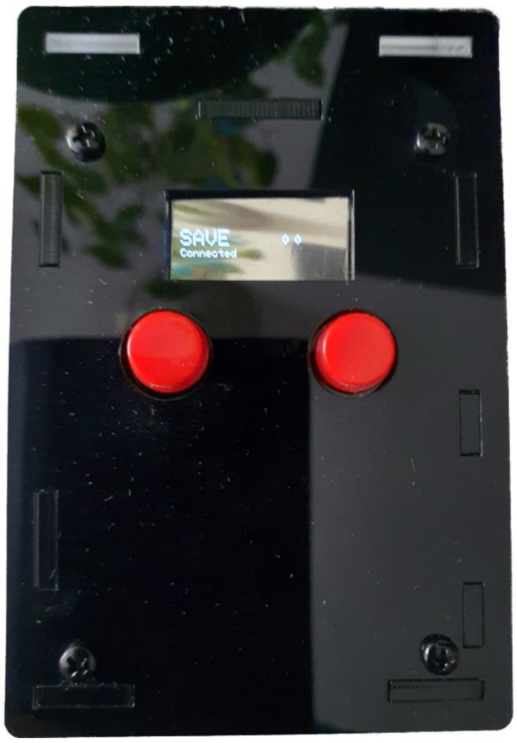


Figure 32 - SAVE Sensors Adapter [46]

5. Databases

Databases are foundational components of modern information systems, serving as repositories for storing, organizing, and retrieving data. They play a vital role in various domains, including business, healthcare, finance, education, and research. A database is essentially a structured collection of data, typically organized in tables or other structures, with built-in mechanisms for accessing and managing that data efficiently.

The evolution of databases can be traced back to the 1960s, with the advent of hierarchical and network models. Since then, relational databases emerged as the dominant paradigm, offering a flexible and powerful way to organize data into tables with rows and columns. With the rise of the internet and big data, newer database models such as NoSQL and NewSQL have emerged to handle diverse data types and scale-out requirements.

Databases can be categorized based on their data model, scalability, and use case:

- **Relational Databases**: Organize data into tables with predefined schema and support SQL for querying.
- **NoSQL Databases**: Designed to handle unstructured or semi-structured data and offer flexibility and scalability.
- **NewSQL Databases**: Combine the benefits of relational and NoSQL databases, offering scalability and ACID compliance.
- **In-memory Databases**: Store data in main memory for faster access and lower latency.
- **Distributed Databases**: Span multiple nodes or servers to handle large volumes of data and provide fault tolerance.

Key concepts in database management include:

- **Data Models**: Define the structure of the data in the database. Common models include the relational model, hierarchical model, network model, and object-oriented model.
- **Database Management System (DBMS):** Software that enables users to interact with the database, performing tasks such as data insertion, retrieval, modification, and deletion. Popular DBMSs include MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL, MongoDB, and Cassandra.
- **Normalization**: Process of organizing data to minimize redundancy and dependency, ensuring data integrity, and reducing the likelihood of anomalies.
- **Query Language**: Structured Query Language (SQL) is the standard language for managing and manipulating relational databases. NoSQL databases use various query languages tailored to their specific data models.
- **Transactions**: Atomic units of work that ensure data consistency and integrity. ACID (Atomicity, Consistency, Isolation, Durability) properties govern transactional behaviour.

5.1. Relational Database Management Systems

Relational Database Management Systems (RDBMS) represent a cornerstone of modern data management and serve as the backbone for numerous applications across various industries. RDBMSs are predicated on the relational model of data, which organizes data into tables (relations) that are composed of rows and columns. This structure facilitates efficient data retrieval, management, and manipulation, making RDBMS an essential tool for businesses and organizations.

The concept of RDBMS was first introduced by Edgar F. Codd at IBM in 1970. Codd's seminal paper, "A Relational Model of Data for Large Shared Data Banks," revolutionized the approach to database management by advocating for a table-based format rather than the then-common hierarchical or network models. This model emphasized the use of a structured query language (SQL) for data access, which became a standard due to its effectiveness and simplicity [47].

The key features of RDBMS are:

- **Data integrity and security**: RDBMS provides comprehensive tools for maintaining data integrity and security through constraints, views, and transactional control.
- **Data consistency**: Use of transactions in RDBMS, governed by ACID properties (Atomicity, Consistency, Isolation, Durability), ensures that the database remains in a consistent state even in cases of system failure or concurrent data access scenarios.
- **Normalization**: RDBMS employs normalization procedures to minimize redundancy and dependency, optimizing the schema for efficiency and maintenance.
- **SQL support**: SQL is used for creating, retrieving, updating, and deleting data, making it a powerful tool for managing relational databases.

Over the decades, RDBMS technology has evolved to incorporate distributed database systems, data warehousing, and online analytical processing (OLAP) capabilities. These enhancements have improved performance, scalability, and flexibility, catering to the growing demands of large-scale and complex applications [48].

RDBMSs are widely used in business applications ranging from financial services to e-commerce and healthcare. Despite their extensive use, modern RDBMS face challenges such as handling big data and unstructured data, requiring integration with newer database technologies like NoSQL for more flexible data handling [49].

The future of RDBMS likely involves converging with other database technologies to handle diverse data types and massive data volumes efficiently. Innovations in cloud computing and machine learning integration also point towards an adaptive, more intelligent approach to database management [50].

5.2. FOOD System databases

5.2.1. Low level database for the sensors network

The sensors network is built from wireless sensors, most of them off-the-shelf. The communication protocols used by the sensors are ZigBee and the IEEE 802.11 protocols family.

As the general architecture shows (Figure 5), the sensors communicate with the *Home System* that stores the data in a relational database (managed by the MySQL database management system).

Each object in the system (e.g. sensor, appliance etc.) has a corresponding object ID code. Each object has a variable number of properties, each identified by a variable ID code. The value of each property (variable) is represented either as integer (64 bit signed integer), float (single precision floating point) or string (variable length character string). These values are written only by the field management system (FMS).

The database employed by the lowest level of the system contains 3 tables:

State Table

This table stored the newest data of the field elements of the system. It is used to supply information about field read values to other systems.

Table name: *current_status*

Field	Type	Null	Key	Default	Description
timestamp	datetime	no			date and time of last value update, UTC
obj_id	int unsigned	no	PK	0	object ID
var_id	int unsigned	no	PK	0	variable ID in the object
int_value	bigint	yes		null	if not null, value of variable*
float_value	float	yes		null	if not null, value of variable*
string_value	varchar(4096)	yes		null	if not null, value of variable*

* Only one of the three value field can be not null.

Log Table

This table stores all values, as registered by the system.

Table name: *log_values*

Field	Type	Null	Key	Default	Description
id	bigint	no	pri	0	primary key to distinguish events, auto increment

timestamp	datetime	no			date and time of last value update, UTC
obj_id	int unsigned	no		0	object ID
var_id	int unsigned	no		0	variable ID in the object
int_value	bigint	yes		null	if not null, value of variable*
float_value	float	yes		null	if not null, value of variable*
string_value	varchar(4096)	yes		null	if not null, value of variable*

* Only one of the three value field can be not null.

Commands Table

The commands to be sent to the system are written to the commands table that is periodically polled for changes.

Table name: *commands*

Field	**Type**	**Null**	**Key**	**Default**	**Description**
id	int	no	PK	0	primary key to distinguish commands, auto_increment
timestamp	datetime	no			date and time of command, UTC
obj_id	int unsigned	no		0	object ID
var_id	int unsigned	no		0	variable ID in the object
int_value	bigint	yes		null	if not null, value of variable to be written
float_value	float	yes		null	if not null, value of variable to be written
string_value	varchar(4096)	yes		null	if not null, value of variable to be written

As commands are executed by system, they are removed from the table.

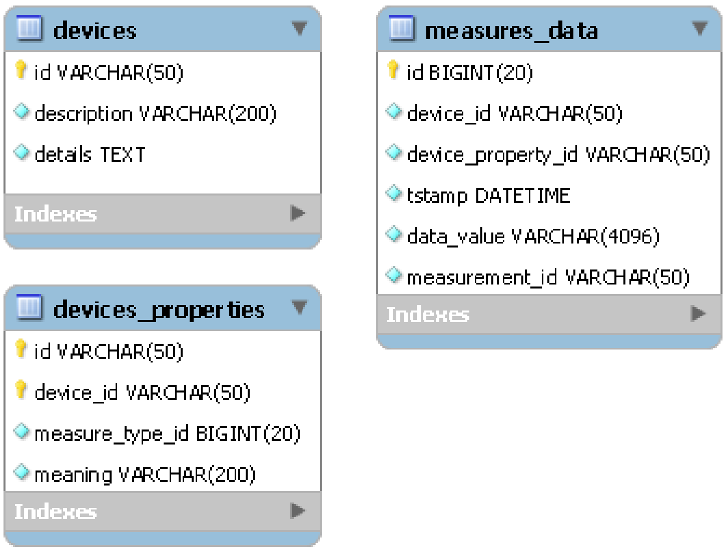
5.2.2. High level database for the web services

At a higher level, the web services use a second relational database (also managed by the MySQL RDBMS) that is synchronized periodically with the sensors (low-level) database. The main tables are presented in Figure 33.

The *devices* table contains a list of all the devices in the smart kitchen (sensors, appliances, etc); this information can be used when developing user interfaces.

Each device can have one or more properties (equivalent to the variables from the lower level), stored in the *devices_properties* table. A meaning can be attached to each property (e.g. oven temperature, air humidity inside the kitchen). Also, the *measure_type_id* field is used to retain what type of data the property stores (e.g. humidity, measured in %).

The *measures_data* table stores the actual data read from sensors. The *device_id* and *device_property_id* fields are used to link the data with the device and the property of the device. Each row in this table has a timestamp attached (the *tsstamp* field); this information is provided by the lower level. The data is stored in the *data_value* field, as a string. The *measurement_id* field is used to link data that must be interpreted together (e.g. if a sensor reads the blood pressure of a person, then we must link the high blood pressure and the low blood pressure for the same measurement, so it can be interpreted correctly); the value of this field is the timestamp of the measurement.



```
devices
id VARCHAR(50)
description VARCHAR(200)
details TEXT
Indexes

devices_properties
id VARCHAR(50)
device_id VARCHAR(50)
measure_type_id BIGINT(20)
meaning VARCHAR(200)
Indexes

measures_data
id BIGINT(20)
device_id VARCHAR(50)
device_property_id VARCHAR(50)
tstamp DATETIME
data_value VARCHAR(4096)
measurement_id VARCHAR(50)
Indexes
```

Figure 33 - Tables for storing the FOOD sensors' data

The commands are written directly to the lower-level database, for performance purposes.

5.2.3. The database structure of the FOOD platform

The FOOD framework is the base for the interaction of the users with the system for reporting and configuration. Also, through its modules and functions facilitates the access to the FOOD web services. The platform will be accessed not only by ordinary users, but also by the administrators, so it must offer options to manage the access rights to its functionalities.

The platform must support multiple languages so it can be used with the same ease, independent of the language spoken by the user.

Figure 34 presents the conceptual schematic of the core database. Module dependent tables will be added as the work on the specific facilities is progressing.

FUNCTIONS table

Functions are the units that implement atomic functionalities. These functions are related to code that fulfils a specific task. For an easier management, the functions are grouped into modules. For traceability purposes, a function can be monitored, so every access to it is recorded into the *monitoring* table.

MODULES table

Modules group functions that work with the same entities (for example, user management, menu management etc.) Modules have a name and a description.

USERS table

The *users* table records all data about the users (name, surname contact data, username, password etc.) For the password an expiry date can be set. The *is_admin* flag makes the difference between the ordinary users and administrators.

Administrators can access all the facilities of the platform.

PROFILES table

For the management of the access right to the platform's facilities, profiles define levels of access. The access rights are granted at function level. All the functions accessible at each level are stored into the *profile_functions* table.

To each user can be assigned profiles and implicitly access rights. These assignments are recorded into the *users_profiles* table.

MONITORING table

This table records all calls to monitored functions. It stores the user and time information. Based on the information from this table the platform will be able to display user and functions related statistics.

SESSIONS table

This table is used to keep track of all active sessions in order to authenticate and authorize on-line users. This table will be visible only for the administrators of the platform.

MENU_ITEMS table

The facilities of the platform will be accessible through a menu. The structure of this menu will be stored into the *menu_items* table in a recursive manner. The *action* field indicates what happens when the menu option is clicked. The actions can be of two types: internal actions – clicking on the option calls a function of the platform, and external actions – opens a URL address. It is possible to open the new facility in a new window by using the flag *open_in_new_window*. Also, it is possible to send additional parameters by using the *param* field.

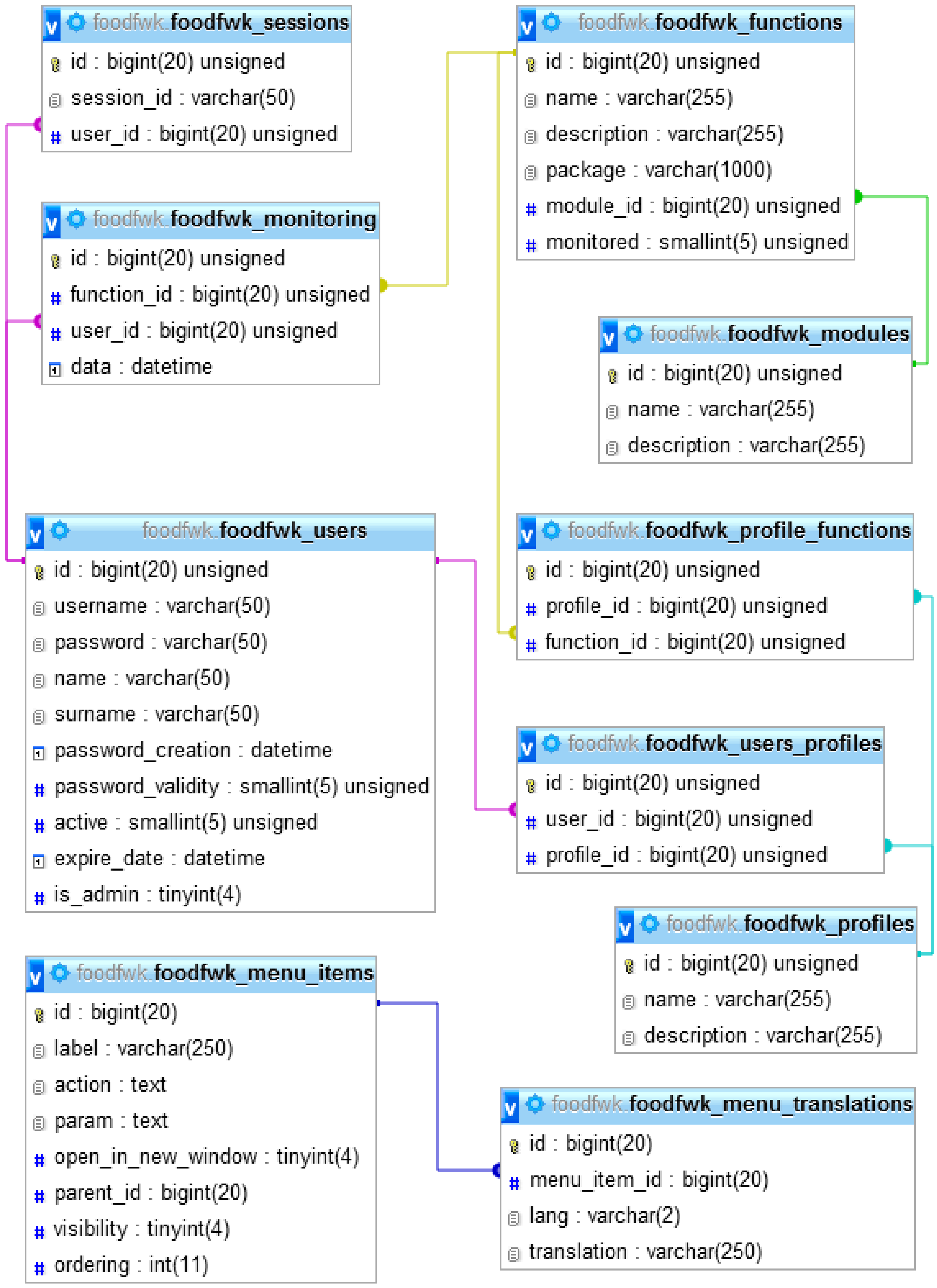

foodfwk.foodfwk_sessions
id : bigint(20) unsigned
session_id : varchar(50)
user_id : bigint(20) unsigned
foodfwk.foodfwk_functions
id : bigint(20) unsigned
name : varchar(255)
description : varchar(255)
package : varchar(1000)
module_id : bigint(20) unsigned
monitored : smallint(5) unsigned
foodfwk.foodfwk_monitoring
id : bigint(20) unsigned
function_id : bigint(20) unsigned
user_id : bigint(20) unsigned
data : datetime
foodfwk.foodfwk_modules
id : bigint(20) unsigned
name : varchar(255)
description : varchar(255)
foodfwk.foodfwk_users
id : bigint(20) unsigned
username : varchar(50)
password : varchar(50)
name : varchar(50)
surname : varchar(50)
password_creation : datetime
password_validity : smallint(5) unsigned
active : smallint(5) unsigned
expire_date : datetime
is_admin : tinyint(4)
foodfwk.foodfwk_profile_functions
id : bigint(20) unsigned
profile_id : bigint(20) unsigned
function_id : bigint(20) unsigned
foodfwk.foodfwk_users_profiles
id : bigint(20) unsigned
user_id : bigint(20) unsigned
profile_id : bigint(20) unsigned
foodfwk.foodfwk_profiles
id : bigint(20) unsigned
name : varchar(255)
description : varchar(255)
foodfwk.foodfwk_menu_items
id : bigint(20)
label : varchar(250)
action : text
param : text
open_in_new_window : tinyint(4)
parent_id : bigint(20)
visibility : tinyint(4)
ordering : int(11)
foodfwk.foodfwk_menu_translations
id : bigint(20)
menu_item_id : bigint(20)
lang : varchar(2)
translation : varchar(250)

Figure 34 - FOOD platform database schematic

MENU_TRANSLATIONS table

To implement multiple languages into the platform, the menu option must be dynamically selected according to the platform's language. The *menu_translations* table stores translations for all options of the menu. If a translation is not found for an option, then the label of the menu item will be used.

5.3. SAVE solution database

To manage data persistence, the SAVE system relies on a relational database management system, Oracle's MySQL. This choice provides an efficient solution for organizing and storing the system's information. The database structure includes tables for kits, device types, and devices (Figure 35). Kits are registered in the kits table and receive user-friendly names for easy identification. Device types are stored in the *device_types* table, receiving acronyms and default descriptions. The acronyms help user interfaces identify the relevant components for handling data from specific types of devices. Devices are registered in the devices table, associated with a particular kit (of devices) and linked to a device type. Short and long descriptions provided by end-users facilitate identification and differentiation of multiple sensors of the same type.

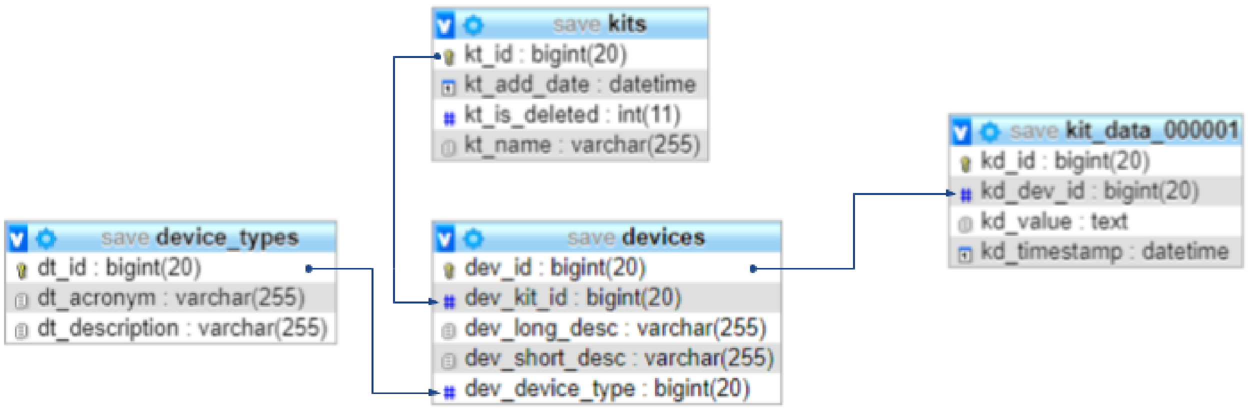


Figure 35 - Tables for the data collecting system [27]

The values read from the sensors (the „data") are persisted, partitioned at kit level, in the *kit_data_** tables. There is one table for each kit. The *kd_value* field contains the sensor data, as transmitted by the sensors itself or a sensor adapter.

Figure 36 presents the structure of users table and several other tables used by the microservices that implement the user interfaces and the link to the smartwatch applications. The password is not kept in clear text, but only the SHA-1 hash is stored.

The *not_data* table stores the scheduled notifications of the end-users. These are transferred periodically (every 10 minutes or soon as the smartwatch connects to the SAVE cloud application) to the smartwatch. The SAVE web application user interface manages these records.

It must be highlighted that the data coming from the sensors are not linked directly to the end-users, but to the kits. Accessing only this data does not reveal anything about the users' identities.

The temporary GPS data is collected in the *gps_data* table. The maximum duration for which the records are kept is 10 days.

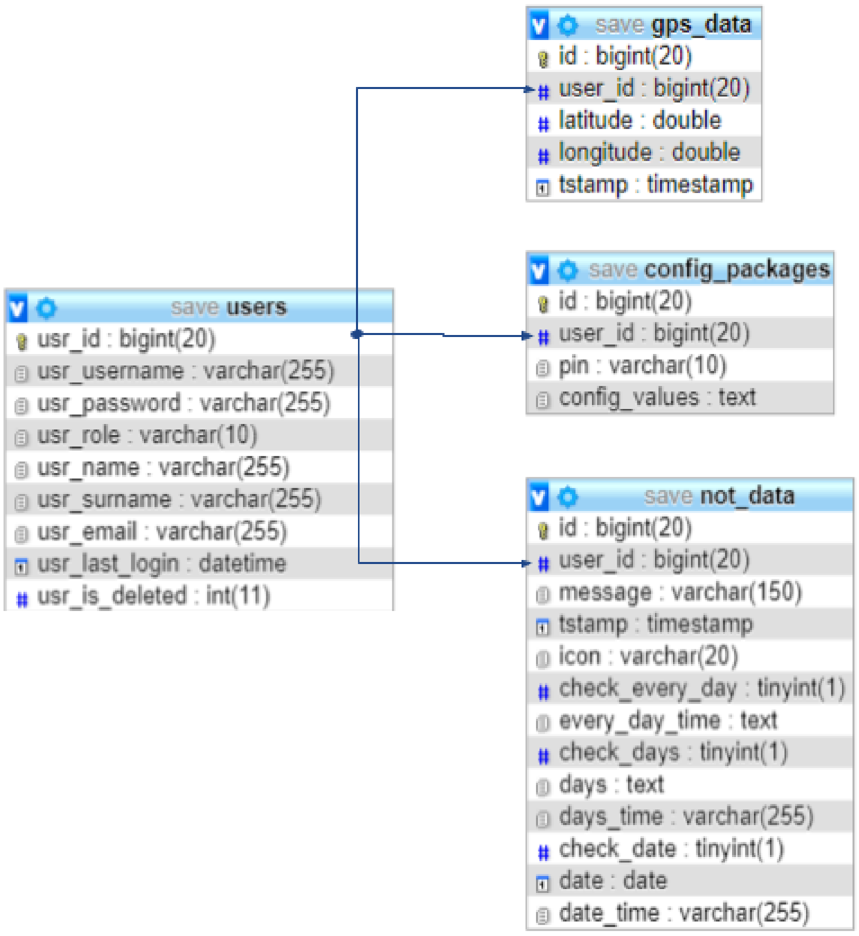


Figure 36 - Tables for the users, notifications, smartwatch app configuration, GPS data [27]

6. Human-computer interaction

6.1. User interface design and development requirements

User interface design or user interface engineering is the design of websites, computers, appliances, machines, mobile communication devices, and software applications with the focus on the user's experience and interaction. The goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals—what is often called user-centered design. Good user interface design facilitates finishing the task at hand without drawing unnecessary attention to itself. Graphic design may be utilized to support its usability. The design process must balance technical functionality and visual elements (e.g., mental model) to create a system that is not only operational but also usable and adaptable to changing user needs.

Interface design is involved in a wide range of projects from computer systems to cars, to commercial planes; all these projects involve many of the same basic human interactions yet also require some unique skills and knowledge. As a result, designers tend to specialize in certain types of projects and have skills centered around their expertise, whether that is software design, user research, web design, or industrial design.

A device pixel (or physical pixel) is the smallest physical unit in a display.

Screen density refers to the number of device pixels on a physical surface. It is often measured in pixels per inch (PPI).

A CSS pixel (or a device-independent pixel) is an abstract unit used by browsers to draw content precisely and consistently on Web pages (Figure 37).

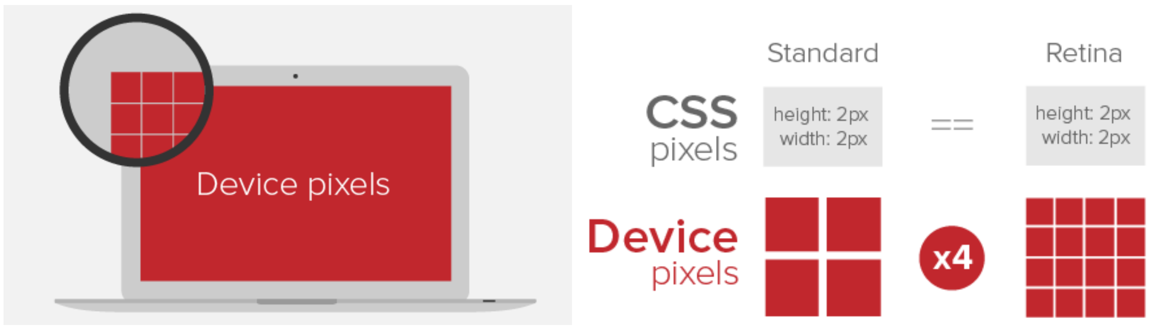


Figure 37 - Device pixel

Pixel ratio is the ratio between the number of device pixels needed to draw a CSS pixel on a tablet screen and the number of device pixels needed to write a CSS pixel on a standard screen. For the *Xperia Tablet Z* this ratio is 0.(6):

- An image of size 1920 x 1080 pixels would use 1920 × 1080 device pixels to be drawn on screen. On *Xperia Tablet Z* display, the same image would use 1280 × 720 device pixels to keep the same physical size.

To be compatible with devices having different pixel ratios, the UIs must define different CSS for each supported pixel ratio (implemented in sensors UIs) [51].

6.2. UI design and development requirements for elderly

Possible implication in software design according to cognitive decline in the elders:

- **Vision**: one of the most appropriate texts for information display is sans-serif fonts with the size between 12 and 14 points - the width of the visual field of elderly people is reduced.
- **Color**: older people have less sensitivity to color contrast especially in the blue green range [52]; designers should not use colors to communicate meaning but should use for supporting information presentation.
- **Memory**: the average of the related items to be shown in the display panel used for the older people should be around 5.5 items [53] - for older people, use of long-term memory is much more effective than short term memory.
- **Sound**: the devices that require sound as alarm, instruction or any activity that require attention from the elderly users should use the lower range of frequency (between 500 and 1000 Hz) - high pitched sounds with peaks over 2500 Hz are mostly missed by the elderly [54].
- **Attention and Simplicity**: the use of relevant graphics and pictures are more significant than the use of detailed decorations; multitasking operations should not be applied as well - older adults have problems maintaining attention over long periods of time [55].
- **Motor decline**: small screens on mobile devices may limit usage of elderly users, but a tablet device still has mobility and is still not compromised with screen size - physical decline is one of the general problems for the elders.
- **Reduction of complexity**: the design of the interaction should avoid complexities, for example using short and long press, using combination keys, using multi fingers, using multi touch, etc. - simpler is more useable for elderly users.
- **Clear structure of tasks**: clearly separated task is the factor that may increase usage performance for elderly - single task per page reduces attention load for older users.
- **Consistency of information**: navigation bars, labeling or any interface components should be used to communicate exactly where the users are in the application - elderly users are easier to recognize information than to recall memory.
- **Rapid and distinct Feedback**: feedback of every action should be provided within a certain time, and it should indicate the result or response of each action – due to limitation of short-term memory of older adults.
- **On screen help**: on screen help within the operation page should be used - older people have anxiety about using new products.
- **User support**: reduce usage of error messages to become as low as possible; in case that error messages must be shown, error messages should be simple, precise, polite, and understandable - elderly users are sensitive for errors from their actions.

Interface optimization (according to the limitation of motor skills and cognitive abilities of the elderly people) [51]:

- Make use of proper size interface components:
  - Touch sensitive area/Size of button should be 16.5 mm to 19.05 mm
  - Spacing size between button/touch sensitive 3.15 mm to 12.7 mm
- Avoid using scroll bar
- Keep operation area in the center of working page
- Make use of multi model communication
- Make use of real object-liked interface
- Present text the simplest way:
  - Size 14 (~5mm on 72 dpi screen) or higher
  - Make use of sans-serif fonts
  - Make use of black font on white background
  - Avoid using fancy text (moving, non-horizontal orientation, splash etc.)

6.3. Planning of services and user interfaces

Several methods of describing the services were employed in the project:

- *conceptual maps*: employed for identifying relations between services, their touch points and required facilities (which later imposes interface elements) (Figure 38).
- *blueprints*: used to express the customer journeys (Figure 39).
- *mockups and screen flows*: for showing how the interface will look (Figure 40).
- *wireframes*: combines blueprints and mockups to express the look-and-feel of the user interface.

6.4. End-users' FOOD user interface

The technologies employed to implement the user interface as a tablet app are:

- Android OS (version 4.x)
- PhoneGap as framework for developing mobile applications for Android
- HTML5 for layout &
- CSS3 for styles (look)
- JavaScript
- jQuery library for AJAX calls to services
- jQuery Mobile library for form elements and transitions

- XML for data structuring

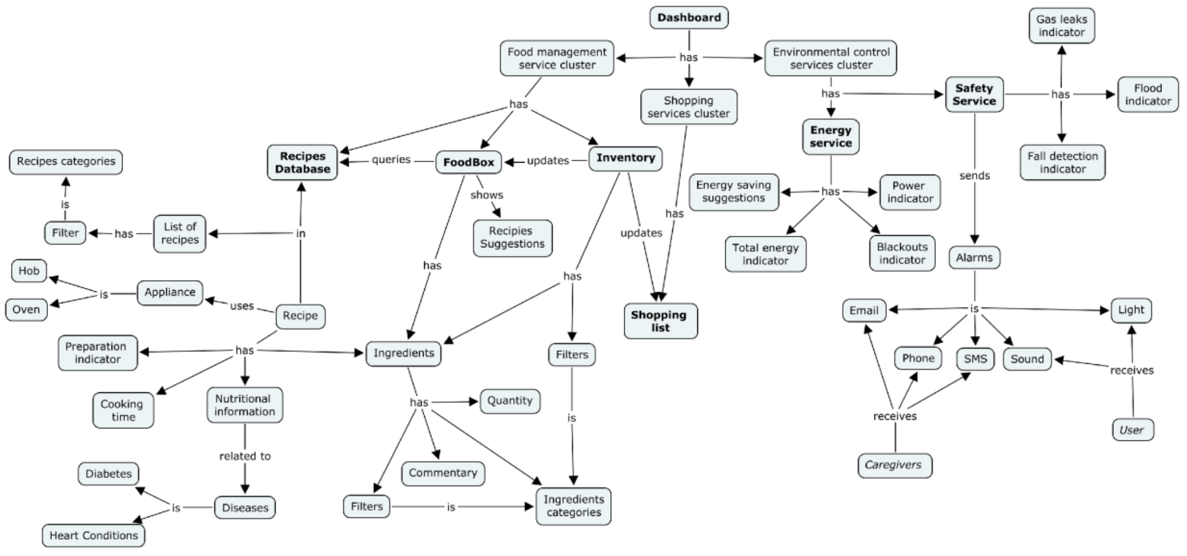
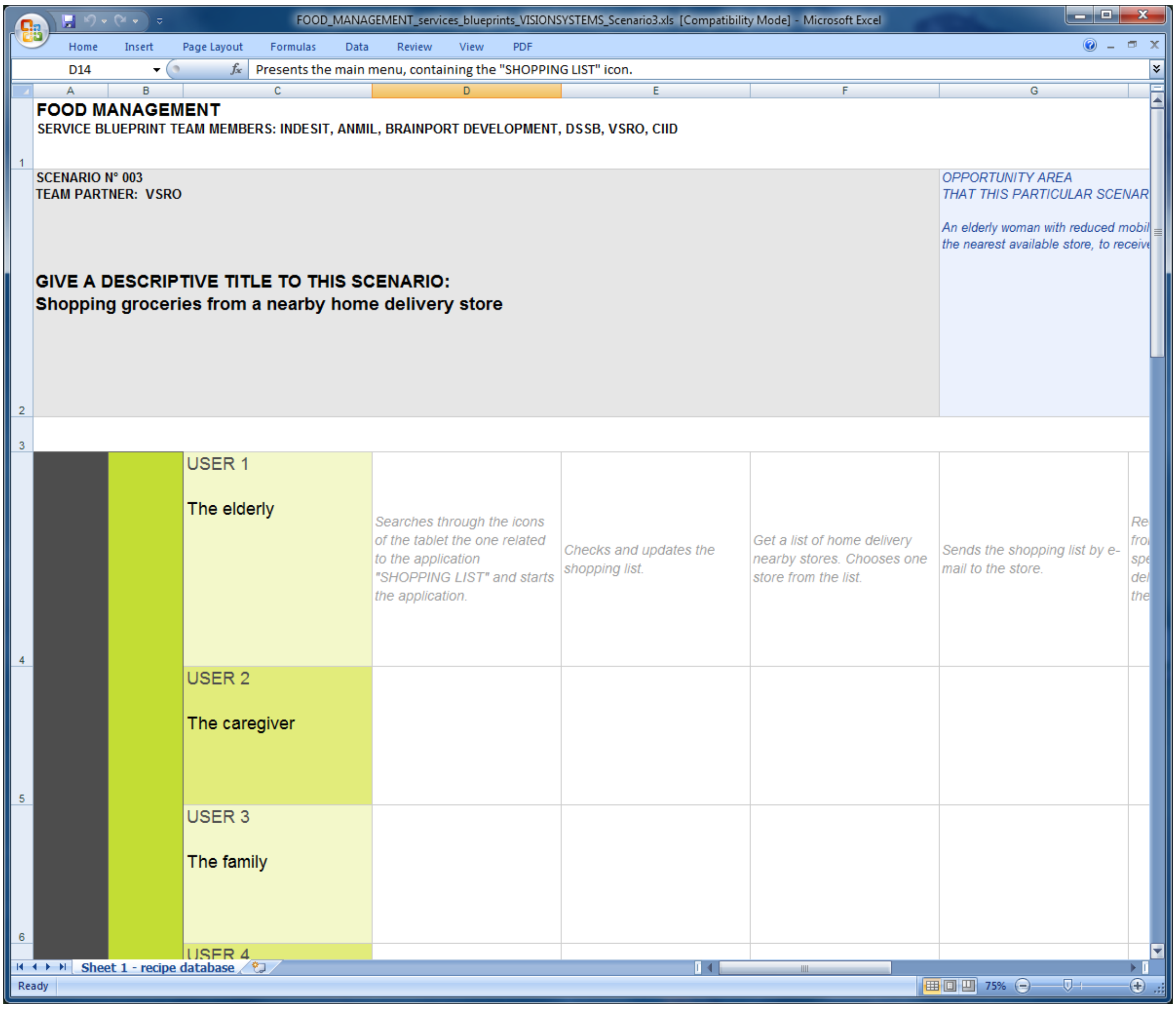


Figure 38 - Conceptual map for describing services and interface elements example



FOOD_MANAGEMENT_services_blueprints_VISIONSYSTEMS_Scenario3.xls [Compatibility Mode] - Microsoft Excel
Home
Insert
Page Layout
Formulas
Data
Review
View
PDF
D14
Sheet 1 - recipe database
Ready
75%

Presents the main menu, containing the "SHOPPING LIST" icon.

FOOD MANAGEMENT
SERVICE BLUEPRINT TEAM MEMBERS: INDESIT, ANMIL, BRAINPORT DEVELOPMENT, DSSB, VSRO, CIID

SCENARIO N° 003
TEAM PARTNER: VSRO

GIVE A DESCRIPTIVE TITLE TO THIS SCENARIO:
Shopping groceries from a nearby home delivery store

OPPORTUNITY AREA
THAT THIS PARTICULAR SCENAR

An elderly woman with reduced mobil
the nearest available store, to receiv

USER 1 The elderly	*Searches through the icons of the tablet the one related to the application "SHOPPING LIST" and starts the application.*	*Checks and updates the shopping list.*	*Get a list of home delivery nearby stores. Chooses one store from the list.*	*Sends the shopping list by e-mail to the store.*	*Re fro spe del the*
USER 2 The caregiver					
USER 3 The family					
USER 4					

Figure 39 - Blueprint example [56]

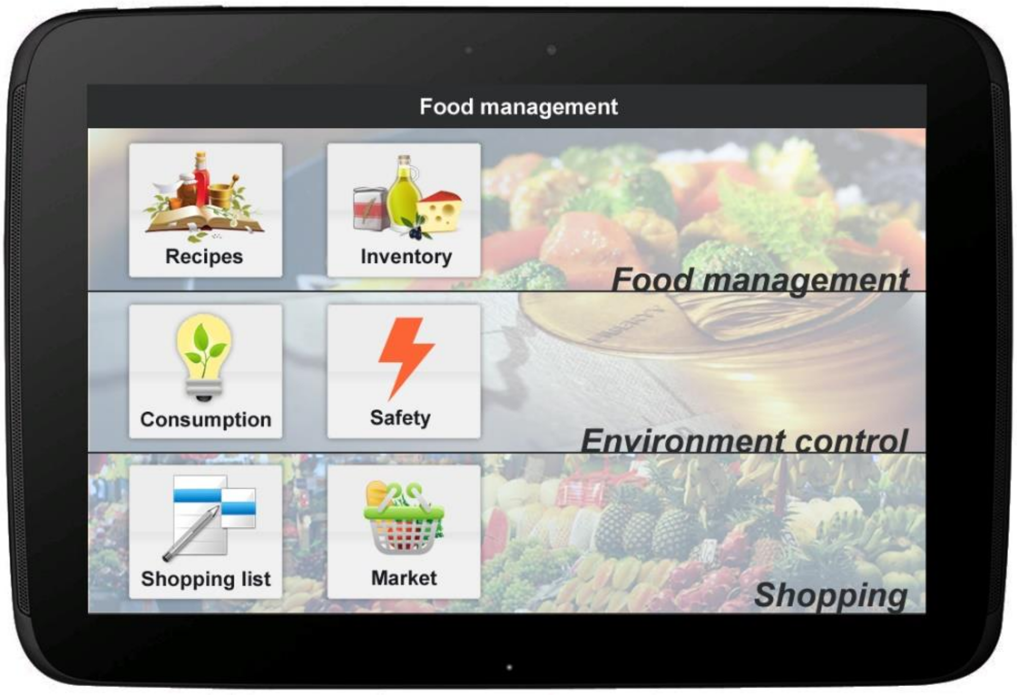


Figure 40 - Mockup example

The stack of technologies used to develop client applications (mobile applications) is shown in Figure 41.

The *PhoneGap* framework offers a JavaScript API that hides the Android API.

The *jQuery Mobile* library implements graphic controls useful for mobile applications.

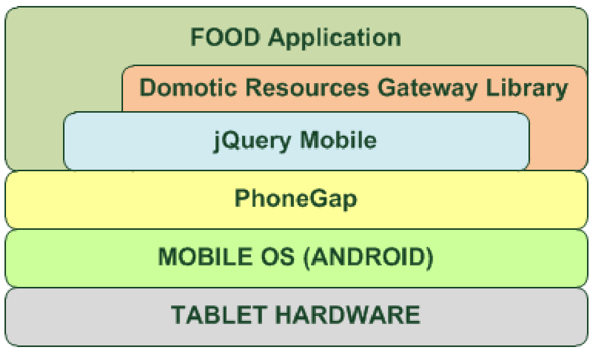
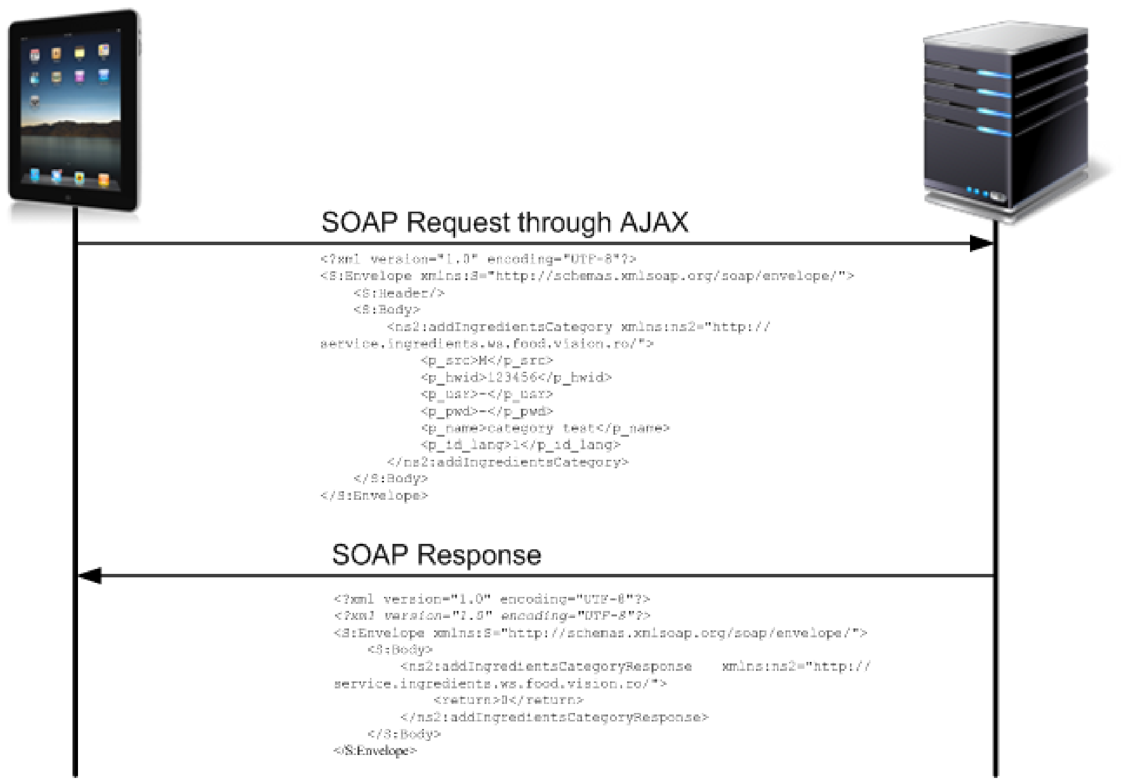


Figure 41 - FOOD mobile application technology stack

The *Domotic Resources Gateway* library is a Javascript library developed to offer an API to access the FOOD webservices. This API manages the creation of SOAP calls and the management of the response (Figure 42). Also, this API manages the selection of the right language for the application.



```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:addIngredientsCategory xmlns:ns2="http://
service.ingredients.ws.food.vision.ro/">
            <p_src>M</p_src>
            <p_bwid>123456</p_bwid>
            <p_usr>-</p_usr>
            <p_pwd>-</p_pwd>
            <p_name>category test</p_name>
            <p_id_lang>1</p_id_lang>
        </ns2:addIngredientsCategory>
    </S:Body>
</S:Envelope>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:addIngredientsCategoryResponse    xmlns:ns2="http://
service.ingredients.ws.food.vision.ro/">
            <return>0</return>
        </ns2:addIngredientsCategoryResponse>
    </S:Body>
</S:Envelope>
```

Figure 42 - SOAP request-response [57]

The flow chart in Figure 43 illustrates the way that the application for the tablet starts and initiates contact with the GATEWAY (GW) and the MAIN SERVER (MS) every time.

On every launch the tablet app searches for the configuration file, which is the one that indicates whether an account has been created on the device (if found, the app starts to establish connection with the MS and with the GW) or if the app is at its first launch (if it isn't found, the app lets the user create a new account).

After the first verification, the tablet app verifies the external connection (with the MS) and the internal connection (with the GW). At this stage, the app can perform in four different ways:

- If the app establishes **only a local connection** (with the GW): the tablet app has limited functionality (only pages with offline functionality are active, such as 'Home', 'Oven' and 'Settings').
- If the app establishes **only an external connection** (with the MS): the tablet app has limited functionality (only pages with online functionality are active, such as 'Home' and 'Messages').
- If the app establishes a **local and an external connection** (with the GW and the MS): the tablet app launches with full functionality.
- If the app doesn't succeed in establishing **neither a local nor an external connection**: the tablet app remains on the 'LOGIN' page and verifies connection periodically to see if any of the servers are available for connection.

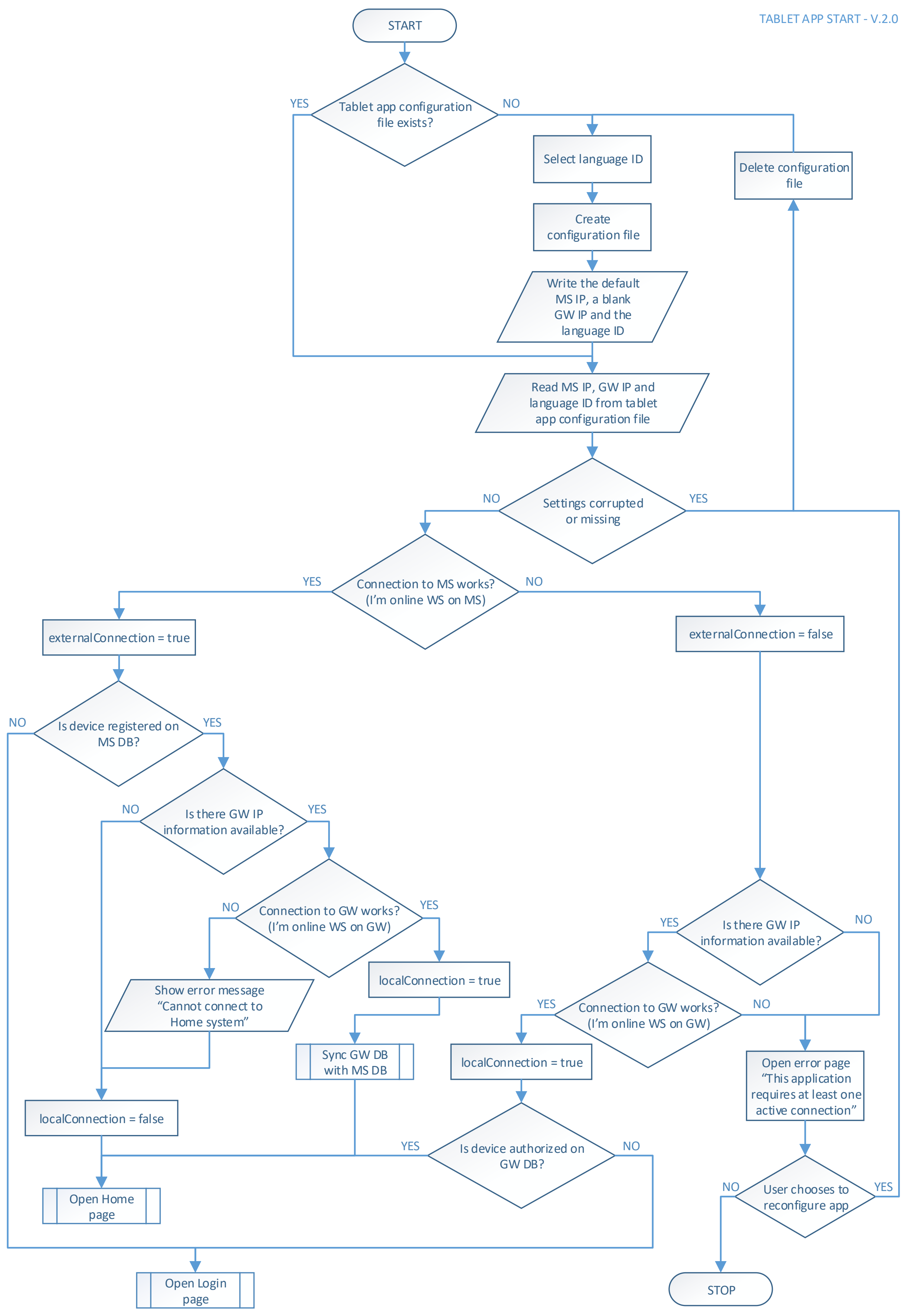

TABLET APP START - V.2.0
START
Tablet app configuration file exists?
YES
NO
Select language ID
Create configuration file
Write the default MS IP, a blank GW IP and the language ID
Delete configuration file
Read MS IP, GW IP and language ID from tablet app configuration file
Settings corrupted or missing
NO
YES
Connection to MS works? (I'm online WS on MS)
YES
NO
externalConnection = true
externalConnection = false
Is device registered on MS DB?
NO
YES
Is there GW IP information available?
NO
YES
Connection to GW works? (I'm online WS on GW)
NO
YES
Show error message "Cannot connect to Home system"
localConnection = true
Sync GW DB with MS DB
localConnection = false
Open Home page
Is there GW IP information available?
YES
NO
Connection to GW works? (I'm online WS on GW)
YES
NO
localConnection = true
Open error page "This application requires at least one active connection"
Is device authorized on GW DB?
YES
NO
User chooses to reconfigure app
NO
YES
Open Login page
STOP

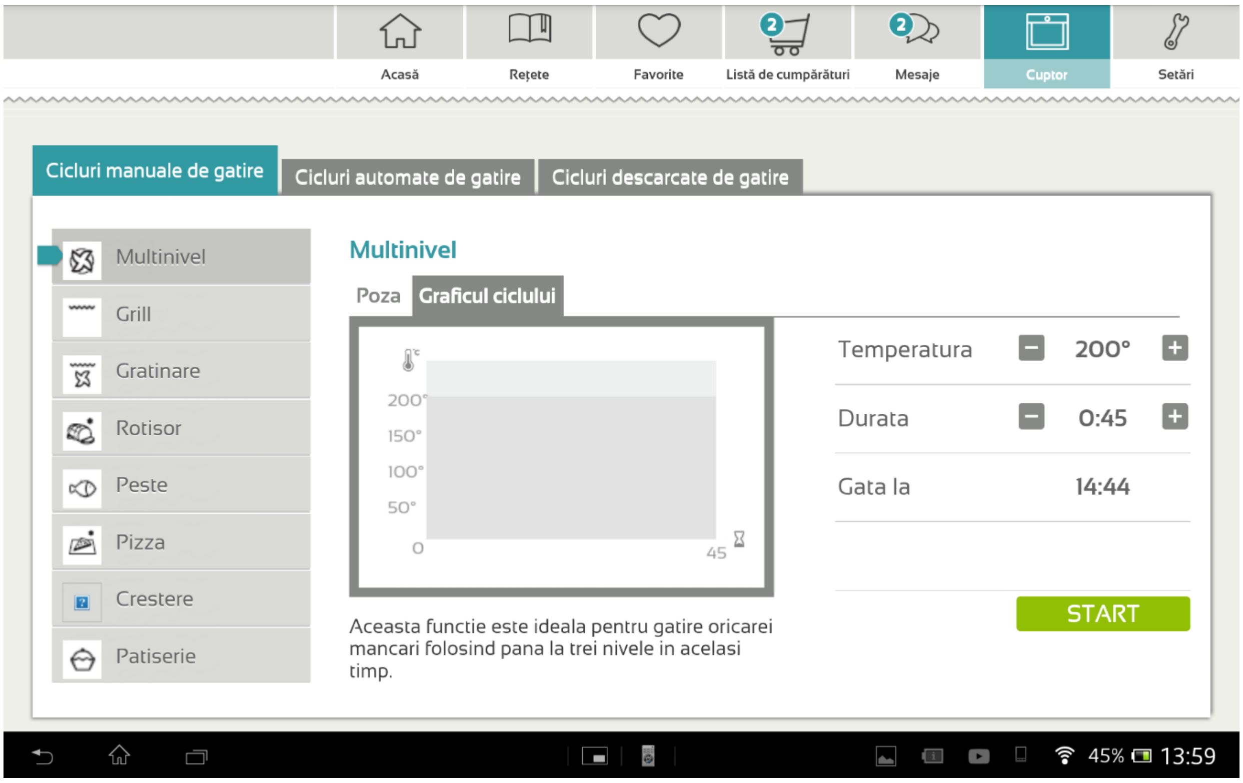
Figure 43 - Flow chart of the initialization and login of the FOOD app

6.4.1. Oven interface functionality

The "Oven" interface displays all the cooking cycles that are currently available on the oven. Cooking cycles are separated into three categories: manual, automatic and downloaded.

Users may switch between cycle types by tapping their respective tabs. Since the oven has a limited available memory, it can hold a predefined number of cooking cycles. It has been decided that the oven will hold all the manual and automatic cycles along with as many downloadable cycles as possible.

After tests and discussions with the manufacturer (Indesit), it has been determined that the ideal number of downloadable cooking cycles to be registered on the oven at any given time should be 5. As a result, users may need to delete older downloaded cooking cycles if they want the oven to be reprogrammed with newer ones. By using the AAL Food tablet application users have easy access to this functionality.



Temperatura 200°
Durata 0:45
Gata la 14:44

Aceasta functie este ideala pentru gatire oricarei mancari folosind pana la trei nivele in acelasi timp.

Figure 44 - Screenshot of the FOOD Application showing the Oven interface [57]

The application uses a JSON file which describes all the characteristics of the cooking cycles that are currently installed on the oven (names, descriptions, durations, etc.). The JSON file is split into three areas that describe the three cooking cycle categories (Figure 45).

```
"cooking_cycles":{
       "manual_cooking_cycles": [
       {
              "id": "",
              "icon": "",
            "name": "",
            "photo": "",
            "graph": [{"m":"", "t":""}, ...],
            "description": "",
            "temperature": "",
            "temperature_min": "",
            "temperature_max": "",
            "duration": ""
            }, ....
    ],
    "automatic_cooking_cycles": [
    {
            "id": "",
            "icon": "",
            "name": "",
            "photo": "",
            "graph": [{"m":"", "t":""}, ...],
            "description": "",
            "duration": "",
            "duration_min": "",
            "duration_max": "",
            "consumption": ""
    }, ....
    ],
    "downloaded_cooking_cycles": [
    {
            "id": "",
            "icon": "",
            "setting_code" : "",
            "name": "",
            "photo": "",
            "graph": [{"m":"", "t":""}, ...],
            "description": "",
            "temp_default" : "",
            "temp_min" : "",
            "temp_max" : "",
            "duration": "",
    }, ....
    ]
}
```

Figure 45 - Cooking cycles JSON file prototype

The JSON file is stored locally, on the gateway system. In the future, this will allow the user to access oven functionality even without being connected to the main Food System.

Building a new JSON file is done upon installing a new cooking cycle on the oven. The information in the JSON file is obtained by using cooking cycle characteristics from a local database. The database holds all the required information about any possible cooking cycle, registered in every one of the project's languages.

Updating the available cooking cycles on the oven is done by installing a new binary file on this smart device. The binary file stores all the required information about the cooking cycles in a way that the oven can understand. Encrypting the binary file is done through an external system, the *Indesit File Composer*. The File Composer is accessed through a C# based webservice from the local gateway machine. The local Food System uses WSDL to implement the methods required by the Indesit System. The following diagram describes the composition process that takes place on the Indesit System.

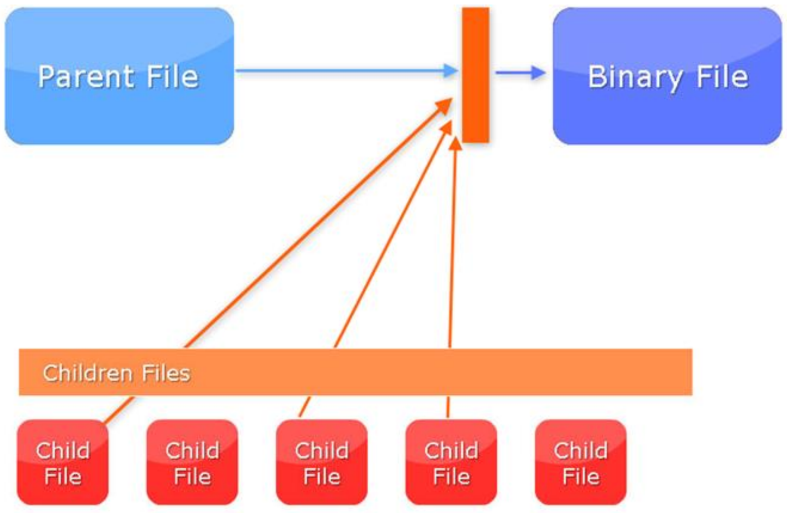


Figure 46 - Indesit file composition diagram

At the beginning of the composition process, the Food System supplies a base (parent) setting code. This code will be used to uniquely identify the parent file that needs to be used in the composition process. Along with the parent code, the Indesit System also requires an array of codes that represent each one of the child files. The final file is then the result of composing the parent file with all the child files.

The newly created binary file is then sent to the Food System as a streamed byte array which is written to a fixed location on the gateway machine. This will ensure that the oven can read it from a pre-established location and update itself, as required.

6.4.2. User interface internationalization

The FOOD UI is available in 4 languages: Romanian, Dutch, Italian and English.

Figure 47 shows the Romanian version of the application and the Diets filter along with its sub-filters. This type of filter may only be supplied for recipes added by the FOOD System.

Figure 48 and Figure 49 show pages from the Dutch and Italian versions of the application.

6.4.3. Automatic update of the tablet application

An automatic delivery of new versions of the application was developed, that requires a minimal involvement from the user. In this way, the newest features are available on all the tablets from the pilot sites.

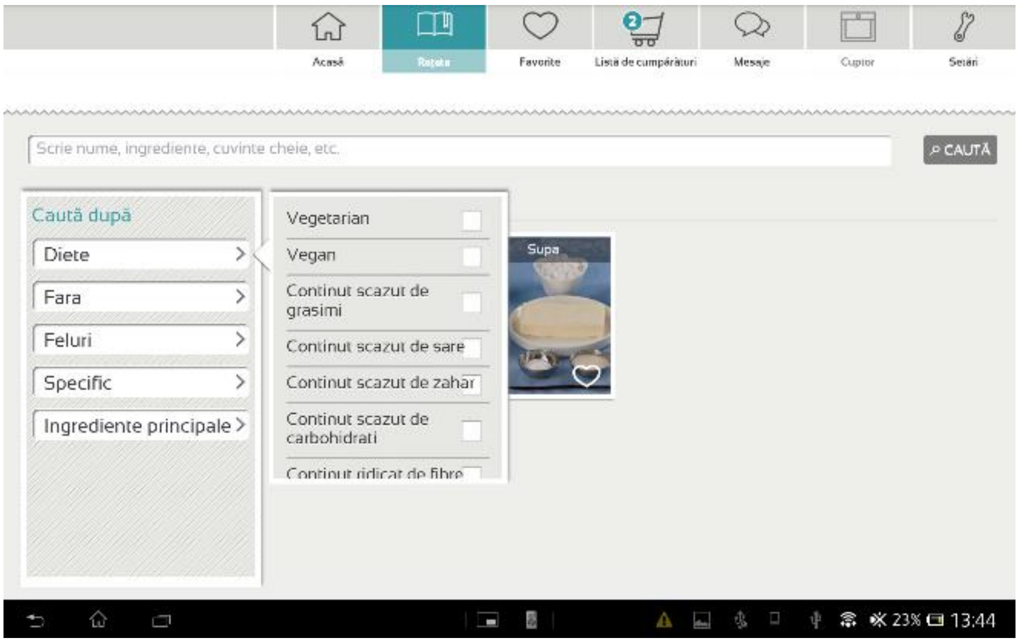


Figure 47 - FOOD UI in Romanian

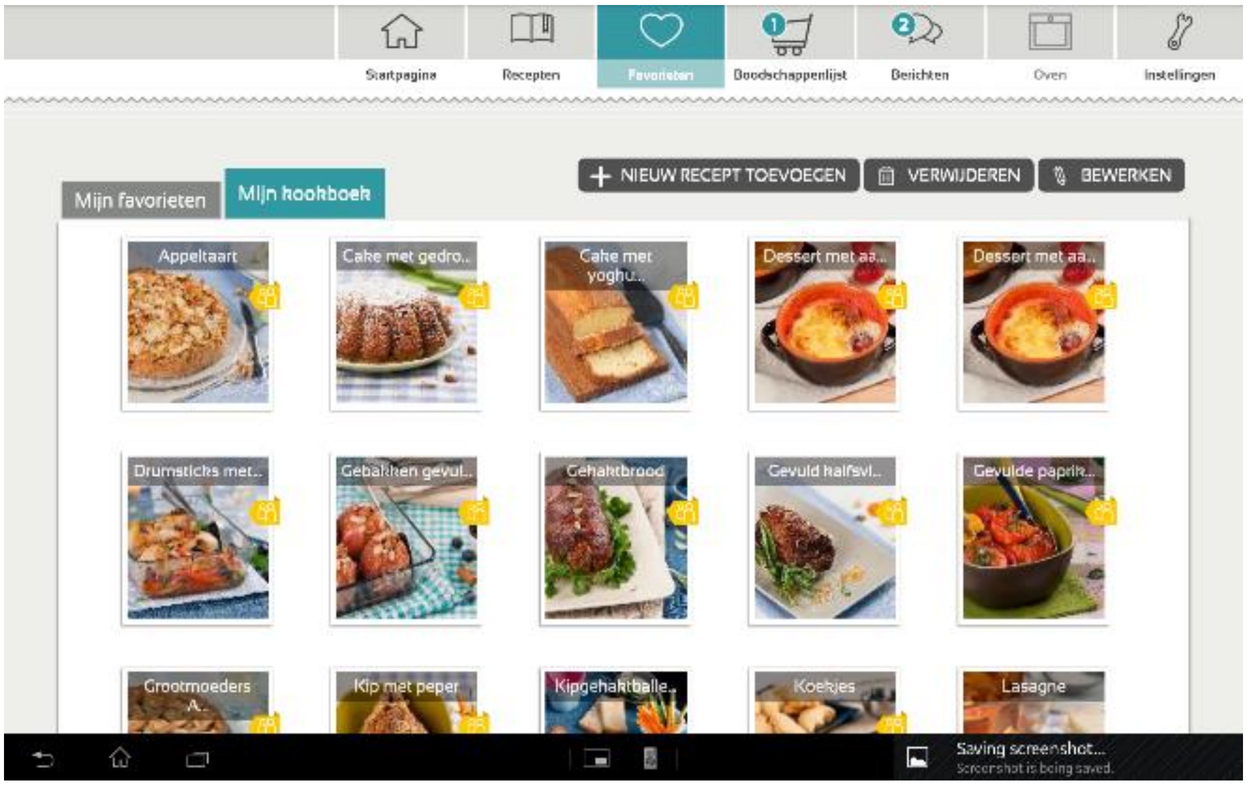


Figure 48 - FOOD UI in Dutch

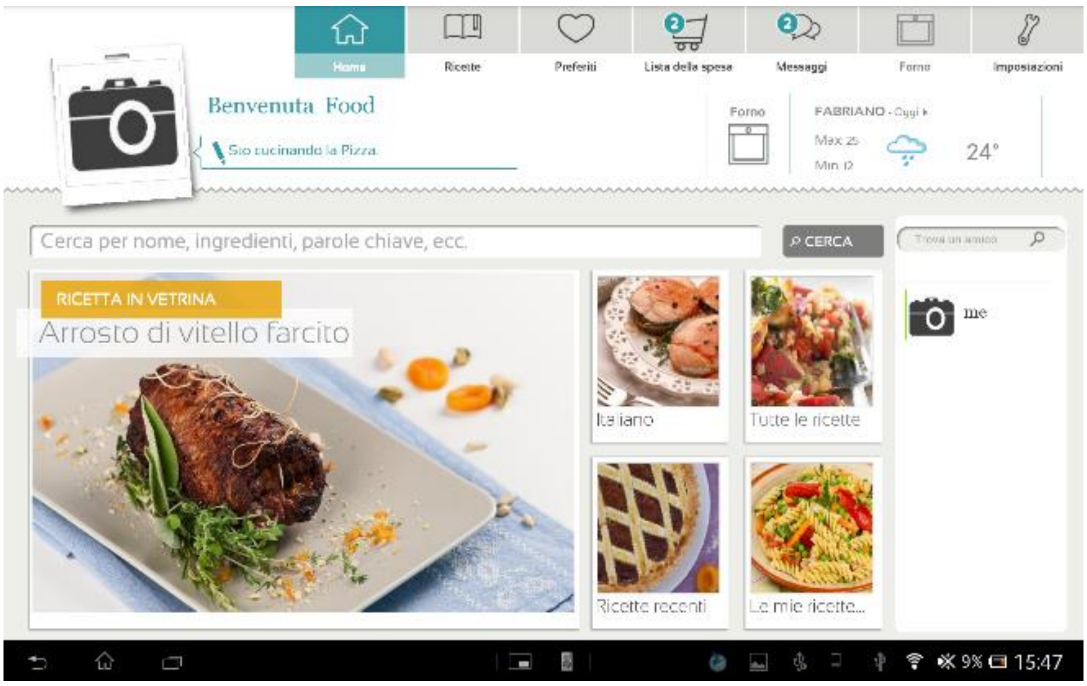


Figure 49 - FOOD UI in Italian

Every time the user accesses the Home page, the system checks for the latest application version. If it finds a new version of the application, it will begin to automatically update (Figure 50). The user may not stop the update, since a new update may introduce critical modifications that can make older versions unusable.

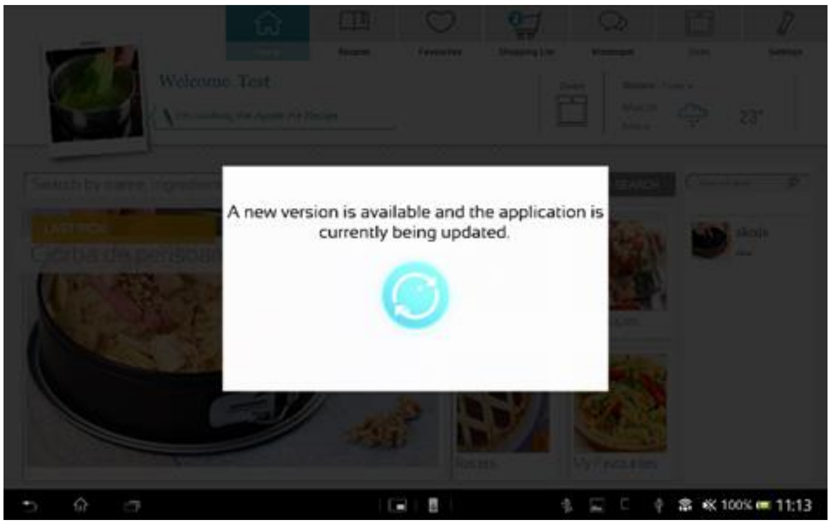

A new version is available and the application is currently being updated.

Figure 50 - FOOD App automatic update interface

6.4.4. Cross-origin resource sharing

Cross-origin resource sharing is a mechanism that allows JavaScript on a web page to make *XMLHttpRequests* to another domain, not the domain the JavaScript originated from.

Such "cross-domain" requests would otherwise be forbidden by web browsers, per the same origin security policy. CORS defines a way in which the browser and the server can interact to determine whether to allow the cross-origin request. It is more powerful than only allowing same-origin requests, but it is more secure than simply allowing all such cross-origin requests.

The CORS standard works by adding new HTTP headers that allow servers to serve resources to permitted origin domains. Browsers support these headers and enforce the restrictions they establish. Additionally, for HTTP request methods that can cause side-effects on user data (in particular, for HTTP methods other than GET, or for POST usage with certain MIME types), the specification mandates that browsers "preflight" the request, soliciting supported methods from the server with an HTTP OPTIONS request header, and then, upon "approval" from the server, sending the actual request with the actual HTTP request method. Servers can also notify clients whether "credentials" (including Cookies and HTTP Authentication data) should be sent with requests.

The webservices developed in the framework of the FOOD project use the HTTP protocol to transfer the SOAP messages, so the CORS restrictions apply. To resolve this issue, a filter was developed that scans all the requests and grants access rights to the webservices' operations [25].

A fragment of this filter is listed below. Modifying the response headers allows the filter to authorize the requests coming from AJAX calls.

```
@WebFilter(filterName = "CORSFilter", urlPatterns = {"/*"})
public class FiltruCORS implements Filter
{
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                            FilterChain chain)
                            throws IOException, ServletException
    {
        HttpServletResponse sResp = (HttpServletResponse) response;
        HttpServletRequest sReq = (HttpServletRequest)request;

        if (sResp.getHeader("Access-Control-Allow-Origin")==null)
        {
            sResp.addHeader("Access-Control-Allow-Origin", "*");
            sResp.addHeader("Access-Control-Allow-Headers", "Origin, " +
                                "X-Requested-With, Content-Type, Accept, " +
                                "Access-Control-Allow-Headers");
        }

         // ........
    }
}
```

7. Cloud Computing in AAL

Cloud computing plays an important role in enabling AAL's objectives by providing scalable computing resources, storage, and advanced data processing capabilities.

Cloud computing offers on-demand computing services - from applications to storage and processing power - over the internet on a pay-as-you-go basis. In the context of AAL, cloud services provide several advantages, including scalability, flexibility, and the ability to handle large volumes of data while ensuring high availability and reliability [58].

7.1. Applications of Cloud Computing in AAL

7.1.1. Health monitoring

Cloud computing facilitates continuous health monitoring by collecting data from various sensors and devices worn by users or embedded in their living environment. For example, smartwatches measuring heart rate and mobile apps tracking daily activities can have their data stored and analysed in the cloud. This setup allows healthcare providers to access real-time data, enabling timely interventions [59].

7.1.2. Emergency response systems

In the event of an emergency, such as a fall or medical crisis, cloud-based systems can immediately notify caregivers and emergency services, providing them with necessary data about the user's condition and location. This rapid response can be crucial in preventing severe health outcomes [60].

7.1.3. Smart home integration

Cloud computing powers smart home technologies that automate and control lighting, heating, and security systems to adapt to the needs of the elderly, thereby enhancing their comfort and safety. For example, voice-controlled systems can adjust settings in the home based on voice commands processed in the cloud [61].

7.2. Benefits of cloud computing in AAL

7.2.1. Data management and analytics

Cloud platforms offer advanced data analytics tools that help in deciphering complex patterns from the vast amount of data generated by AAL systems. These insights can lead to personalized care plans and proactive health management. *IBM Watson Health* is an example of a cloud-based analytics tool used in healthcare to analyse and interpret large datasets from various sources [62].

7.2.2. Scalability and flexibility

The cloud provides AAL systems with the ability to scale resources up or down as needed without the need for significant initial investment. This flexibility is vital for accommodating an increasing number of users or handling varying loads of data traffic [28].

7.2.3. Cost efficiency

By utilizing cloud services, AAL providers can reduce costs associated with the purchase and maintenance of hardware and software. This reduction can make AAL technologies more accessible and affordable [63].

7.3. Challenges and considerations

7.3.1. Privacy and security

Data security and privacy are paramount, as AAL systems handle sensitive personal and health information. Ensuring data encryption, secure access, and compliance with regulations like GDPR are critical challenges that must be addressed [38].

7.3.2. Reliability and connectivity

The dependency on internet connectivity makes cloud-based AAL systems vulnerable to downtimes and connectivity issues. Ensuring robust and reliable internet access is crucial, especially in remote areas [64].

7.3.3. Integration and standardization

Integrating various devices and technologies into a cohesive cloud-based system requires robust interoperability standards and protocols, which are often lacking in the rapidly evolving IoT and healthcare technology landscape [41].

7.4. Flow-based programming in IBM Cloud with Node-RED

7.4.1. Flow-based programming

Invented by J. Paul Morrison in the 1970s, flow-based programming is a way of describing an application's behaviour as a network of black-boxes, "nodes" as they are called in *Node-RED*. Each node has a well-defined purpose; it is given some data, it does something with that data and then it passes that data on. The network is responsible for the flow of data between the nodes. It is a model that lends itself very well to a visual representation and makes it more accessible to a wider range of users. If someone can break down a problem into discrete steps, they can look at a flow and get a sense of what it is doing; without having to understand the individual lines of code within each node [65].

7.4.2. Node-RED

Node-RED started life in early 2013 as a side-project by Nick O'Leary and Dave Conway-Jones of IBM's Emerging Technology Services group.

What began as a proof-of-concept for visualising and manipulating mappings between MQTT topics, quickly became a much more general tool that could be easily extended in any direction.

It was open-sourced in September 2013 and has been developed in the open ever since, culminating in it being one of the founding projects of the JS Foundation in October 2016.

Node-RED consists of a Node.js based runtime that you point a web browser at to access the flow editor. Within the browser you create your application by dragging nodes from your palette into a workspace and start to wire them together. With a single click, the application is deployed back to the runtime where it is run.

The palette of nodes can be easily extended by installing new nodes created by the community and the flows you create can be easily shared as JSON files [65].

7.5. NOAH Server cloud application

The NOAH system is supported by a server application that constitutes a Rest API and is hosted on the *IBM Cloud* platform (formerly known as *IBM Bluemix*).

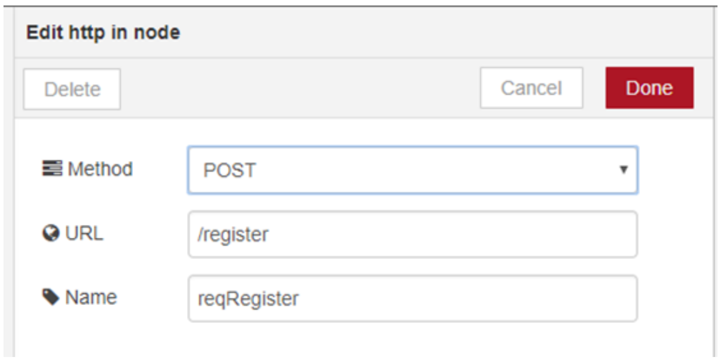
For development, the *Node-RED* tool was used, which runs a *Node.js* server and presents a visual programming environment. The programmer consists of several predefined or customized nodes, organized in **flows**.

This server application deals with data collection from sensors (using MQTT), alert generation, and serving applications for caregiver-type users and end users.

The NOAH server application is scalable; it runs in a single instance using 512MB of RAM. The application can be scaled according to needs or financial plans.

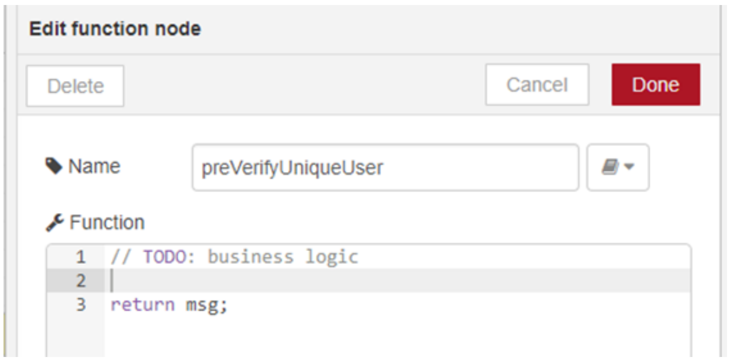
Predefined or custom nodes are used, organized across multiple flows to implement the application logic.

The graphical interface provided by *Node-RED* offers two options for configuring the nodes: one by filling in properties in predefined nodes (Figure 51), and another through the implementation of custom JavaScript code (Figure 52).



Method: POST
URL: /register
Name: reqRegister

Figure 51 - Sample of predefined Node-RED nodes



Name: preVerifyUniqueUser
Function

```
// TODO: business logic

return msg;
```

Figure 52 - Sample of custom Node-RED nodes

For the REST API, each function follows a pattern that involves an HTTP request type, a body that processes the request and generates the response, and an HTTP response type. An example is shown in Figure 53.

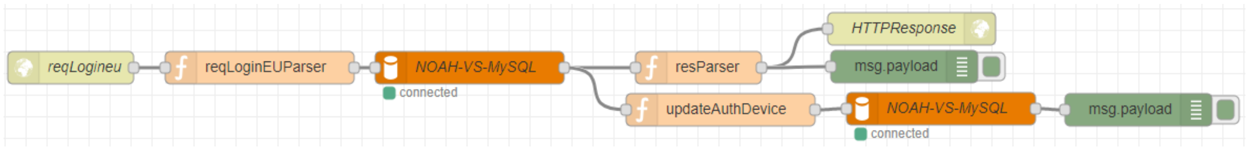


Figure 53 - Sample of Node-RED flow

7.5.1. Main Flows

The application is logically organized into several workflows, respectively different modules [66].

- **User Management flow** - encompasses the functions necessary for managing users in both mobile applications: one intended for caregiver users and the other for end-users. This flow implements the requisite functions for a caregiver user, including registration, authentication, automatic login, associating end-users under care, and modifying personal information as needed. For the end-user, functions are implemented for authentication and managing two contact points.
- **Data Processing flow** - contains the necessary functions for data processing, whether data are collected from sensors or generated by the system. This section includes functions that provide notifications, alerts, and the sensor statuses for the caregiver application. Also, from this workflow, alerts for the end-user application are formulated, as well as the latest available version of the application.
- **Data Collection flow** - includes the functions necessary for collecting information from connected devices (sensors) at pilot locations and compiling this data in a MySQL database. In other words, methods are implemented here that monitor the IoT service, which receives and transmits data from sensors. Furthermore, there is a mechanism that generates alerts in accordance with the sensors' statuses and makes them available to the mobile applications. The most recent sensor status is updated to keep track of sensor performance in case of malfunctions for various reasons.
- **Simulation and Testing flow** - contains functions for data collection and on-demand alert generation and implements facilities for the development and testing of the server application. This implies that methods are implemented to simulate input to the IoT service, reset data for certain functionalities, test database connectivity, and visualize stored data (Figure 54).

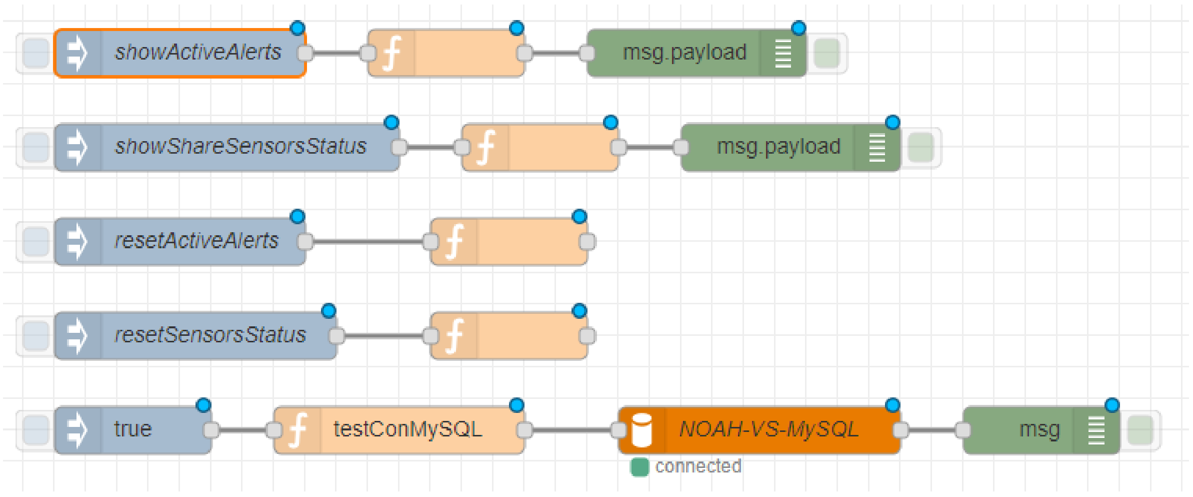


Figure 54 - Simulation and testing flow

Each flow includes a mechanism to catch all exceptions thrown and logs them in the server console.

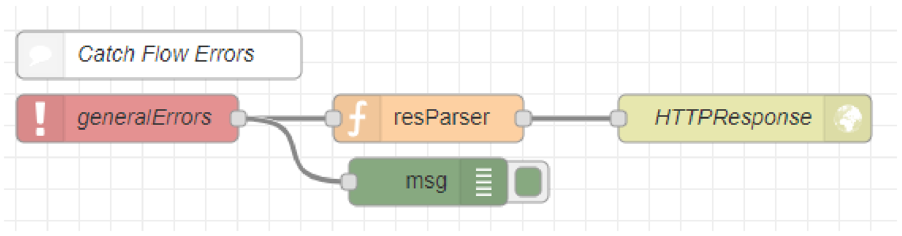


Figure 55 - Error handling flow

Regardless of its nature, data processing can have one or more of three types of results:

- **Alerts** - warnings related to technical or non-technical aspects of the devices. Alerts provide details about the status of the sensors, the connection with the system or the battery level, situations in which the sensors are in a certain position, which is unusual for a person's routine. For example, situations can be considered where the entrance door or the refrigerator door has been open for too long.
- **Notifications** - warnings related to changes in the behaviour of the monitored person. These are generated by the behavioural analysis module (BAM) integrated into the system and represent an aspect of the person's lifestyle, which needs to be analysed and intervened in an appropriate manner.
- **Statistical data** - a logging of the evolution of the data taken from the set of sensors. over a certain period, which may be relevant for an analysis performed by competent persons.

From the perspective that the system is modular and interacts with other components, it presents an input interface and an output interface.

The application's input interface consists of nodes that represent the IoT service, through which messages are received from sensors and nodes the ways in which requests are received using the HTTP protocol. The application's output interface consists of response nodes from an HTTP request and message logging nodes to the server's standard output (console).

7.5.2. Data collecting

Data collection begins with the *IBM IoT Watson* service, to which connected devices send captured information. In essence, data collection involves listening to events that are generated when IoT devices transmit a message to the system. Events are generated by sensors following a specific protocol, where a message is sent to the server when their status changes or a predetermined time interval has passed. In other words, an event is generated when a sensor is triggered, its state changes, or a certain amount of time has elapsed since the last message [67].

This rule covers important aspects of the system's operational continuity. Events are generated only in the case of changes optimizes sensor autonomy and reduces message traffic. Events generated after a delay from the last reading are treated as an “online” signal, meaning that if the pre-set period has expired, the system considers the device disconnected for some reasons.

The system's operational continuity is also ensured by compatibility with multiple versions of registered sensors. Thus, the system can capture events generated by sensors and interpret the information received from them appropriately.

All messages received by the system have the same format regardless of the type of sensor used and what it measures, and they also contain the data necessary for device authentication and authorization, namely the identifier and type of the sensor and the security code.

In all cases, the message is authorized and interpreted for the purpose of storing the values measured by sensors, which may then be transformed, if necessary, into notifications and alerts.

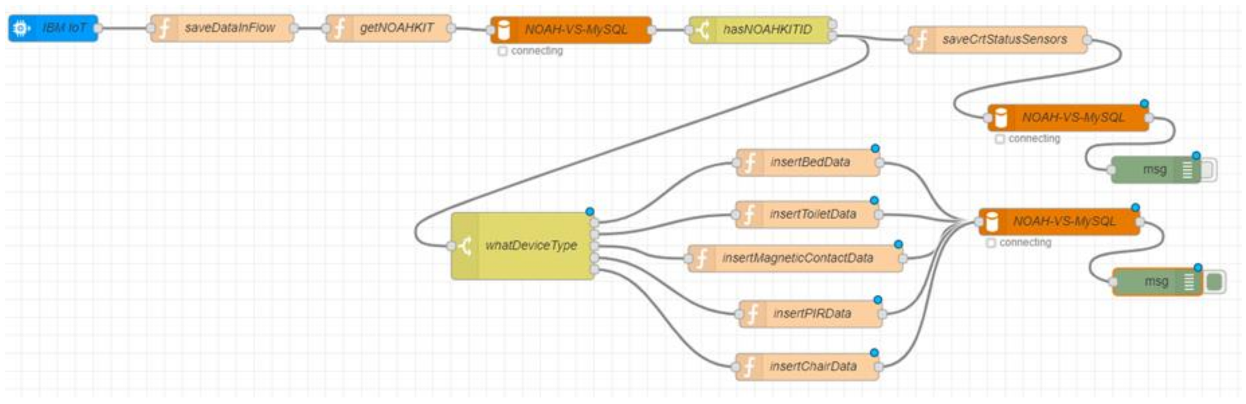


Figure 56 - Data collecting flow [19]

Firstly, when processing a message it is saved in a buffer memory area available at the processing flow level.

The sensors connected to the developed system are grouped into kits corresponding to a pilot site location. In this way, it is distinguished that each elderly person using the system has a set of sensors of different types, as specified above.

Secondly, it is checked whether the device that transmitted the current message is registered and belongs to a kit. If it is associated with a set of sensors, data processing continues; otherwise, the event is ignored.

At this point, if the scenario is complete, several operations are carried out simultaneously and asynchronously:

- The current status of the sensor is saved.

- The measurement taken is stored in the database table corresponding to the respective sensors kit.
- Alerts are generated if necessary.

To keep track of the current states of the sensors, the latest value is stored in a buffer area, available across the entire application. This is due to the need to access this information in other processes from other flows. A flow to rebuild the current status is available and it is called when the app restarts (Figure 57).

At the same time, the information from the current message is saved in the database, in the corresponding table, considering the type of device. Thus, all values measured by sensors are stored for a relevant period, which constitutes the data source used by the behavioral analysis module to provide relevant results.

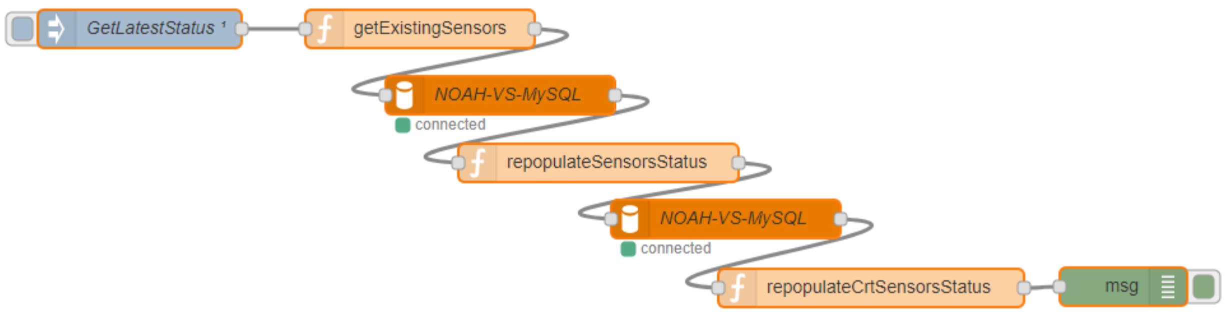


Figure 57 - Rebuilding the current status flow

Also, the necessity of generating an alert is checked. In this regard, there are two methods of generating these alerts: they can be transmitted by sensors (low battery, cable disconnection) or can be generated by the system (loss of connection with the IoT service).

Both the alerts generated directly by sensors and those generated by the system are represented in a standardized form as follows [67]:

```
{
    "alert_id": alertID,
    "user_id": userID,
    "alert_type": alertType,
    "checked": value,
    "tstamp": tstamp
}
```

7.5.3. Data processing

The flow responsible for data processing consists of methods that process data either upon arrival in the system or upon request.

When an event is generated in the *IoT Watson* service, the information received from sensors is processed for storage in the database and to construct notifications or alerts for the targeted users.

On-demand data processing is built on the basis of the HTTP request-response pattern (Figure 58). Briefly, at the time a request is made, data are processed in accordance with the desired aspect and served as a response.

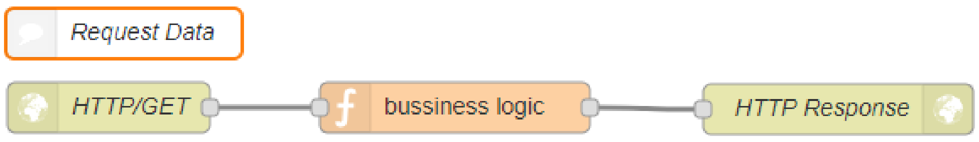


Figure 58 - Typical HTPP request-response in Node-RED [67]

All responses to HTTP requests are also standardized, thereby ensuring uniformity in interactions with other internal or external modules:

```
{
    "errorCode": 0,
    "response": { ... }
}
```

As per the example above, any response is generated over the HTTP protocol, in the form of a JSON object consisting of two properties:

- *errorCode* – The encoding is binary; the value "0" signifies that the operation was successful, while the value "1" indicates that errors occurred during the request processing.
- *response* – This is the actual message that the system returns as a response. Depending on the scenario and requirements, this property can take various forms and therefore must be appropriately handled in accordance with the requested path.

The REST API provides other modules of the application, or external ones, with resources such as notifications, alerts, sensor statuses, and unprocessed data for further processing or interpretation in the form of graphs [67].

7.5.4. Notifications

The behavioural analysis module interprets stored data and generates notifications, which are also saved in the database. The notifications are dependent on a system user.

In the example below, the structure of such a notification provided to users by the system is exemplified. It follows a standard format and contains a random number of such alerts depending on their occurrence, within a predefined period [67]:

```
[
        {
           "notification_id": 1,
           "behaviour_type": nt,
           "tstamp": ts,
           "user_id": uid
        },
        {
           "notification_id": 2,
           "behaviour_type": nt,
           "tstamp": ts,
           "user_id": uid
        }
 ]
```

The content of the response message consists of a list of objects describing the notifications. Details are divided into four properties [67]:

- *notification_id* which is a numeric identifier for the notification.
- *behavior_type* which is a numeric value representing the type of notification, i.e., a pattern found in the monitored person's behaviour or a change therein.
- *tstamp* specifies the date and time the notification was produced.
- *user_id* is the identifier of the user targeted by the notification.

7.5.5. Alerts

Like the current measurements, notifications and alerts are stored in the database, but they are also kept in a buffer area at the application level. This is because alerts are generated in another processing flow, and through the volatile memory area, they are accessed much more quickly, and also because alerts have a validity term.

There are two different scenarios in which alerts are provided, but in both cases, the result is similar. Alerts can be provided depending on the desired number of them and can be accessed in different ways. If only one alert is desired, it is provided immediately from the application's buffer area, but if multiple alerts are desired simultaneously, they are retrieved from the database in a limited number based on a pre-established period that assumes the alerts are active.

In the example below, the standard model according to which alerts are constructed is illustrated [67].

```
[
    {
          "alert_id": alertID,
          "user_id": userID,
          "alert_type": alertType,
          "checked": value,
          "tstamp": tstamp
```

```
        },{
              "alert_id": alertID,
              "user_id": userID,
              "alert_type": alertType,
              "checked": value,
              "tstamp": tstamp
        }
    ]
```

The content of the message consists of a list of alert type objects which are described by:

- *alert_id* – the unique identifier of the respective alert.
- *user_id* – the identifier of the user targeted by the produced alert.
- *alert_type* – type of alert, describing the reason for its generation.
- *checked* – validator for acknowledging it.
- *tstamp* – the date and time at which it was generated.

7.5.6. Simulation and testing

The system's server application also presents a testing and simulation mechanism to facilitate the development process and to identify potential problems. Each type of communication between devices and the system has testing methods for developers and simulation but presenting different implementation in accordance with the requirements.

This mechanism targets predominantly technical aspects, offering methods to verify the communication between connected sensors and the data collection system, the storage of information in a persistent database in a set format, the processing of collected data to generate alerts and notifications, etc.

For verifying and simulating the connection to the system's IoT service, templates are used. In addition, nodes are used that offer the possibility to inject certain messages at the developer's request or at configurable periods of time.

In Figure 59 the method for verifying the data collection process recorded by sensors is presented.

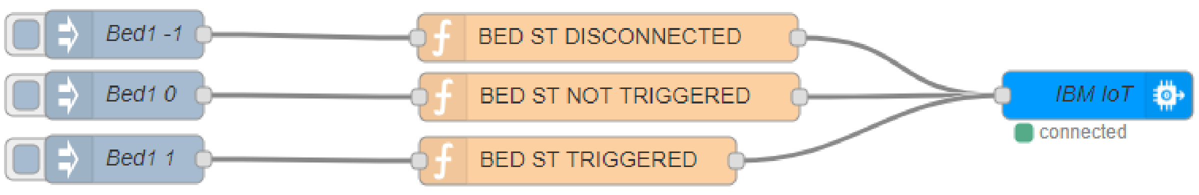


Figure 59 - Sensor status testing flow

In this case, the three states in which a sensor can be are verified, namely: connected/disconnected, triggered or untriggered. Similarly, the battery level can also be tested by assigning values to the corresponding property, which are on either side of a pre-established threshold.

On the other hand, the logic of generating alerts can be tested. The following listing includes the method for generating alerts, which is also implemented on the main flow but fed with constant data.

Once the data collection events and the generation of alerts have been tested, they can be visualized by developers and, if necessary, be reset (Figure 60) [67].

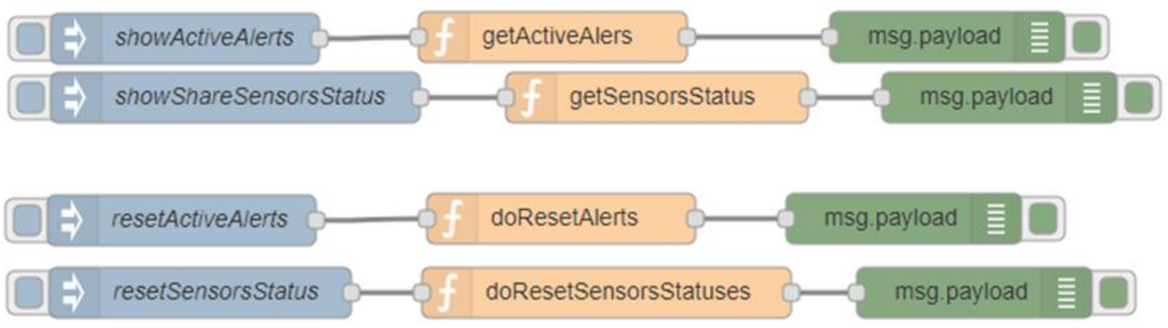


Figure 60 - Viewing and resetting alerts and statuses

7.5.7. Communication with sensors

Messages are sent to an agent via the MQTT protocol which involves a level of protection through SSL security certificates and a level of authentication through a username and password.

The format of messages received by the system is JSON and has the following structure:

```
{
    "sid": <text>,
    "rtc_send" : <number>,
    "battery" : <number>,
    "msg_type" : <text>,
    "payload" :
    {}
}
```

- *sid* – is the device identifier and consists of two characters representing the type of sensor and 12 characters representing the MAC address.
- *rtc_send* – is the numeric value of the timestamp of the message sending and represents the number of seconds elapsed since the date of 1/1/2000 00:00:00.
- battery – is the numeric value, in mV, indicating the battery level at the time of sending the message.
- *msg_type* – is a key term that differentiates data transmission messages from alert messages.

- *payload* – is the content of the message and has a different structure depending on its type. It may contain a variable list of pairs of numerical values, for the state of the sensor and the timestamp of the measurement or a variable list of parameters that describe the generated alert.

Messages are transmitted once an hour, which means that each message timestamp must be interpreted, thus there are two cases:

- If the timestamp value of the message falls within a certain error margin from the server's timestamp at the time of reception, then it is considered to have been transmitted correctly and is used as such.
- If the timestamp value exceeds the error margin relative to the server's, the timestamp is considered relative and must be interpreted. Thus, the gap is calculated by the difference between the message sending date and the receiving date, which is then subtracted from the server date to obtain the exact moment of measurement.

(B-ii) The evolution and development plans for career development

Didactic activity

The author started his academic career in 2005 in the Automation Department of Transilvania University of Brasov (part of the Electrical Engineering and Computer Science Faculty), right after graduating the *Automation and Industrial Informatics* program study at the same university. He continued with a MSc program at the same university – *Information and Communication Systems and Technologies*, which he graduated in 2007. He was awarded the PhD title in 2011.

The author's academic interests gravitate towards the Information Technology field, which is highlighted by the courses he supports in the Department. At present, the author gives lectures for the following courses:

- *Computer Programming and Programming Languages – 2nd Module (Java Programming Language)* – 1st year for AIA[2], TI[3] and RO[4] program studies
- *Object Oriented Programming (with applications in C# language)* – 2nd year for AIA and TI students
- *Data acquisition and processing* – 3rd year TI
- *Cryptography* – 4th year TI
- *Video signal capture and image processing* – 1st year of the SAATI[5] MSc program
- *Fundamentals of Cryptography in Applied Scenarios* – 1st year of the CS[6] MSc program (in English)
- *Web application – 2nd Module* – 2nd year of the TIN[7] MSc program (in English)

Also, he serves the practical applications activities for:

- *Data structures and Algorithms* – 2nd year for TI students
- *Object Oriented Programming (with applications in C# language)* – 2nd year for AIA and TI students
- *Web programming* – 3rd year for TI program study
- *Cryptography* – 4th year TI
- *Video signal capture and image processing* – 1st year of the SAATI[8] MSc program

[2] *AIA – Automation and Applied Informatics*
[3] *TI – Information Technology*
[4] *RO – Robotics*
[5] *SAATI – Advanced systems in automation and information technology*
[6] *CS – Cyber Security*
[7] *TIN – Internet Technologies*
[8] *SAATI – Advanced systems in automation and information technology*

- *Web application – 2nd Module* – 2nd year of the TIN[9] MSc program (in English)

In the past, the author also supported other disciplines, like: *Computer Programming and Programming Languages – 1nd Module (Programming in C/C++)* (both in Romanian and English), *Graphic processing*, *Reliability and diagnosis*, *Information security* and *Integrated Software Systems.*

The author published 5 volumes to support these teaching subjects. The volumes are updated periodically, to keep up with the technological changes and available tools.

To support actively the evolution and of education and to help increase the quality of teaching the author participated in several POSDRU projects:

- *Stagii moderne de practică în domeniul electrotehnic*, 2009 - 2012
- *FlexFORM - program de formare profesională flexibilă pe platforme mecatronice,* 2010 - 2013
- *Învață automatică*, 2010 – 2013
- *Program Strategic pentru Promovarea Inovării în Servicii prin Educație Deschisă, Continuă* (INSEED), 2010 – 2013

In his career, the author coordinated over 100 Diploma projects and Dissertation thesis with subjects from the fields of Information Technology and Automation.

The author has always been supporting the students and has been involved in their academic endeavours, encouraging them to expand their knowledge beyond the subjects treated in the typical courses and to participate in conferences or competitions, such as the *Student Scientific Circles Session* that is organized every year by the University (the author coordinated tens of projects in this event). Also, the author coordinated teams of students for internal projects and competitions, such as AFCO[10] or "My Faculty". In 2017, a team coordinated by the author won 2nd place in the 2nd edition of the "Java Competition for Universities" organized by ADFABER and sponsored by Oracle Academy.

Some of the research efforts of the author were directed to subjects related to education (such as tools and methods for vocational education or ITC tools for a more efficient education process).

The author will continue to support students' activities and initiatives in the future and maintain a truthful relationship with them, to help them achieve their potential. In the near future the author plans to organize a club for students with passion for the IT domain (*IT Guild*).

To the extent that legal and economic circumstances allow, and provided that the author's performance and results meet the criteria for advancement in the academic hierarchy, he will pursue this path.

[9] *TIN – Internet Technologies*
[10] *AFCO – Absolvenții în fața companiilor*

Research activity

The author has been involved in international and national research projects since 2005, as a member or as the coordinator for the Romanian partner.

In 2006 he enrolled in the PhD program offered by Transilvania University of Brasov. In 2011 he was awarded the title of PhD in the *Electrical Engineering* domain, based on the thesis "Contributions on the unified treatment of capture equipment in video surveillance systems".

All the author's research efforts have been focused on the Technology Information field. Both the PhD thesis and the author's publications reflect his attraction to this dynamic field, with a particular interest in distributed applications and the use of programming languages and modern technologies for creating platforms that can serve as the basis for complex systems, useful in multiple fields and for integrating different systems.

The author published over 50 research papers, with more than 70 citations, covering broad subjects like software architecture, AAL systems, IoT, complex energy monitoring system, green energy, ICT tools for e-learning, smart houses:

- 2 book chapters
- 23 articles and conference papers, included in WoS (4 in Q1 and Q2)
- 13 conference papers indexed in recognized international databases (such as Scopus or IEEE)
- 15 conference papers indexed in other databases

The author has been collaborating since 2005 with several entities in international projects coordinated by the author (for the Romanian partner), such as:

- *Framework for Optimizing the prOcess of FeeDing (FOOD)*, AAL Program, 2011 – 2014
- *Not Alone at Home* (NOAH), AAL Program, 2016 – 2020
- *SAfety of elderly people and Vicinity Ensuring* (SAVE), AAL Program, 2019 – 2023
- *Increase sElf Management and counteract social IsoLatIOn using a vocal assistant enabled virtual concierge* (EMILIO), AAL Program, 2022-present

Also, the author was involved, as member of the team, in many other international research projects, like:

- *DB2IMS – An Information Management System Increasing Reliability in Data Transfer Using XML Technology*, IBM Faculty Awards Cycle 3 CEMA , 2005
- *Portale della Conoscenza per Mobile Object Learning strutturate* (MOLECOLE), LdV Program, 2005 – 2007

- *Vocational Education Training to PROFessionals for a European Solid Space of collaboratION And Learning* (VET PROF.E.S.S.ION.A.L.), LdV Program, 2005-2007
- *ELMSET – Using Eclipse to Develop a Learning Management System, a SCO Editor, and a Test Tool for Evaluation SCOs' Compatibility with SCORM Standards*, IBM Innovation Award, 2006
- *Individualized Learning Enhanced By Virtual Reality* (IDENTITY), Minerva Program, 2006 – 2008
- *Organizazione delle ceRtificazioni con Smart cArd nei Mestleri e nelle inNOvazioni del maRE* (O.R.S.A. MI.NO.RE.), LdV Program, 2006 – 2008
- *Valorization of an experiment-based training system through a transnational education network development* (VETTREND), LdV Program, 2006 – 2008
- *Testing and Implementing EQF - and ECVET-Principles in Trade Organizations and Education* (TipToe) – LdV Program, 2008 – 2010
- *WEEGEN- Smarter Buildings: Intelligent Distributed Workspace for Energy Efficiency in the GENIUS Campus*, Share University Research – IBM, 2009 – 2011
- *NEW employees Development And Learning: technological methods and tools in favour of the professional development of new employees* (NEW DEAL) – LdV program, 2012 – 2014
- *Using IBM CloudBurst and rational Application Developer toDevelop Mobile Applications for Remote Healthcare Monitoring with Feedback Functions MHMON*, IBM Faculty Award, 2012
- *Healthy Life support through Comprehensive Tracking of individual and Environmental Behaviors* (HELICOPTER), 2013 - 2016

The author was also involved in national grants/contracts, as a team member in:

- *Tehnici și tehnologii de realitate virtuală aplicate în inginerie, medicină și artă* (TRIMA), CNCSIS, 2006 – 2008
- *Sistem Integrat de Gestiune, control al accesului și de comUnicare intranet și inteRnet în căminele studențești – S.I.G.U.R extins - Complex Memorandului și Complexul Colina Universității* - 2006 – 2007
- *Sistem complex pentru securitatea fizica a PRO-DD (alarma intruziuni, incendiu, control acces si supraveghere video)*, 2009 – 2011
- *Soluții GREEN pentru managementul energiei din Centrul de Date în contextul funcționării unui sistem distribuit de management al resurselor și documentelor din PRODD. Proiect pilot pentru Mini Centrul de Date care deservește echipele de proiectare și management ale PRO-DD*, 2009 – 2010
- *Program strategic de CD pentru creștere și inovare în domeniul serviciilor* - CRIS, 2010 – 2011
- *Realizarea unei strategii pentru implementarea unui pilot de rețea electrica inteligentă în cadrul SC ELECTRICA SA* , SC ELECTRICA SA, 2011 – 2012

- *Creșterea competitivității economiei românești prin cercetare, dezvoltare și inovare - Transfer la operatorul economic- Măsurarea consolidată și transmiterea parametrilor energetici spre punctele de colectare* (CON-INTEL), 2016 – 2018
- *Cercetări privind identificarea și proiectarea soluțiilor optime de modernizare a atelierelor de Vopsitorie, Mase plastice si a instalațiilor de compensare cu aer proaspăt preîncălzit la atelierul Tratamente de suprafață*, 2017
- *Cercetări privind testarea sistemelor de calcul și de comunicație*, 2017

He has been also involved in organizing international conferences, such as:

- *The 11th International Conference on Optimization of Electrical and Electronic Equipment* OPTIM'08, 2008
- *The 12th International Conference on Optimization of Electrical and Electronic Equipment* OPTIM'10, 2010
- *The 6th Conference on Speech Technology and Human-Computer Dialogue* (SpeD), 2011
- *The 13th International Conference on Optimization of Electrical and Electronic Equipment* OPTIM'12, 2012
- *The 14th International Conference on Optimization of Electrical and Electronic Equipment* OPTIM'14, 2014
- *New Trends on Sensing - Monitoring - Telediagnosis for Life Sciences*, 2015
- *10th International Conference on Photoexcited Processes and Applications*, 2016
- *2nd International Conference on Nuclear Photonics*, 2018

The author has been taking part in the coordinating team for several PhD students:

Nr. crt.	**Doctorand / Doctor**	**Titlul tezei**	**Teze finalizate (TF) sau în stagiu (TS)**
1	Costin GRIGORESCU	Cercetări privind monitorizarea și controlul consumatorilor energetice	TF - 2012
2	Florian NEUKART	System Applying High Order Computational Intelligence in Data Mining and Quantum Computational Considerations Concerning the Future of Artificial Intelligence Inteligența computațională în data mining și calculul cuantic, precum și considerații privind viitorul inteligenței artificiale	TF - 2013
3	Peter SZAKACS-SIMON	Monitorizarea persoanelor intr-un mediu inteligent	TF - 2013
4	MUHAMMAD Manar Ahmad Sabry	Application of Fuzzy Logic in Electrical Power Network, Industry and Safety	TF - 2013

	Mahmoud Saeed Ahmad Salih	Aplicarea logicii FUZZY în rețelele electrice de distribuție, industrie și securitatea muncii	
5	Valentin GHIȘA	Cercetări privind modalități de echilibrare și optimizare a transferului informațional	TF - 2015
6	Milian BADEA	Cercetări privind sisteme de comutare și poziționare pentru surse de energie regenerabilă - 2013	TF - 2018
7	Florin OGIGĂU-NEAMȚIU	Cercetări privind securitatea informației în sistemele cloud computing	TF - 2018
8	Adrian MANEA	Cercetări privind securitatea datelor în sistemele informatice	TF - 2018
9	Ligia Georgeta GUȘEILĂ	Integrarea serviciilor de tip cloud computing în centrele de prelucrare a informației	TF - 2019
10	Dragoș Vasile BRATU	Inteligența Artificială i ̂n sprijinul persoanelor cu nevoi speciale	TF - 2023
11	Ovidiu PĂSCUȚOIU	Cercetări privind performanța securității rețelelor	TS
12	Liviu Doru DOGAR	Provocări de securitate cibernetică la gestionarea în cloud a înregistrărilor video	TS
13	Cristian VIZITIU	Abordări sistemice ale etapelor ciclului de viață al soluțiilor integrate și/sau conectate în domeniul performanței umane	TS
14	Denis SINANAJ	User Interaction for device software update	TS
15	Maria Alexandra ZOLYA	Inteligența Artificială în domeniul medical	TS
16	Edel Abreu HERNANDEZ	Scholarly Knowledge Graph relation extraction	TS
17	Reyder Cruz de la OSA	Improvements of the associative classifier accuracy	TS
18	Asday Savon BERENGUER	Systems for Storage and Interhospital Transmission of Medical Images	TS

The author can outline several directions of interest that he is intending to pursue in the future:

- Using information and communication technologies (ICT) to optimize the education process (e.g. using gamification to stir the interest of the novel student)
- Software platforms for distributed computing (e.g. developing and assessing a software platform designed to improve the efficiency and scalability of distributed computing tasks in various scientific domains.)
- Hardware and software systems to assist elderly or disabled people (e.g. creating and evaluating a series of integrated hardware and software solutions tailored to assist disabled or elderly individuals in their daily activities.)
- Industrial process automation applications (e.g. investigate the implementation of IoT and real-time data analytics to enhance automation and efficiency in manufacturing processes.)
- The use of Machine Learning in tools for day-to-day use (e.g. developing machine learning models that can be integrated into daily use tools such as personal assistants, recommendation systems, and health monitors.)

Also, one of the main concerns in the future will be to raise funds for the research activities, by applying for grants, searching for contracts with local and national industrial partners, and participating in project competitions.

The results of these studies, that align with his personal interests and those of the department. will be published in articles and papers, particularly targeting top rated journals (mainly from Q1 and Q2).

Support and administrative activities

Besides the didactic and research activities, the author has been actively involved in the Department's, Faculty's, and University's day-to-day activities. He is responsible for coordinating the team in charge of organizing the schedule of the Electrical Engineering and Computer Science Faculty, which thought him the importance of maintaining an atmosphere of understanding and balance among the members of a group to achieve positive outcomes. The author is also in charge of updating the Faculty's website and official *Facebook* page (created together with the students).

He also has been involved in the accreditation and reaccreditations processes for the Technology Information study program.

The author is a member of the *Permanent Admission Office* for the Faculty, member of the *Bachelor's Degree Committees* for the TI and AIA study programs, the *Quality Assurance Committee* of the Department, member of the *GDPR committee* of the Faculty. From 2024 the author is member in the *Faculty Council* (previously was an alternate member).

The author will continue to be involved in all activities of the Department, Faculty and University.

(B-iii) Bibliography

[1] L. Lamport, "The Part-Time Parliament," in *ACM Transactions on Computer Systems*, 1998.

[2] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *USENIX Annual Technical Conference*, 2014.

[3] C. Cachin, R. Guerraoui and L. Rodrigues, in *Introduction to Reliable and Secure Distributed Programming*, Springer, 2011.

[4] A. Tanenbaum and M. Van Steen, in *Distributed Systems: Principles and Paradigms.*, Pearson, 2017.

[5] G. Coulouris, J. Dollimore, T. Kindberg and G. Blair, in *Distributed Systems: Concepts and Design*, Addison-Wesley, 2011.

[6] W. Stallings, in *Cryptography and Network Security: Principles and Practice*, Pearson Education, 2006.

[7] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, 2005.

[8] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2015.

[9] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional, 2003.

[10] N. Narkhede, G. Shapira and T. Palino, Kafka: The Definitive Guide, O'Reilly Media, 2017.

[11] D. Agrawal, S. Das and A. E. Abbadi, "Big data and cloud computing: Current state and future opportunities," in *Proceedings of the 14th International Conference on Extending Database Technology*, 2012.

[12] P. J. Sadalage and M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley, 2012.

[13] B. Nedelcu, Beginner's Guide to NGINX: Application Deployment and Security, Packt Publishing, 2019.

[14] J. Turnbull, The Docker Book: Containerization is the new virtualization, James Turnbull, 2016.

[15] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446," 2018.

[16] B. Beyer, C. Jones, J. Petoff and N. R. Murphy, "Site Reliability Engineering: How Google Runs Production Systems," O'Reilly Media, 2016.

[17] FOOD Consortium, "AAL FOOD Project," [Online]. Available: https://www.aal-europe.eu/projects/food/. [Accessed 29 04 2024].

[18] NOAH Consortium, "NOAH Project," [Online]. Available: https://www.aal-europe.eu/projects/noah/. [Accessed 29 04 2024].

[19] S. Moraru, A. Mosoi, D. D.M. Kristaly, M. I., V. Petre, D. Ungureanu, L. R. D. Perniu and M. Cocuz, "Using IoT assistive technologies for older people non-invasive monitoring and living support in their homes," *International Journal Of Environmental Research And Public Health,* vol. 19, no. 10, 2022.

[20] S. Moraru, L. Perniu, D. Ungureanu, A. Mosoi, D. Kristaly, F. Sandu and A. Manea, "Home assisted living of elderly people using wireless sensors networks in a cloud system," in *International Symposium in Sensing and Instrumentation in IoT Era (ISSI)*, Shanghai, People's Republic of China, 2018.

[21] HELICOPTER Consortium, "HELICOPTER AAL project," [Online]. Available: https://www.aal-europe.eu/projects/helicopter/. [Accessed 29 04 2024].

[22] SAVE Consortium, "SAVE AAL Project," [Online]. Available: https://save-aal.eu/en/. [Accessed 29 04 2024].

[23] M. Papazoglou, Web Services: Principles and Technology, Pearson Education, 2008.

[24] G. Alonso, F. Casati, H. Kuno and V. Machiraju, Web Services: Concepts, Architectures and Applications, Springer-Verlag, 2004.

[25] D. Kristaly, S. Moraru, F. Neamtiu and D. Ungureanu, "Assistive monitoring system inside a smart house," in *International Symposium in Sensing and Instrumentation in IoT Era (ISSI)*, Shanghai, People's Republic of China, 2018.

[26] NYX Wolves, "Implementing Micro Frontend Architecture in the Web App," [Online]. Available: https://nyxwolves.com/implementing-micro-frontend-architecture-in-the-web-app. [Accessed 02 05 2024].

[27] D. Kristaly and S. Moraru, "An incorporated solution to support elder people in staying in their familiar surroundings," in *International Congress on Information and Communication Technology (ICICT)*, London, United Kingdom, 2022.

[28] J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems,* vol. 29, no. 7, pp. 1645-1660, 2013.

[29] A. Botta, W. de Donato, V. Persico and A. Pescapé, "Integration of Cloud computing and Internet of Things: A survey," *Future Generation Computer Systems,* vol. 56, pp. 684-700, 2016.

[30] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: A survey," *Computer Networks,* vol. 54, no. 15, pp. 2688-2710, 2010.

[31] C. Lord, J. H. Colvin and A. Mihailidis, "An analysis of the technology acceptance model in understanding university students' behavioral intention to use e-learning," *Educational Technology & Society,* vol. 21, no. 3, pp. 25-47, 2018.

[32] D. J. Cook, J. C. Augusto and V. R. Jakkula, "Ambient intelligence: Technologies, applications, and opportunities," *Pervasive and Mobile Computing,* vol. 5, no. 4, pp. 277-298, 2009.

[33] S. Patel, H. Park, P. Bonato, L. Chan and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of NeuroEngineering and Rehabilitation,* vol. 9, no. 1, p. 21, 2012.

[34] D. J. Cook, J. C. Augusto and V. R. Jakkula, "Ambient intelligence: Technologies, applications, and opportunities," *Pervasive and Mobile Computing,* vol. 5, no. 4, pp. 277-298, 2009.

[35] B. Allen, K. Dresner and H. Wallach, "Surveillance and Security. Technological Challenges," vol. 12, pp. 445-456, 2005.

[36] G. Lopez, L. Quesada and L. A. Guerrero, "Ubiquitous computing: Applications, challenges and future trends," *Smart Computing Review,* vol. 1, no. 4, pp. 299-308, 2011.

[37] V. C. Gungor and G. P. Hancke, "Industrial wireless sensor networks: Challenges, design principles, and technical approaches," *IEEE Transactions on Industrial Electronics,* vol. 56, no. 10, pp. 4258-4265, 2009.

[38] R. H. Weber, "Internet of Things – New security and privacy challenges," *Computer Law & Security Review,* vol. 26, no. 1, pp. 23-30, 2010.

[39] B. Mittelstadt, "Ethics of the Health-Related Internet of Things: A narrative review," *Ethics and Information Technology,* vol. 19, no. 3, pp. 157-175, 2017.

[40] L. D. Xu, W. He and S. Li, "Internet of Things in industries: A survey," *IEEE Transactions on Industrial Informatics,* vol. 10, no. 4, pp. 2233-2243, 2014.

[41] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal,* vol. 1, no. 1, pp. 3-9, 2014.

[42] R. Khan, S. U. Khan, R. Zaheer and S. Khan, "Future internet: The internet of things architecture, possible applications and key challenges," *Frontiers in Artificial Intelligence and Applications,* vol. 10, no. 2, pp. 256-270, 2012.

[43] M. Zeng, P. H. Pathak and P. Mohapatra, "Analyzing shopper's behavior through WiFi signals," in *Proceedings of the 2nd workshop on Workshop on Physical Analytics*, 2017.

[44] V. Rialle, C. llivet, C. Guigui and C. Hervé, "What do family caregivers of Alzheimer's disease patients desire in smart home technologies?," *Gerontechnology,* vol. 7, no. 2, pp. 96-100, 2008.

[45] C. Grigorescu, S. Moraru, D. Kristaly and M. Badea, "DB4Objects based buffering application for use in software monitoring systems," in *International Danube-Adria-Association-for-Automation-and-Manufacturing Symposium (DAAM)*, 2011.

[46] V. Stara, M. Rampioni, A. Mosoi, D. Kristaly, S. Moraru, L. Paciaroni, S. Paolini, A. Raccichini, E. Felici, L. Rossi, C. Vizitiu, A. Nistorescu, M. Marin, G. Tonay, A. Toth, T. Pilissy and G. Fazekas, "A technology-based intervention to support older adults in living independently: protocol for a cross-national feasibility pilot," *International Journal Of Environmental Research And Public Health,* vol. 19, no. 24, 2022.

[47] E. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* 1970.

[48] C. Date, An Introduction to Database Systems, Addison-Wesley, 2004.

[49] T. Härder and E. Rahm, "Database Systems for New Applications," *ACM Computing Surveys,* 2012.

[50] M. Stonebraker and R. Cattell, "10 Rules for Scalable Performance in 'Simple Operation' Datastores," *Communications of the ACM,* 2012.

[51] S. Moraru, A. Moșoi, D. Kristaly, F. Sandu, D. Floroian, D. Ungureanu and L. Perniu, "“Save” - An integrated approach of personal and home safety for active assisted living," in *IFIP WG*

12.5 International Conference on Artificial Intelligence Applications and Innovations (AIAI), Hersonissos, Greece, 2021.

[52] C. Owsley, R. Sekuler and S. D., "Contrast sensitivity throughout adulthood," *Vision Research,* vol. 23, no. 7, pp. 689-699, 1983.

[53] J. Botwinick and M. Storandt, Memory, related functions and age, 1974.

[54] J. Berkowitz and S. Casali, "Influence of Age on the Ability to Hear Telephone Ringers of Different Spectral Content," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting,* vol. 34, no. 2, pp. 132-136, 1990.

[55] M. Vercruyssen, "Aging and Technology: A Developmental View," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting,* vol. 40, no. 3, pp. 138-140, 1996.

[56] L. Burzagli, L. Di Fonzo, P. Emiliani, L. Boffi, J. Bak, C. Arvidsson, D. Kristaly, L. Arteconi, G. Matrella, I. De Munari and P. Ciampolini, "The FOOD project: Interacting with distributed intelligence in the kitchen environment," in *International Conference on Universal Access in Human-Computer Interaction (UAHCI)*, Heraklion, Greece, 2014.

[57] C. Cristoiu, S. Moraru, D. Kristaly, D. Ungureanu and I. Moraru, "Home-based System for Elderly Assisted Living," in *International conference on research and innovation in computer engineering and computer sciences*, 2017.

[58] P. Mell and T. Grance, The NIST Definition of Cloud Computing. National Institute of Standards and Technology, 2011.

[59] A. Kulkarni and S. Sathe, "Healthcare Applications of the Internet of Things: A Review," *International Journal of Computer Science and Information Technologies,* vol. 5, no. 5, pp. 6221-6225, 2014.

[60] A. J. Jara, M. A. Zamora and A. F. G. Skarmeta, "An Internet of Things-based personal device for diabetes therapy management in Ambient Assisted Living (AAL)," *Personal and Ubiquitous Computing,* vol. 18, no. 4, pp. 1017-1028, 2014.

[61] M. R. Alam, M. B. I. Reaz and M. A. M. Ali, "A Review of Smart Homes—Past, Present, and Future," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 42, no. 6, pp. 1190-1203, 2012.

[62] "Watson Health," IBM, 2021. [Online]. Available: https://www.ibm.com/watson-health.

[63] X. Li, D. Li, J. Wan, A. V. Vasilakos, C.-F. Lai and S. Wang, "A review of industrial wireless networks in the context of Industry 4.0," *Wireless Networks,* vol. 23, no. 1, pp. 23-41, 2013.

[64] S. Zeadally and N. Jabeur, "Privacy in Internet of Things (IoT) technologies," *Procedia Computer Science,* vol. 98, pp. 461-466, 2016.

[65] OpenJS Foundation & Contributors, "Node-RED," [Online]. Available: https://nodered.org/. [Accessed 02 05 2024].

[66] D. Kristaly, S. Moraru, V. Petre, C. Parvan, D. Ungureanu and A. Mosoi, "A solution for mobile computing in a cloud environment for ambient assisted living," in *Mediterranean Conference on Control and Automation (MED)*, Zadar, Croatia, 2018.

[67] D. Kristaly, V. Petre and S. Moraru, "Using IoT and cloud technologies in monitoring systems for elderly," in *International Conference on Sensing and Instrumentation in IoT Era - ISSI*, Lisbon, Portugal, 2019.