Using a Large Data Model Explorer to Maintain a Healthcare Information System

Rubén Martínez Martínez*, Francisco Javier Bermúdez Ruiz*
Manuel Campos Martínez*†, José Manuel Juarez Herrero*†
*MedAI Lab, University of Murcia, Spain
†Murcian Bio-Health Institute (IMIB-Arrixaca)
Email: {ruben.martinez11, fjavier, manuelcampos, jmjuarez}@um.es

***Abstract*—Maintaining large scale healthcare information systems poses significant challenges due to their evolving complexity and sparse documentation. This paper presents a methodology and supporting tool for exploring and understanding extensive data models, motivated by and applied on the Wise Antimicrobial Stewardship Support System (WASPSS) application, an antimicrobial stewardship support system in production across several Spanish hospitals. The proposed tool enables incremental, interactive visualization of the Java Persistence API (JPA) based domain model in WASPSS, including inheritance and relationships, facilitating maintenance and onboarding. Key features include partial view saving, entity tagging, and dynamic inheritance propagation. An evaluation based on the Technology Acceptance Model (TAM) confirms the tool's perceived usefulness and ease of use among software professionals, supporting its applicability in real world maintenance workflows and its potential reuse in other healthcare contexts.**

Keywords-WASPSS; healthcare system; maintenance; large data model; model explorer.

I. Introduction

Hospital acquired infections, also known as Nosocomial Infections (NI), have become a serious public health problem. They are defined as infections that appear 48 hours after hospital admission and are often caused by multidrug resistant bacteria, which have lost susceptibility to common treatments, increasing their spread, severity, and lethality [1] [2]. Misuse of antibiotics contributes to the emergence of new resistances, limiting available therapeutic options [3]. NIs increase morbidity, mortality, and hospital costs due to more complex treatments and prolonged stays [4].

In the context described above, the `WASPSS` tool [5] (*Wise Antimicrobial Stewardship Program Support System*) emerges, as a decision support platform for Antimicrobial Stewardship Programs (ASP). `WASPSS` was initially developed by the University of Murcia and the Getafe University Hospital, where it was implemented in 2015, and has since been used daily by ASP teams as part of the National Plan against Antibiotic Resistance. In 2018, it began piloting in seven hospitals across various autonomous communities. Currently, `WASPSS` is in production in all hospitals of the Basque Country and the Murcia Health Service of the Region of Murcia.

`WASPSS` is a web application developed following the Model-View-Controller pattern and based on standard Java technologies. It offers interactive interfaces tailored to different ASP team profiles. The business logic is organized through façade objects, encapsulating specific functionalities and accessing persisted data via data access objects. Persistence is managed by Java Persitence API (`JPA`) and a `PostgreSQL` relational database, which serves as the central information core for all modules. `WASPSS` integrates with hospital systems via an `HL7` interface, a widely used standard in clinical environments for real time information exchange. Finally, the knowledge modules use `Drools`, allowing clinical rules to be applied to hospital data to generate alerts and support clinical decision making by the ASP team.

The `WASPSS` project has been under development for over 12 years. Over this time, its codebase has grown organically, incorporating new features and adapting to changing business requirements. However, the project documentation has not scaled at the same pace as the application. This hinders onboarding new developers and maintaining the system. This growth has far outstripped the capacity of the existing static documentation, which has not been systematically updated at the code's pace. The commitment of the developing team to software quality and ongoing development makes the effort to enhance the application maintainability, both meaningful and motivating. This will, in turn, facilitate to add new features, fix bugs, or make improvements without significant prior knowledge of the code. In addition, this also will speed up development, reduce the risk of errors, and ease knowledge transfer between developers. Examining further, the project includes over 1,000 Java classes, many organized in inheritance hierarchies and complex relationship networks. Additionally, the JPA domain model includes over 200 Java classes, and the database contains 166 tables across 9 schemas. These figures contrast sharply with the sparse documentation available. A good percentage of the code is undocumented, and there are few tool reports. Maintenance tasks thus become difficult due to the challenge of understanding the underlying model. Furthermore, existing market tools for exploring information system data models face difficulties in generating complete visual reports or diagrams containing all the model information. This feature is unsuitable for very large data models, as in the `WASPSS` project. In other words, trying to visually represent hundreds of entities in a single diagram becomes unintelligible to humans, with graphics that overlap and hide much of the information. All of this highlights the challenge of maintaining the tool, both for reasoning about and understanding the underlying domain model that drives the system and for grasping the structure of resources comprising the `WASPSS` project itself.

Note that in the development of an information system, the domain model represents the core of the software since it encapsulates the fundamental knowledge, rules, and processes of the business. Together with requirement specifications, this

model forms the foundation upon which the system is built and evolves. In approaches like Domain Driven Design [6], it is recognized that the true complexity of software lies not in the technology but in the domain itself, reflecting the design of the system. Thus, the domain model is not just another system component, but rather the central structure guiding its design. This work proposes a methodology for discovering and exploring a large data model in sparsely documented large scale information systems. The methodology is supported by a tool that allows loading the model information to reason about it incrementally and progressively, discovering the domain entities structuring the application and enabling development teams to work with customized views that allow them to segment the part of the model they want to work on. Since the exploration methodology relies on a visualization tool that progressively reveals the model, it is important that the information displayed matches the graphical context at each stage. This implies the need to visually propagate inherited information between entities in an object oriented paradigm–based model. The information propagated through inheritance will be displayed in some entities or others, depending on which entities are currently visualized (and the inheritance involved). Specifically, it means that our system parses each JPA entity, including its attributes, inheritance hierarchy, and relationships, to produce a normalized representation. Each entity is then rendered as a node in the interactive graph, while attributes and relationships are displayed as labeled fields and directed edges, respectively. Inheritance is represented visually by propagating the properties of parent classes to their child nodes when the parent is not explicitly shown. This mapping ensures that developers can incrementally reveal and reason about the full domain model without overwhelming diagrams. The current version of the tool offers only a basic level of customization for visualization. For instance, users cannot yet selectively choose which specific relationships to display or hide, nor can they freely reposition individual relationship edges within the diagram.

To guide our work, we formulate the following research questions: (i) how can the exploration and comprehension of a large, sparsely documented data model in a healthcare information system be improved?; (ii) what functionalities are required in a model exploration tool to support effective maintenance and onboarding of developers?; and (iii) to what extent is the proposed tool perceived as useful and easy to use by software professionals?. Research questions are aligned to the main contributions of this work: the proposal of a methodology for exploring large-scale, poorly documented data models; the development of a supporting tool; and the validation of this tool by end users.

In Section II, we enumerate the main tools that address similar problems. Section III presents the outcomes obtained from the proposed approach. Next, in Section IV, we evaluate how useful the tool is perceived. Finally, in Section V, we summarize the key findings and outline potential directions for further research.

II. Related work

Previous research has also explored interactive visualization techniques to manage the complexity of large healthcare information systems. [7] proposed *Owlready*, an ontology-oriented programming framework for biomedical ontologies that enables incremental exploration and automatic classification of complex domain models, addressing similar challenges of scalability and maintainability. [8] introduced *Health Timeline*, a timeline-based visualization that allows clinicians to progressively explore patient records, demonstrating that focused, interactive views improve comprehension of large clinical datasets. Likewise, [9] presented a richly interactive exploratory visualization tool for electronic health records, highlighting the value of dynamic filtering and navigation for handling extensive medical data. These works support the need for incremental, user-centered visualization approaches, which our proposed JPA model visualizer extends to the maintenance of large-scale healthcare data models.

In Java environments, complex data models with JPA entities, relationships, and inheritance are difficult to grasp directly from code as their size increases. To address this, several tools and plugins provide graphical visualization of domain models from JPA entities or database tables. Next, a brief review of notable tools is enumerated: (i) **JPA Diagram Editor (Eclipse Dali)** [10]: Free Eclipse plugin for visual JPA editing, but limited to IDE use and lacks dynamic, interactive diagrams; (ii) **Hibernate Tools / Mapping Diagram** [11]: Shows entity mappings and relationships in read-only form; offers basic hiding/collapsing of connections; (iii) **Jeddict (JPA Modeler)** [12]: NetBeans plugin for creating and editing entities with code–diagram sync and reverse engineering from databases; (iv) **IntelliJ IDEA Ultimate** [13]: Provides a persistence view for JPA entities with simple navigation and layout, but limited editing options; (v) **JPA Buddy** [14]: IntelliJ plugin focused on code generation with minimal visualization features; (vi) **Generic modeling tools** (DBeaver, SchemaSpy, etc. [15], [16]): Can draw database/UML diagrams but do not handle JPA annotations or source-level models.

The conclusion after analyzing various tools is that, although each one offers certain features that may be of interest, none of them allow for incremental and interactive exploration starting from an initial entity or for automatic management of inherited attribute propagation features we consider essential for understanding a large scale model. Moreover, most of them do not allow saving persistent partial views of the model, nor do they offer a smooth integration between visual editing and an interactive, dynamic diagram.

III. Results

The solution proposed in this work is based on offering, through a support tool, the ability for developers to progressively and interactively explore the application data model. Additionally, the information displayed in the visualization tool must be dynamic and adapted to the entities shown on screen, according to the data propagation behavior inherent to inheritance relationships in an object oriented data model.

Inheritance is a mechanism that allows a class (called a subclass or child class) to inherit the properties (attributes) and methods (behaviors) of another class (called a superclass or parent class). In this context, when entities (classes) are related through properties (for example, a property referencing another class), those relationships are also propagated to the subclasses. That is, if a superclass has a property managing a relationship with another entity (e.g., a list of related objects), the subclass will also inherit that relationship, allowing for the management of relationships between objects through inherited properties. Therefore, we propose the design and implementation of a graphical visualization tool that enables any member of the development team to explore and navigate the JPA model interactively, displaying both the own class and inherited information depending on the selected visualization elements. A diagram based on UML class diagrams [17] will be used, with slight visual modifications to the standard UML notation. Since we are only interested in the structural information of the data model, the current version of the tool omits information about methods (functions) of each entity in favor of a clearer visualization. Figure 1 shows an example of entities with attributes and relationships.

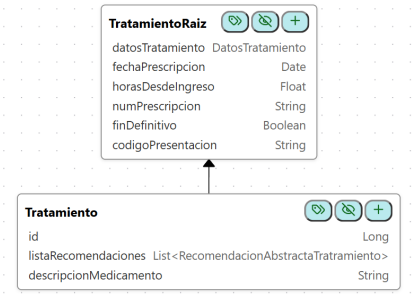

Figure 1: Example of entities with attributes and one inheritance relationship.

A. Main functionalities

The basic functionalities of the application must be as follows:

- Ability to progressively and interactively explore the model to discover relationships and inheritance from the entities themselves.
- Ability to dynamically add and remove entities from the visualized diagram. Each time the diagram is modified, our visualizer will show inherited information consistent with the current state of the diagram.
- Ability to save and retrieve partial views of the model. This allows designing visualizations focused on specific parts of the model, facilitating understanding and reasoning.
- Ability to categorize entities using tags. This allows filtering the model based on the tags applied to entities, enabling the display of all entities with a specific set of tags selected by the developer with a single click.

Although the visualizer developed in this work was designed with the primary goal of integration into the WASPSS project ecosystem, it is important to highlight that its implementation is not tightly coupled to that system. The modular architecture of the visualizer allows it to be reused in other contexts, as long as the data model can be adapted to be consumed by the visualizer. The only requirement is that the model be object oriented, i.e., composed of entities with attributes, relationships, and hierarchical structures similar to those of an object oriented system. This domain independence opens the door for the visualizer to be applied in other medical information systems beyond WASPSS, reinforcing its value as a generic tool for exploring complex data models.

B. Architecture of the Visualization Tool

The architecture consists of two functional units (see Figure 2):

- A parser responsible for analyzing and extracting information from the data model based on project resources (source code).
- An interactive visualizer that displays the information extracted from the data model.

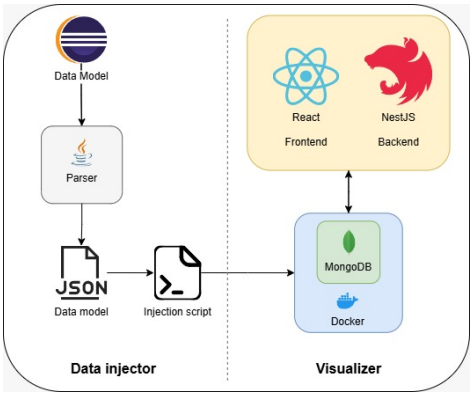

Figure 2: Architecture of the Visualization Tool.

C. Extraction and Normalization of the JPA Model

The solution arises from the need to visually represent a complex data model. To this end, a parser has been developed that automatically analyzes the project's source code and extracts structured information about JPA entities and their relationships. This process identifies each entity, its

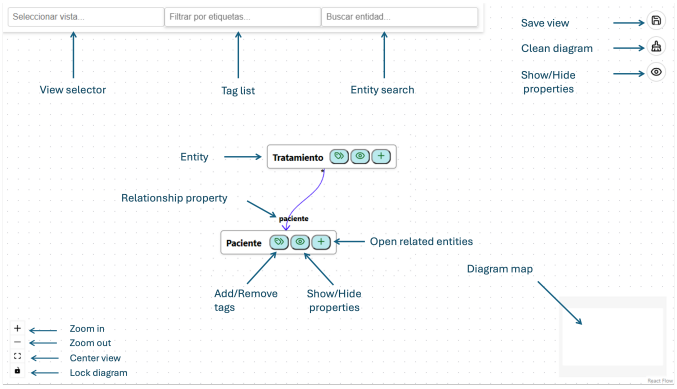

Figure 3: User Interface of the Visualization Tool.

inheritance hierarchy, and the different types of relationships. The output is a set of structured representations (one per entity) stored in a avaScript Object Notation (JSON) file, which can be processed by subsequent components of the solution. Thanks to the use of JSON as an intermediate format, the visualizer is not directly coupled to the `WASPSS` project and can be reused in any other information system for which source code is available. This means that, as long as a properly structured JSON file is available, the tool can be reused without modifications. This promotes reusability of the visualizer in other contexts and ensures greater flexibility and scalability. The JSON metamodel defines the basic properties for an entity identified by the parser. These are: `className`, `parentClassName` (the class from which it inherits, if any), `packageName` (the full class path), and a collection of `fields`, which are defined by `fieldName`, `fieldType`, `isRelationship` (indicating whether the attribute is a simple attribute or whether it represents a relationship with another entity), `relationshipType` (indicating the cardinality of the relationship) and `targetEntity` (indicating the entity with which it is related).

D. Storing Information

The parser analyzes the project source code (i.e. the JPA classes) to extract the model representation in JSON format. This JSON file is then loaded via a script into a MongoDB database (running in a Docker container). The choice of database is due to its ease of working directly with JSON documents. The visualizer uses the database to read the data model and persist interactive visualization information from developers, managing views and tags.

Once the model is extracted, the data is inserted into a database via a script. This database, in addition to containing the complete list of model entities and their relationships, provides the visualizer with the capability to store user created views and entity assigned tags. Therefore, this component acts as the system persistence core and ensures data consistency throughout user interaction with the visualizer.

E. Interactive Model Visualization

The core functionality of the solution is a graphical interface that allows users to interactively explore the entity model. Through this interface, the user can:

- Visualize JPA entities as nodes in a dynamic diagram.
- Explore relationships between entities, visually represented as directed edges.
- Save partial views of the model for later retrieval.
- Tag entities to aid organization and filtering.

During usage, the visualizer automatically manages the visual graph's consistency, including the incorporation of inherited relationships when ancestor entities are not present in the current view.

F. Interface Elements

The entity visualizer interface includes various interactive elements designed to facilitate data model exploration and customization. At the top of the interface are three key components (see Figure 3):

- `View Selector`: allows the user to switch between different views previously created. A view is a set of saved entities that can later be restored.
- `Tag List`: provides tag based filtering to reduce the number of visible entities, showing only those associated with selected categories. This is useful for locating all entities belonging to a specific concept without searching one by one.
- `Entity Search`: offers a text field to search for a specific entity by name or navigate through the full list.

In the central workspace, entities are represented as nodes. Each node appears as a rectangle with three buttons, each providing specific functionality (explained below). Relationships between entities are shown as directed edges, labeled with the name of the relation (e.g., patient) and its cardinality. At the top right, three additional buttons provide global functionality:

- `Save View`: stores the current diagram layout as a new view, including entities, relationships, and their positions.
- `Clear Diagram`: removes all visible nodes and any selected views or tags from the current diagram.
- `Show/Hide Properties`: toggles the visibility of all entity fields globally.

A minimap is displayed in the bottom-right corner, offering a quick overview of the canvas and enabling faster navigation. Finally, in the bottom-left, there are navigation controls enabling: `Zoom in / Zoom out`, `Center view` and `Lock diagram` (temporarily disables interaction to prevent accidental changes).

Figure 4: Node example.

Each node consists visually of a header with the entity name and three buttons. As seen in Figure 4, from left to right:

- `Tag Button`: opens a dropdown for assigning or removing tags. Users can search, activate, deactivate, or create new tags. Any change triggers an call to sync the backend.
- `Show/Hide Fields Button`: toggles the visibility of the entity's attribute list via an internal boolean. It also respects the global "show all fields" toggle.
- `Connection Button`: arguably the most important one. It displays a dropdown of related entities not yet shown in the diagram. These relationships include explicit ones, inherited children, and non-visible ancestors. When a user selects an entity, handleSelectedEntity is called to add the node and update edges automatically, enabling interactive discovery of the data model.

G. User Experience Enhancements

Several technical improvements were added to enhance user experience:

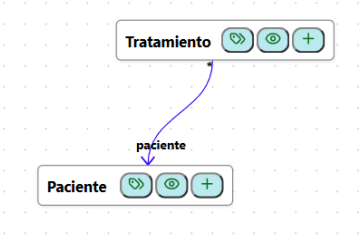

Figure 5: Simple relationship.

Figure 6: Multiple relationship.

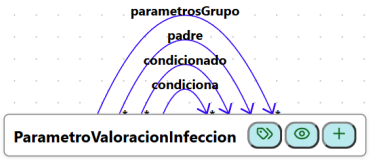

Figure 7: Reflexive relationship.

- When opening the tag creation modal, the focus is automatically placed in the input field.
- Scrolling inside the relationship dropdown does not affect canvas zoom.
- Clicking outside dropdowns closes them automatically (e.g., when navigating or using top filters).

H. Relationships

Figures 5, 6, and 7 illustrate simple, multiple, and reflexive relationships, respectively. Custom algorithms calculate the exact label and cardinality positions, accounting for node geometry and displacement. These implementations ensure uniform, clear rendering of semantic model elements. Three key elements are rendered per relationship:

- `Relation Label`: shown near the center of the edge; represents the field name of the association.
- `Source Cardinality`: marked with * for "ManyTo..." cases, shown near the source node.
- `Target Cardinality`: marked with * for "...ToMany" cases, shown near the target node.

All labels are interactive: clicking on any of them highlights the corresponding edge and its metadata (e.g., increasing size or changing color).

I. Saving and Managing Views

One key feature is the ability to save partial views of the data model. This allows users to focus on a subset of entities relevant to a specific task, hiding unrelated elements. Internally, each view stores exact canvas positions so when reloaded, the layout is reconstructed identically, maintaining the user's custom organization.

An interactive dropdown enables quick access to saved views:

- Typing in the search field dynamically filters available views.
- Selecting a view automatically loads its entities and their positions.
- Each view includes a delete button for direct removal.

J. Filtering by Tags

The main goal of tag based filtering is to allow users to instantly retrieve all entities marked with a certain tag. These tags can be user defined at any point. The tag dropdown appears when clicking the label icon in the nodes. It allows assigning existing or creating new tags. The tag creation modal includes validations to prevent empty or duplicate names.

IV. Evaluation

A validation of the tool was carried out by different users following the Technology Acceptance Model (TAM) [18], which allows us to understand how useful the tool is perceived to be, according to its purpose and the methodology defined in this work. We will describe the methodology used to evaluate the visualizer, as well as the results obtained and the conclusions drawn from them.

A. Technology Acceptance Model

TAM is one of the most established frameworks for analyzing how users adopt new technologies. This framework identifies two main determinants: perceived usefulness (the degree to which using the tool improves task performance) and perceived ease of use (the degree to which using the tool requires minimal effort) [18]. Therefore, this evaluation determines whether the tool meets two key hypotheses derived from TAM:

- H1. The application is simple and intuitive to use.
- H2. The application is perceived as useful.

Additionally, the model considers the user attitude toward the technology and their future intention to use it [18].

B. Evaluation Instrumentation

To conduct the evaluation of our visualizer, we designed an exercise [19] to be carried out using the application. Once the exercise was completed, the participants were asked to fill out a questionnaire divided into two sections:

- Demographics and experience: Age, gender, level of experience using computers, and experience with web applications.
- TAM Dimensions: Questions [20] grouped by perceived ease of use (items B1 to B9), perceived usefulness (items B10 to B16), attitude toward use (items B17 and B18) and intention to use (items B19 and B20) the visualizer.

C. Participant Demographics

The questionnaire was administered to 13 faculty members and staff from the Faculty of Computer Science at the University of Murcia. 61.5% of the participants were male and 38.5% female. The average age was 39.23 years. Most participants reported a high level of experience with both computer use and web applications.

D. Validation Results

Based on the responses collected, the mean values and standard deviation were calculated for each item using a 5-point Likert scale (1 = Strongly disagree, 5 = Strongly agree). The results obtained reflect an overall positive assessment of our visualizer by the participants. We can observe this in Table I.

TABLE I: MEANS AND STANDARD DEVIATIONS

Questions	Mean	Stand. Dev.	Questions	Mean	Stand. Dev.
B1	4.69	0.48	B11	4.92	0.28
B2	4.46	0.52	B12	4.46	0.88
B3	4.92	0.28	B13	4.85	0.55
B4	4.85	0.38	B14	4.23	1.09
B5	4.62	0.51	B15	4.46	0.66
B6	4.69	0.63	B16	4.92	0.28
B7	4.85	0.38	B17	4.85	0.38
B8	4.92	0.28	B18	4.69	0.63
B9	4.77	0.60	B19	4.69	0.48
B10	4.77	0.44	B20	4.77	0.44

For Perceived Ease of Use (items B1 to B9), a mean score of 4.75 was obtained, indicating that users found the visualizer easy and intuitive to use. The questions related to Perceived Usefulness (items B10 to B16) achieved a mean score of 4.66, highlighting that participants view the visualizer as a functional and valuable solution. The questions related to Attitude Toward Use (items B17 and B18) had a mean score of 4.77, indicating that participants believe using the visualizer is a good idea. Finally, the Intention to Use (items B19 and B20) received a mean score of 4.73, suggesting a strong willingness to use the visualizer in a real world context. The four values are close to 5, which corresponds to the rating *Strongly Agree*. These results allow us to conclude that our visualizer shows high levels of user acceptance.

V. Conclusion and Future Work

This work presents a methodology and tool to support the exploration and maintenance of large scale, poorly documented data models in healthcare information systems. Applied to the WASPSS platform, the tool enables incremental, inheritance visualization of complex JPA based domain models, significantly improving comprehension and maintainability. Our solution allows developers to interactively explore entities, manage customized views, and filter components using tags. The modular architecture ensures reusability beyond WASPSS, offering potential benefits for other object oriented medical systems. The positive results from the user evaluation, based on the TAM demonstrate that the tool is both intuitive and effective, confirming its value in real world development.

Future work will focus on extending the tool with collaborative features, model editing capabilities (such as the ability to show/hide relationships or reposition edge labels), and integration with automated documentation pipelines. In addition, we plan to extend the evaluation by increasing the number of participants and including professionals outside the academic environment, such as developers and IT staff from companies, in order to obtain a broader and more representative assessment of the tool.

Acknowledgment

This work was partially funded by the CONFAINCE project (Ref: PID2021-122194OB-I00) by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", by the "European Union".

References

[1] A. Iqbal *et al.*, "Nosocomial vs healthcare associated vs community acquired spontaneous bacterial peritonitis: Network meta-analysis", *The American Journal of the Medical Sciences*, vol. 366, no. 4, pp. 305–313, 2023.

[2] R. B. McFee and G. G. Abdelsayed, "Clostridium difficile", *Disease-a-Month*, vol. 55, no. 7, pp. 439–470, 2009, Clostridium difficile: Emerging Public Health Threat and Other Nosocomial or Hospital Acquired Infections.

[3] WHO Regional Office for Europe and European Centre for Disease Prevention and Control, *Antimicrobial resistance surveillance in europe 2022 – 2020 data*, Copenhagen, 2022.

[4] R. E. Nelson *et al.*, "National estimates of healthcare costs associated with multidrug-resistant bacterial infections among hospitalized patients in the united states", *Clinical Infectious Diseases*, vol. 72, no. Supplement$_1$, S17–S26, Jan. 2021.

[5] B. Cánovas Segura, A. Morales, J. M. Juarez, M. Campos, and F. Palacios, "Waspss: A clinical decision support system for antimicrobial stewardship", in *Recent Advances in Digital System Diagnosis and Management of Healthcare*, K. Sartipi and T. Edoh, Eds., IntechOpen, 2020.

[6] E. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[7] J.-B. Lamy, "Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies", *Artificial Intelligence in Medicine*, vol. 80, pp. 11–28, 2017, ISSN: 0933-3657.

[8] A. Ledesma *et al.*, "Health timeline: An insight-based study of a timeline visualization of clinical data", *BMC Medical Informatics and Decision Making*, vol. 19, Aug. 2019.

[9] C.-W. Huang *et al.*, "A richly interactive exploratory data analysis and visualization tool using electronic medical records", *BMC medical informatics and decision making*, vol. 15, p. 92, Nov. 2015.

[10] Eclipse-Foundation, *Dali java persistence tools*, https://projects.eclipse.org/projects/webtools.dali, Last accessed: July 13, 2025.

[11] Red-Hat, *Hibernate tools*, https://tools.jboss.org/features/hibernate.html, Last accessed: July 13, 2025.

[12] Jeddict-Project, *Jeddict - jpa modeler and more*, https://jeddict.github.io, Last accessed: July 13, 2025.

[13] JetBrains, *Jpa in intellij idea*, https://www.jetbrains.com/help/idea/jpa-buddy.html, Last accessed: July 13, 2025.

[14] J. B. Team, *Jpa buddy — productivity tool for jpa/hibernate developers*, https://www.jpa-buddy.com, Last accessed: July 13, 2025.

[15] D. Corp., *Dbeaver — universal database tool*, https://dbeaver.io, Last accessed: July 13, 2025.

[16] S. Team, *Schemaspy — database documentation tool*, https://schemaspy.org, Last accessed: July 13, 2025.

[17] I. Jacobson, G. Booch, and J. Rumbaugh, *UML: The Unified Development Process*. Addison-Wesley, 2000.

[18] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology", *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[19] https://github.com/fjavier-umu/healthinfo25/blob/main/exercise, Last accessed: July 13, 2025.

[20] https://github.com/fjavier-umu/healthinfo25/blob/main/questionnaire, Last accessed: July 13, 2025.