A Framework for Fast Development of Customized Telehealth Applications

Baptiste Alcalde
eHealth Institut
FH JOANNEUM
Graz, Austria
baptiste.alcalde@fh-joanneum.at

Lukas Wechtitsch
Independent
Graz, Austria
wechtitsch.lukas@gmail.com

***Abstract* — Telehealth has the potential to enhance health services and lower their costs, particularly in remote regions. Most telehealth applications in the literature and on the market try to fulfill a similar set of functionalities. These functionalities are identified and compared in several theoretical frameworks. However, to our knowledge, there are no practical implementations of such frameworks. An easy-to-use framework to generate customizable telehealth applications would be beneficial for health care providers, particularly for small or middle-sized providers who lack the technical knowledge or/and the budget for that. In this paper, we design and develop a framework enabling health care providers the fast development and extension of such applications.**

Keywords – telemedicine; teletherapy; framework; personal ehealth.

I. INTRODUCTION

Telemedicine and telehealth are defined as "the use of information and communication technologies (ICTs) to solve health problems, especially for people living in remote and underserviced areas" [14]. Various services can be offered by telemedicine, such as symptom assessments or the provision of information about medications [6].

The range of new telehealth applications, including changes and innovations in the digital health sector resulting from the COVID-19 pandemic, significantly increased [27]. Not only larger, mostly government or institutional providers but also private health providers in various specialization fields need to resort to telehealth solutions.

However, independently of the specialization field, these solutions share several basic functionalities [6]. For instance, the solution must enable patient management, and remote communication options.

In [31], a systematic review compares frameworks for the implementation of telehealth services. However, the focus is the contribution to the success rate of these services. Most of the selected papers (and other like [32]) discuss the evaluation of telehealth services. The framework proposed in [30] identifies 6 structural layers for the key structural components in telehealth applications along the patient journey. However, this framework (and other like [33], or [34]) is only theoretical, i.e., to our knowledge, no practical implementations are available. Therefore, the aim of this paper is to provide an easy to use and modular practical implementation for telehealth services by means of a framework.

Bearing in mind that the development of a software solution is a costly activity [17], and that telehealth applications share similar functionalities, we propose a framework which aims at the fast development of customized applications with almost no previous technical knowledge. Therefore, utilizing this framework, the health service provider can alleviate the development costs and benefit from a customized application. This framework is particularly targeted at small or middle-sized providers, who often lack a budget for software development. This framework should also fulfill several requirements among others modularity, extensibility, and scalability. Moreover, the applications generated through this framework share the same structure, leading to higher interoperability.

II. REQUIREMENTS

Literature research was performed to identify the most essential functionalities of a framework for telehealth applications.

First, we identified relevant telehealth applications available on the market. We selected the following applications:

- Care01 [5]: Care01 is an application that specializes in digital surgery management. Care01 covers functions, such as appointment scheduling, patient management, but also video calls with patients.
- Clearstep [7]: Clearstep offers components for symptom checking and patient management. With the Smart Care Routing™, and the use of Artificial Intelligence (AI), the user should receive information about the health status or whether a visit to the doctor is necessary.
- Doxy.me [11]: With Doxy.me, components, such as video telephony, chats, patient management, and a dashboard are freely available. Certain functions, such as referrals can also be commercially subscribed to.

- Latido [20]: offers a product that can be integrated into existing medical software. Latido focuses on video communication and patient management. Additionally, functionalities, such as appointment scheduling or financial management are also provided.
- OpenEMR [23]: OpenEMR stands for Open Electronic Medical Record and basically offers the storage of EMRs, with which clinical data of patients, information on invoices, or medical history can be easily stored [22]. OpenEMR also offers functionalities, such as creating patient appointments or searching for medications.
- Samedi [26]: Samedi includes the following functions: online appointment scheduling, calendar and resource planning, payment function, video consultation, vaccination management, online patient forms, and a patient portal.

As a result of our literature research, we identified the following most relevant functionalities or components:

- Patient management (core): information, such as first and last names, and date of birth, should be stored.
- Video calls: enabling health service providers and their patients to communicate.
- Symptom tracking: recording symptoms of a given patient is needed to assess the course of an illness.
- Online appointments: the patient and the health service provider can book appointments with one another.
- Content management: the health service providers can create content, e.g., mini lessons, video exercises, or information to enhance health literacy, and grant access to this content to their patients.
- Interoperability & Data exportability: medical data should be interoperable so that they can be integrated in third party tools fulfilling the supported standards.
- Personalization: the application should be adaptable to the corporate design of the company. This includes defining a name, a background color, and a logo for the application.

In addition to these functionalities, we identified requirements related to the flexibility and scalability of the framework. These are:

- Open-closed principle [15]: it should be easy to add new modules/functionalities to the framework without modifying the existing code base.
- Dynamic: the user can select the modules that should be included in the application.
- Secure: data security should conform to the General Data Protection Regulations (GDPR) [12].
- Performance and availability [16]: modules and databases should be replicable to allow on-demand scalability.

III. FRAMEWORK ARCHITECTURE

The framework should contain the functionalities identified in Section II. An overview of the framework architecture is displayed in Figure 1.

The *patient management* functionality is the core of any application and should always be present. All other functionalities are optional and can be added to the generated application when needed.

For instance, the users can decide that their application may only contain the *video calls*, *online appointments*, and *content management* functionalities in addition to the mandatory *patient management*. This would be typical for applications where a remote support or caring, but no medical data are required: the user can book appointments which will take place through video calls, and between these appointments, the user can perform some exercises provided in the *content management*.

The implementation of this architecture applies concepts similar to the ones of microservices and modular programming [19]. Microservices should fulfill the following requirements:

- Independent databases per service.
- Independent hosting.
- Independent codebase.

As displayed in Figure 1, these requirements are only partially fulfilled by our framework. Indeed, there is only one database, and the hosting takes place on one target. This decision was made to simplify the deployment of the resulting telehealth application, since the framework should also be used by persons with little technical knowledge. However, the implementation is modular and sufficiently scalable to fulfill all requirements defined in Section II.

Figure 2 displays the database model. The database stores the data required for the functionalities explained previously and illustrated in Figure 1.

The *User* table is needed for the core functionality *patient management*. The patient data is limited to a minimum but can be extended as we will discuss in Section VI. On the one hand, we need data related to the registration for the resulting application, such as an email-address, a username, and an encrypted (hashed and salted) password.

On the other hand, we need real-life data related to the user, such as the first name, last name, and date of birth. To differentiate between patients and medical professionals, a *Role* table was added. Therefore, the user will be assigned a given role through an attribute in the *User* table.

For the *symptom tracking* functionality, we modeled the Symptom table. A symptom has a name and a description, is related to a user, and started on a given date. Moreover, we can record if the symptom is active through the eponymous attribute. This structure would be sufficient for relatively small applications. However, the splitting of the symptom name and description, as well as a code, in a separate table could be performed if the resulting application must use standards, such as SNOMED, or the International

Classification of Diseases (ICD), or Logical Observation Identifiers Names and Codes (LOINC) [4].

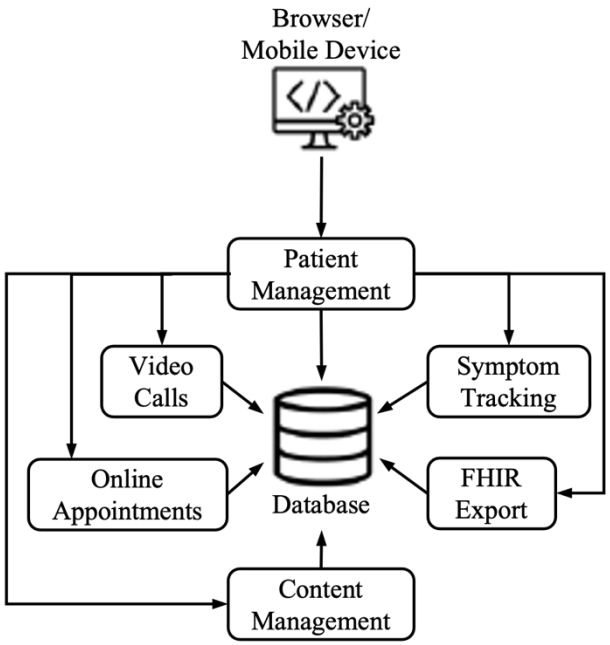

Figure 1. Framework architecture.

In order to implement the the *online appointment* functionality, the *Appointment* table records a description for this appointment, the location (for instance a link to a video call), an attribute that states if the appointment is accepted, and a reference to the two participants of the appointment. Note that the participants simply are users recorded in the *User* table. Checking if users are patients or medical professionals can be done through the role (*role_uid*) attribute of the user.

We designed two tables for the *content management* functionality: *Course* and *CourseEntry*. Thus, the content belonging to a course can modularly be organized in entries. These entries can represent textual information or exercises, in which case the *text* attribute can be used. However, they can also incorporate other media, in which case the *attachment* attribute can be utilized. The date enables the chronological ordering of the course items.

The video communication and the data export functionalities do not require their own tables. However, the data export functionality needs to access the data stored in other tables, for instance, the *User* and *Symptom* tables.

If a microservice architecture with separate databases shall be implemented, the database and the relevant tables are split. Then, a solution should be implemented to manage and grant the appropriate access to the databases connected to the various functionalities. This could be done, for instance, through Application Programming Interfaces (APIs) for each microservice.

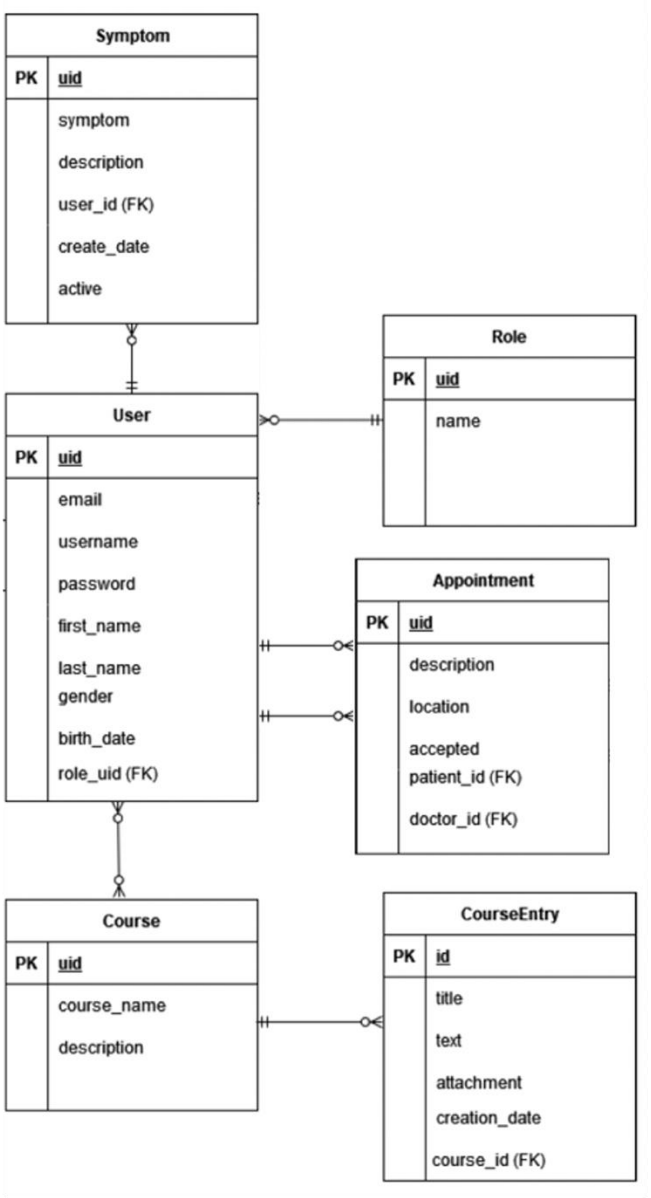

Figure 2. Database Model.

IV. TECHNICAL DESCRIPTION OF THE FRAMEWORK DEVELOPMENT

The implementation of this framework involved several tools.

MariaDB [18] was used as the database. The database was installed locally for development. It is a relational database and freely available. It has a similar implementation to MySQL but is a fork of the MySQL project. We chose a relational database for its ACID (atomicity, consistency, isolation, and durability) compliance and its transaction capabilities.

The programming language **Java** [28] in its version 11 was chosen for the backend and the initializer (see Section VI). Java was preferred due to its platform independence, its

relative popularity among developers over time, and the availability of further frameworks such as the Spring Framework.

The **Spring Framework** offers several functionalities in the form of standalone projects for web applications [8]. The following Spring projects were used as part of the development:

- Spring Boot: for a standalone application that can replace or embed resources, such as a web server.
- Spring Web: for the simplified creation of web Model View Controller (MVC) applications.
- Spring Java Persistence API (JPA): as a persistence framework alternative to Hibernate for database operations.
- Spring Security: for user administration and security in the application, as well as for authentication, authorization, and registration.
- Spring Session: for managing the sessions of the logged-in users.
- Spring Websocket: for the creation of web sockets, needed for instance in the video communications.

WebRTC [3] is a technology or standard that offers free video communication.

In the frontend, **HTML** was used as the basis, **JS** for client-side operations and **CSS** for styling [25].

Maven was used as the build and configuration tool [9]. Gradle can be used as an alternative.

V. Technical Description of Deployment

As we explain in Section VI, the framework allows two use cases: initializer and application development kit.

To perform the former use case, the initializer can be hosted on a public website or locally.

For the latter use case, the framework components are simply checked out and the required configurations, such as the connection information (IP, etc.) of the database must be completed.

Since all components are Spring Boot applications, no separate web server, such as Tomcat and Glassfish is required. However, Spring Boot still offers the option of using a different web server.

The components are all implemented as independent web applications. This means that they must also be accessible on different ports. Six ports must therefore be reserved for the six components. During development, an additional port must also be reserved for the initializer. These seven ports are set per default (8080-8085 and 8090) and can be configured in the property files, if needed. The port configuration and protection should be carefully performed to minimize security risks and having several open ports might imply more overhead for this task.

Each component and the initializer have their own property file, which can be accessed from outside the compiled component. This implies that the port and other settings can be adjusted without having to rebuild the component and create a new resource file. On the one hand, similarly to port configuration, having many property files could increase the complexity and potentially the security risks. On the other hand, a centralized property file would be a single point of failure and would be less flexible.

If this number of ports is too high or unsuitable for practical use, a reverse proxy can be used. This maps requests and stands between the clients and the internal server, i.e., requests to the one public port are mapped to internal ports. Therefore, only one public port would be needed and declared as an open port in the reverse proxy configuration. Nginx offers a possible implementation of reverse proxies [2]. Obviously, the use of a reverse proxy would add another layer of complexity and potential points of failure to the system.

Docker can also be used to create the infrastructure more easily. With Docker, the application and its dependencies can be encapsulated [24]. The Docker image and container can then be deployed independently of the underlying operating system and enables more scalability. Nginx can also be used within Docker to implement a reverse proxy and thus minimizes the opened public ports as mentioned earlier.

VI. Use Cases

The framework can be used mainly in two ways:

- Initializer or application generator: the users just want a way to produce a telehealth application in a few clicks by selecting the functionalities they desire and optionally their corporate design.
- Application development kit: the users modify and enhance the framework with their own functionalities and customizing. The framework is then like a Software Development Kit (SDK), i.e., the core of the application is already available.

When utilizing the initializer, the user will go through several steps leading to the generation of the desired application. The user is guided through the process with help of descriptions and explanations in each step.

In the first step or graphical user interface (GUI), the user can personalize the appearance of the resulting application. This is the chosen way to fulfill the personalization functionality identified in Section II. Indeed, in the initializer, the users can personalize the application in three ways: a name corresponding to the name of their company or product, a color corresponding to the corporate design of their company, and a picture corresponding to the logo of the company or product.

In the second step, the users select, by means of checkboxes, the components, or functionalities that they want to include in the generated application. The available components are the ones listed in Section II: *video calls*, *symptom tracking*, *online appointments*, *content management*, *interoperability & data exportability*. Note that there is no checkbox for the patient management component since this component is mandatory and therefore, always part of the generated application.

Then, the user can click on a *Download* button and get a zip file containing all the resources needed to start the telehealth application. The resources are stored in three directories: a *config* directory containing the configuration files for each of the selected components, a *file* directory containing the database creation scripts, and a *module* directory containing the generated components as individual jar-files. The database still needs to be created using the included scripts. Subsequently, the user can already start the application.

When utilizing the framework as an SDK, the developer only needs to download the sources and develop the desired extensions or modifications.

VII. LIMITATIONS

Economical, ethical, social, and legal aspects might not have been considered to their full extent. The aim of this paper was to present a prototype, not a market-ready product.

Telemedicine and telehealth result in cost savings through reduced expenses for examinations in the clinical facilities or travel times [29], as patients also receive treatment at home, and could decrease the waiting times or bed occupancy in hospitals [1].

The framework proposed in this paper aims at reducing the effort required to design and develop a new software architecture for telehealth applications, even if some tasks for customization and deployment remain. However, new or existing expenses should be taken into account, such as investments in new hardware and software, organizational and structural integration into existing supply structures [10].

Several ethical aspects should be considered when using telemedicine and telehealth, such as the difference in quality of the collected data. The quality of the data can have an impact on the quality of the treatment and the trust between medical staff and patients [1].

Patients may also be reticent to agree to a continuous tracking and monitoring, even if this monitoring could lead to a more successful treatment [21].

Telemedicine and telehealth can be a means against the shortage of doctors in rural areas and poorer care in remote areas, as patients can communicate and have access to health services remotely.

However, it must also be noted that this might lead to further concentration of the medical infrastructure in bigger cities.

Moreover, the personal contact between patients and health service providers, as well as between patients and their relatives, for example when accompanying them to doctor's visits could be gradually reduced [21]. Since social contacts contribute to the health of patients, this aspect would be counter-productive.

In this paper, we developed a framework for telehealth and took legal aspects, such as GDPR into account. The use of Spring Security enables the fulfillment of several requirements of the GDPR through authentication, authorization, and user administration and security in the application. However, the users are responsible for the security of the database – for instance the strength of the password for the root users – which could be a security weakness.

Other questions regarding the liability, attribution, distribution, copyright, and warranty of the framework are still open. For instance, the following options are available for offering the framework: open source, GNU, or Creative Common (CC).

Another question is to which extent the framework or part of the framework fall under the Medical Device Regulations (MDR) [13].

VIII. CONCLUSION AND FUTURE WORKS

In this paper, we propose a framework for the fast development of customized telehealth applications based on requirements identified in the literature and in applications available on the market (see Section II). We present an architecture fulfilling these requirements (see Section III), show how to implement it (see Section IV) and deploy the product (see Section V). After that, we explain how the user can create an application with help of this framework or extend its functionalities (see Section VI). Finally, we identify several limitations for this framework (see Section VII).

As described in this paper, we tried to focus on the most wide-spread functionalities found in telehealth applications. The implementation of further components or functionalities is obviously a topic of interest for future implementations. For instance, one could evaluate the interest of an AI-chatbot, or components dedicated to accounting tasks.

We also mentioned that the recorded data are only partially following standards in the framework. To enable an easier integration with other tools, more data interoperability, reached through the implementations of standards, would be an advantage.

We are aware that scalability is a challenge, and we show how Docker can answer this question. However, with an increasing number of components, Kubernetes could be a better choice for the management of the containers.

Finally, we would like to evaluate the most appropriate model (open source, GNU, CC, proprietary) and make this framework available to the public.

ACKNOWLEDGMENT

We thank our colleague Anita Töchterle for the comments and proof-reading that greatly improved the manuscript.

REFERENCES

[1] A. Atac, E. Kurt, and S. E. Yurdakul, “An overview to ethical problems in telemedicine technology”, Procedia-Social and Behavioral Sciences, vol.103, pp. 116–121, 2013.

[2] A. Baeldung, “Serving Multiple Proxy Endpoints Under a Location in Nginx”, 2022.

https://www.baeldung.com/linux/nginx-multiple-proxy-endpoints. Retrieved: February, 2024.

[3] N. Blum, S. Lachapelle, and H. Alvestrand, "Webrtc: Real-time communication for the open web platform", Communications of the ACM, vol. 64, no. 8, pp. 50–54, 2021.

[4] O. Bodenreider, R. Cornet, and D. J. Vreeman, "Recent developments in clinical terminologies — snomed ct, loinc, and rxnorm", yearbook of medical informatics, vol. 27, no. 01, pp. 129–139, 2018.

[5] Care01. https://www.care01.com/. Retrieved: February, 2024.

[6] M. F. Chiang, J. B. Starren, and G. Demiris, "Telemedicine and Telehealth", Biomedical Informatics, pp. 667-692, 2021, doi:10.1007/978-3-030-58721-5_20.

[7] Clearstep. https://www.clearstep.health/. Retrieved: February, 2024.

[8] I. Cosmina, R. Harrop, C. Schaefer, and C. Ho, "Pro Spring 6: An In-Depth Guide to the Spring Framework", Apress, July 2023. ISBN: 9781484286401.

[9] M. Tyson, "What is Apache Maven? Build and dependency management for Java.", 2020. https://www.infoworld.com/article/3516426/what-is-apache-maven-build-and-dependency-management-for-java.html. Retrieved: February, 2024.

[10] S. Demirci, F. Kauffeld-Monz, and S. Schaat, "Perspectives for Telemedicine – Prerequisites of the Scalability and Market Potential" (original title: "Perspektiven für die Telemedizin – Voraussetzungen der Skalierung und Marktpotenzial"), IIT Berlin, May 2021.

[11] Doxy.me. https://doxy.me/en/. Retrieved: February, 2024.

[12] European Commission, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)", European Commission, 2016.

[13] European Parliament, "Regulation (EU) 2017/745 of the European Parliament and of the Council of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC (Text with EEA relevance.)", European Parliament, 2017.

[14] "Fundamentals of Telemedicine and Telehealth", edited by Shashi Gogia, Academic Press, Elsevier, 2019. ISBN: 978-0-12-814309-4.

[15] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Professional, 1994, ISBN: 0201633612.

[16] R. Jiménez-Peris and M. Patiño-Martínez, "Replication for Scalability", in: L. Liu and M. T. Özsu (eds), Encyclopedia of Database Systems, Springer, Boston, MA, 2009. doi:10.1007/978-0-387-39940-9_314.

[17] M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies", in *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33-53, Jan. 2007. doi: 10.1109/TSE.2007.256943.

[18] E. Kenler and F. Razzoli, "MariaDB Essentials", Packt Publishing Ltd, 2015.

[19] M. Larsson, "Microservices with Spring Boot 3 and Spring Cloud", 3rd edition, Packt Publishing Ltd, Aug. 2023. ISBN: 9781805128694.

[20] Latido. https://latido.at. Retrieved: February, 2024.

[21] A. Manzeschke, "Telemedizin und ambient assisted living aus ethischer perspektive", Bayrisches Artzeblatt, 2014.

[22] J. Noll, S. Beecham, and D. Seichter, "A qualitative study of open source software development: The open emr project", in 2011 International Symposium on Empirical Software Engineering and Measurement, pp. 30–39. IEEE, 2011.

[23] OpenEMR. https://www.open-emr.org/. Retrieved: February, 2024.

[24] N. Poulton, "Docker Deep Dive", Nielson Book Services, Mai 2023. ISBN: 978-1916585256.

[25] J. N. Robbins, "Learning Web Design - A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics", 4th edition, O'Reilly, 2012. ISBN: 978-1-449-31927-4.

[26] Samedi. https://www.samedi.com/en/. Retrieved: February, 2024.

[27] J. Shaver, "The State of Telehealth Before and After the COVID-19 Pandemic", Prim Care, vol. 49, no. 4, pp. 517-530, 2022. doi: 10.1016/j.pop.2022.04.002.

[28] K. Sierra and B. Bates, "Head First Java", 2nd edition, O'Reilly Media, Inc., 2005. ISBN: 0596009208.

[29] C. L. Snoswell et al., "Determining if telehealth can reduce health system costs: scoping review", Journal of medical Internet research, vol. 22, no. 10, 2020. doi: 10.2196/17298.

[30] A. Prakash et al., "Building A Global Framework For Telehealth", Health Affairs Forefront, June 27, 2023. doi: 10.1377/forefront.20230621.134595

[31] L. A. van Dyk, "A review of telehealth service implementation frameworks", Int J Environ Res Public Health, vol. 11, no. 2, pp. 1279–1298, 2014. doi: 10.3390/ijerph110201279.

[32] "Creating a Framework to Support Measure Development for Telehealth", National Quality Forum, August 2017.

[33] D.W. Ford, J. Harvey, J.T. McElligott, and S. Valenta, "TSIM: The Telehealth Framework - A comprehensive guide to telehealth implementation and optimization", Stationery Office Books, 2021. ISBN: 9780117092969.

[34] "ACRRM Framework and Guidelines for Telehealth Services", Australian College of Rural and Remote Medicine, June 2020.

[35] TIOBE Index. https://www.tiobe.com/tiobe-index/ Retrieved: February, 2024.