Acta Cybernetica — *online–first paper version* — pages 1–28.

Standardized Telemedicine Software Development Kit with Hybrid Cloud Support*

Zoltán Richárd Jánki[ab] and Vilmos Bilicki[ac]

Abstract

In modern Web development, it is expected that systems operating in the same area can be easily integrated. For common integration points, it is recommended to use a standardized data model and a common interface during the development as this will facilitate further integrations. The use of the cloud infrastructure is increasingly popular in telemedicine, but taking into account the goals, the productivity of the development, the availability of the system and the various regulations, choosing the right solution is not trivial. Inclouded platform consists of numerous currently active telemedical microservices that are working with a common software development kit. This tool provides a standardized data model for document-oriented database systems, has support for public and private clouds by using the classic Data Access Object (DAO) analogy and contains a lot of convenient functions as well. Furthermore, it is found that our solutions can significantly increase development productivity and is confirmed by measurements taken which involved software developers.

Keywords: telemedicine, hybrid cloud, software development kit, productivity

1 Introduction

Telemedicine applications are getting more and more attention. Google Trends shows that after the appearance of coronavirus disease the number of available

*This research was supported by the EU-funded Hungarian grant GINOP-2.2.1-15-2017-00073, project no. TKP2021-NVA-09 has been implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme; project no. II-NKFIH-1528-1/2021 has been implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the II-NKFIH-1528-1 funding scheme. This study was also supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program (RRF-2.3.1-21-2022-00004).

[a]Department of Software Engineering, University of Szeged, Hungary
[b]E-mail: jankiz@inf.u-szeged.hu, ORCID: 0000-0003-1829-5663
[c]E-mail: bilickiv@inf.u-szeged.hu, ORCID: 0000-0002-7793-2661

DOI: 10.14232/actacyb.299460

telehealth applications increased, not only in the mobile stores but on the World Wide Web (WWW), too. In telemedicine, electronic healthcare records (EHRs) are considered as sensitive data, so it is really important to take into account the regulations.

Since 2018, access to patient healthcare records are governed by the General Data Protection Regulation (GDPR) [13] and individuals have a right to access their own healthcare data, but in limited circumstances they can get information about other people, too. However, the data handlers have to ensure that sensitive data cannot be transferred outside the country. Sometimes it is required to host everything within the boundaries of an organization. In some cases, it is allowed to use the public cloud but data must be stored in an encrypted form. These are limitations which can affect not only the economic solutions but can affect the development processes as well. Today, there is no publicly available software development kit (SDK) that conforms to such requirements and supports both public and private cloud solutions.

The degree of maturity of a research field can be measured by the number of available standards and protocols that belong to the given field. In telemedicine, many standards are adapted from other fields, and only a handful of them are telemedicine specific [4]. The most well-known standard is called Fast Healthcare Interoperability Resources (FHIR)[1] that provides a data model for real telemedicine use-cases. Thanks to its practical design and loose structure, it easily fits into any application.

FHIR defines only the resources that can be present in a medical environment and lists the attributes that can be used to describe these resources. FHIR itself is not appropriate to standardize every component of a telemedicine application but it has recommendations on what and how to use, so it gives vent to other standards, too.

Since FHIR is not a security protocol, it does not provide ready-to-use solutions for authorization, synchronization and digital signatures, but has recommendations. Using OAuth[2] for user authentication is suitable for web-centric applications, but the SMART-On-FHIR[3] specification can be used as an alternative solution.

Clinical terms are also managed by different standards. The most widely spread one is SNOMED CT published by SNOMED International[4]. Diagnoses, clinical documents, vital signs and other data that can be measured are systemized in the so-called LOINC[5] standard. Both standards categorize different terms with coding systems that help group data of the same type. In addition to the recommended standards, FHIR provides the opportunity to extend the predefined data model of resources with custom elements that are not originally part of the standard. However, all extensions must be well defined so that the data stored in an extension

[1] HL7. Fhir overview. https://www.hl7.org/fhir/overview.html
[2] OAuth. Oauth 2.0 - oauth. https://oauth.net/2/
[3] SMART on FHIR. Smart on fhir: Introduction - smile cdr documentation. https://smilecdr.com/docs/smart/smart_on_fhir_introduction.html
[4] International, SNOMED. Snomed - home — snomed international. https://www.snomed.org/
[5] Institute, Regenstrief. Home - loinc. https://loinc.org/

field can be easily identified. Data from special systems such as Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) systems do not have an appropriate place within FHIR, even though these concepts are critical for performing logistics and health management tasks. FHIR provides a number of extension profiles but it can be time consuming and difficult to find the one that describes the data. If there is no suitable extension profile in the standard, developers have to create a new one.

As the number of EHRs is rapidly increasing, the benefits of cloud solutions can be utilized. However, as we mentioned earlier, there can be project specific regulations that determine which type of cloud infrastructure can be used. To begin with using a public cloud, both the development and the maintenance can be convenient and the performance can be significantly high, but if there are restrictions on the location of the servers in the project, shared infrastructure cannot be an option. Private cloud is also a well-scalable solution, but the additional tasks associated with configuration and maintenance should also be considered. A hybrid solution in which a combination of private and public cloud services are available, can perform well because most of the load is on the public cloud and the critical tasks related to data and security can be handled by the private cloud.

In telemedicine, various data types can be present. An EHR can be a simple JavaScript Object Notation (JSON) object or it can be a high resolution image. In some cases, a Relational Database Management System (RDBMS) is sufficient but if the performance is critical then a non-relational database can be a better choice. Hence, it is recommended to introduce the so-called polyglot persistence concept so that we can use different data storage techniques and vary them to meet the needs. However, it is not trivial how different storages communicate with each other.

In telemedicine, offline capability can be critical. Web applications often go to offline status for seconds but occasionally they cannot come back online for hours. Besides offline status, the performance can be increased by adding caches to the data path. In our recent study, we elaborated a taxonomy for telemedicine applications and provided an easily tunable solution that helps in the design of telemedicine systems taking into account their offline capability. There should be various caching techniques that are available and finely tunable depending on the use-case.

The rest of the paper is organized as follows. Section 2 provides an overview of well-known telemedical platforms and the current status of the areas covered by our SDK. In Section 3, statistical information about FHIR and its prevalence in software development is presented. Section 4 offers a comprehensive list of challenges that developers may encounter. Section 5 introduces our SDK as an all-in-one solution to these challenges. The measurement results that demonstrate the effectiveness of our SDK in aiding telemedicine system development are discussed in Section 6. Finally, in Sections 7 and 8, we summarize the key aspects of this paper and highlight potential opportunities for further development.

2 State of the art

In this section, we present the current status of telemedicine platforms and their applied solutions focusing on the main issues that we may face.

2.1 Telemedicine platforms

- Intelehealth is an open-source telemedicine platform[6] that consists of 4 main components: a web application, an Android mobile application, a middleware layer and a medical record server. The mobile app uses complex data-gathering flowcharts and combines them to form an assisted history-taking system, called Ayu, and the web app allows remote doctors to review uploaded data and make referrals, offer advice or prescribe. OpenMRS[7] is the EHR server in this architecture that stores patient data, but it has not gained popularity in Europe and does not offer as many options as FHIR. OpenMRS can receive and convert data from FHIR-compatible systems, but it is not the main domain.
- AdvancedMD[8] is a more than 20 years old telemedicine platform. Unfortunately, it is not free and open-source but it is known that they store their data in Amazon Web Services (AWS). Due to storing data in a public cloud, this platform does not meet European regulations.
- OpenEMR[9] is a lightweight project that presents a video conferencing and chat platform for consultation purposes. Technologically its basics belong to Danphe Health that provides telemedicine softwares embedded in cloud services.
- The telemedicine network called Unimed Floripa was established in Brazil that was first introduced by R. S. Maia, et. al. [15]. Their platform serves radiological demands by maintaining a Web portal, a medical imaging toolset, teleconference tools and multiple Digital Imaging and COmmunications in Medicine (DICOM) and non-DICOM servers. It is obvious that they use standardized solutions for storing and managing data. Health Level Seven (HL7) and its standards (e.g. FHIR) play important roles in their telemedicine network. However, the capabilities of the platform are limited and too use-case specific.
- Inclouded[10] is an open-source telemedicine and smart-city platform that conforms to a number of standards. Development is based on ISO 13485, domain models follow HL7's FHIR and TMForum standards. It provides both public and private cloud solutions that are using our SDKs as connectors and

[6] Intelehealth — Confluence. URL: https://intelehealthwiki.atlassian.net/wiki/spaces/INTELEHEAL/overview

[7] URL: https://openmrs.org/

[8] Cloud-based patient relationship management software — AdvancedMD. URL: https://www.advancedmd.com/medical-office-software/cloud/

[9] URL: https://www.open-emr.org/

[10] URL: http://inclouded.hu/

helping functions with high-level FHIR support. Inclouded SDK has already performed well in many currently active telemedicine projects over the years. It is also proved that the SDK is not only a convenient tool for designing and managing telemedicine systems that can be easily integrated, but also significantly speeds up the development processes. This article provides a detailed introduction to the key components of Inclouded SDK.

Table 1 presents a comparative analysis of the five platforms that have been introduced. It is seen that none of the platforms are GDPR-compliant except for Inclouded. Private and public cloud supports vary based on their focus, but hybrid cloud support is not common. Notably, a majority of the platforms provide support for FHIR.

Table 1: Comparison of telemedicine platforms

Platform	**Open-source**	**Private cloud support**	**Public cloud support**	**FHIR support**	**GDPR compliance**
Intelehealth	✓			✓	
AdvancedMD			✓		
OpenEMR	✓		✓		
Unimed Floripa		✓		✓	
Inclouded	✓	✓	✓	✓	✓

2.2 FHIR

Before diving deep into the details of the SDK, it is important to see how popular the standard used is.

Firstly, we have analyzed the available open-source telemedicine projects. We used GitHub as a datasource because it has the biggest public repository store on the Web. It consists of millions of public projects and provides an API for filtering and gathering information regarding repositories. We wrote a GitHub Crawler for filtering metadata of the repositories and finding specific repositories based on their content. We were focusing on telemedicine projects and we were searching for projects with various keywords. By using the term "health", we found more than 100,000 repositories, but unfortunately many of them were useless due to lack of commits or completely different interests. However, we still found quite a good number of projects that really deal with telemedicine. After analyzing these repositories, we can assume that FHIR is the most popular standard used in telemedicine projects.

FHIR was first released in 2013, but its first presentation was held in 2012 [5]. First three releases were just called Draft Standard for Trial Use (DSTU), but after DSTU3, it reached a maturity level that could have been considered as

a final version. Since it is an open standard, open-source projects can describe its popularity well. For our statistics, we used GitHub as the source. Figure 1 presents that from 2014 the number of projects using FHIR started to grow exponentially and this growth is still continuing. Unfortunately, there are hundreds of repositories that contain a single readme file or just text files referring to the standard. Based on this experience, we have filtered out the GitHub repositories that contain evaluable projects using FHIR. In Figure 2, it is shown that currently R4 is the most supported version, but the number of new projects are increasing as the maturity of the standard levels up.

Today, FHIR is the most popular healthcare standard but except for some interface libraries there is no available toolkit that implements a Representational State Transfer (REST) endpoint in a typed form with FHIR support.

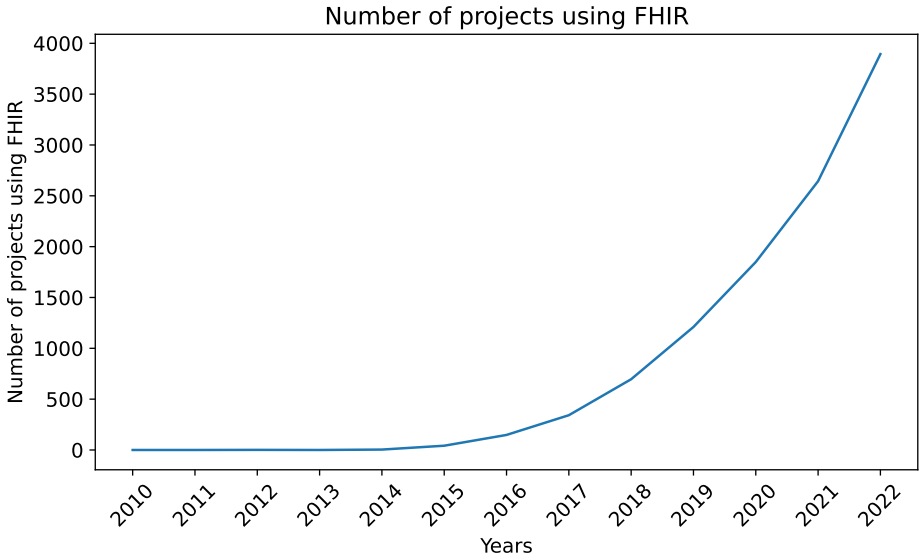

Figure 1: Popularity of FHIR in GitHub

2.3 Public and private clouds

Centering on accessing the information anytime and anywhere, encourages moving the healthcare information towards the cloud. Although the cloud offers several benefits, it also poses threats to health data in terms of privacy and security [1]. Here, we go through the pros and cons that cloud solutions provide taking into account the telemedicine use-cases.

Paper [20] presents a detailed comparison of public and private cloud solutions, highlighting the benefits of the former such as accessibility, scalability, availability, and reliability. Public cloud typically delivers the so-called pay-as-you-go model in

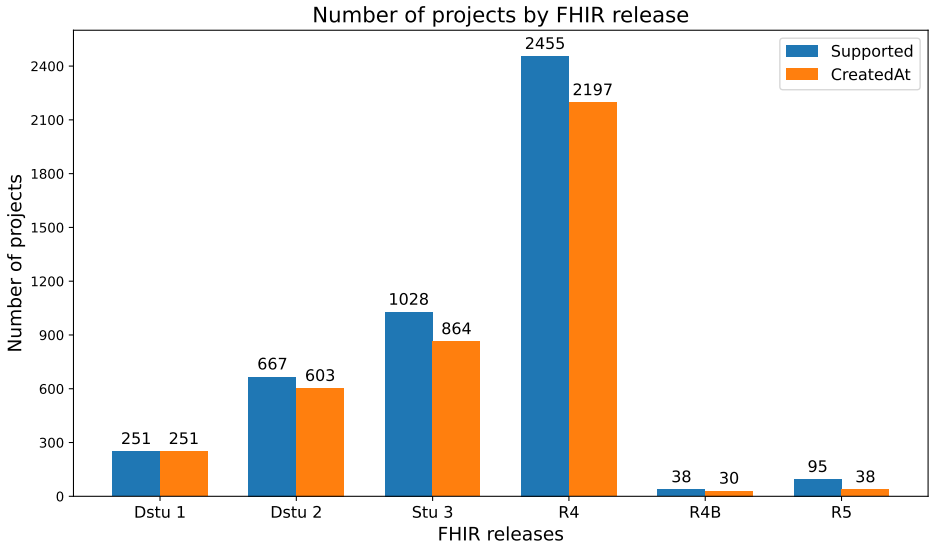

Figure 2: Popularity of FHIR in GitHub considering the version

which you pay after using the resources. Since public cloud services are ready-to-use, developers do not have to take care about the time consuming configurations and the infrastructure below the services. Public cloud services are ready-to-use, which means that developers do not need to worry about configurations and infrastructure. However, public cloud providers such as Amazon, Google, and Microsoft offer a lower level of security, so it is not recommended to store sensitive data in them.

Private cloud is usually dedicated to a single organization and it operates within the network of the organization or company. Thus it is required to buy, build and manage the cloud infrastructure that needs experts and comes with a higher cost. In terms of productivity, the development processes are longer in case of private clouds, but designed services are more use-case specific and they can operate more efficiently in given circumstances. Moreover, the level of security is higher in private clouds.

Chen et al. [12] introduced a solution that uses a hybrid cloud approach. Healthcare records stored in public clouds are encrypted with Symmetric Key Algorithm (SKE) and can be decrypted only through the private content key. Their solution is a redundant hybrid cloud service that is offered at the cost and scale benefits of public clouds, while also offering the security and control of private clouds. Nowadays, this approach is more and more common in telemedicine projects but there is no available library that supports hybrid cloud approach with interchangeable cloud background.

2.4 Serverless development

Startups and smart tech companies have begun to take advantage of serverless scalability, reliability, and performance for rapid growth - and now serverless development is more popular than ever. Moreover, it is also found that the developer productivity can be increased as well. Here, we consider the word "serverless" as a service in which the infrastructure is maintained by the service provider. There are various serverless services, such as databases, storages, runtime environments that can be used to run computational tasks and host Web applications. Vadym Kazulkin [11] collected the main effects that serverless development can have on productivity. The main advantages are the followings:

- no infrastructure maintenance
- auto-scaling and built-in fault tolerance
- less engineers required
- less code written
- bigger focus on business value and innovation
- shorter time-to-market procedures

ip.labs has been following the concepts of serverless development for years but in a hybrid form. They still have monolithic Java applications, but two teams are developing completely serverless. Before they went serverless, they had a central administration team and prioritized the tasks that lead to increased waiting time. Serverless development needs no low level administration at networking level, there are no servers and no operating systems (OS) that developers have to take care about and there is less interaction between developer and administration teams. Thus, the development processes become faster, developers write less code and use managed services.

2.5 Productivity

In Information Communication Technology (ICT) productivity is measured in different fields. Here, we focus on developers' productivity, show which metrics are used to measure it and how new techniques and technologies affect productivity in software development.

Shake [19] collected the most important metrics that can describe the workflows and measure the productivity of development. Firstly, the code quality is a big hit to productivity. There are several metrics to measure code quality and reduce quality defects. Shake is a good tool to create automatic reports that help deal with bugs efficiently.

Code coverage is a valuable metric for monitoring the development team's testing activities. It is easy to measure since it has a concrete formula and it gives feedback about how much of the source code is not covered by test cases. These metrics do not measure productivity explicitly, they have only effects on it.

Cycle time can tell a team a lot about the productivity of developers. It measures the time taken for a task to move from one phase to another. Cycle time is broken down into more stages and it gives information about which stage is problematic and where bottlenecks are. Most of the issue and project tracking softwares provide data about cycle time, so it is a more and more common metric used to measure productivity.

Lead time is very similar to cycle time, but it is a more comprehensive metric that measures productivity from task creation until delivery. So it is a metric for not individuals but for the development team.

Deployment frequency is measured within a specific period of time and it is one of the most valuable metrics in terms of productivity. There are 4 software delivery performance levels: low, medium, high and elite. This level is specified by the deployment frequency. Software delivery performance is low if deployment frequency is fewer than once per six months and it is elite if there are multiple deploys per day. Flickr reported an average of 10 deployments a day in 2009, while Etsy had 11,000 deployments in 2011 [18]. At Facebook each developer released an average of 3.5 software updates into production per week. These numbers prove that Continuous Integration (CI) and Continuous Delivery (CD) can significantly improve productivity.

The DevOps Research and Assessment (DORA) group published their platform and introduced 4 key metrics for measuring DevOps performance. These metrics are deployment frequency, lead time for changes, change failure rate and mean time to recover. Using this platform, Fin500 was able to increase the number of releases to production from 40 to over 800 [6].

The above mentioned tools can show only an approximation for the productivity, based on the committed source codes and the logged work hours, so they do not perform in a way that produces exact results. We found that using our SDK can significantly improve developers' productivity in telemedicine applications and it is confirmed with metrics too that are measured based on the developers' activity in the preferred integrated development environment (IDE).

3 Actuality of using FHIR

To prove that using FHIR is a trend, we analyzed the publicly available telemedicine projects in the world and measured the presence of the standard. We used GitHub as the main source of our research and using its API, we have collected the repositories having telemedicine purposes and repositories using FHIR.

We started using general expressions (e.g. health) to find the most repositories of our interest, but we realized that only a very small part of the results would be useful for us. Due to the limitations of GitHub API, we had to choose the terms carefully and analyze the repositories focused on the results.

On GitHub, there are thousands of empty or almost empty repositories that can be easily found using an expression that is present in its name or description or in a readme file. The valuable repositories may contain source codes too that

can be analyzed and later compared to each other by using code metrics. GitHub has several categorizations for the repositories, and most of them are supported by the API, too.

Based on our experiences, we searched for FHIR-related repositories on GitHub using search terms listed in Table 2 and filtered by 5 main programming languages: Java, C#, Python, JavaScript, and TypeScript. Table 3 shows the most popular packages for these languages that have references to FHIR. HAPI[11] is an open-source FHIR server written in Java, Firely SDK[12] is the official .NET SDK written in C#, FHIR Resources[13] is a Python package for creating and validating FHIR objects, and `ts-fhir-types` [2] is a TypeScript package for FHIR resources.

Table 2: GitHub crawling terms and the number of found repositories

Term	Number of repositories
telemedicine	780
e-health	167
ehealth	703
telehealth	350
teledermatology	5
teleradiology	20
teleeducation	2
healthcare	1795

Table 3: Number of repositories retrieved from a product-based GitHub crawling

Programming language	Product	Number of repositories
Java	HAPI server	780
C#	Firely .NET SDK	167
Python	fhir.resources	703
JavaScript/TypeScript	@ahryman40k/ts-fhir-types	62

Further analysis was made by filtering the results using the selected 5 programming languages. Firstly, we have inspected how popular FHIR is on GitHub. If we check the repository names, descriptions and readme files, we can find 5,931 repositories. Comparing this number to the number of retrieved results if we make a code-based search, it is very few. The number of GitHub code-based search results

[11] Hapi FHIR — The Open Source FHIR API for Java. URL: https://hapifhir.io/hapi-fhir/

[12] Firely .NET SDK — The Official .NET SDK for Hl7 FHIR. URL: https://fire.ly/products/firely-net-sdk/

[13] URL: https://pypi.org/project/fhir.resources/

shows how many files were found on GitHub containing the term. Usually, it is much more than the number of repositories, so these results need further analysis. We made a code search for the FHIR term, but due to the high number of results (2,011,239 occurrences found), we limited them using the 5 language filters. When TypeScript is selected as the main language, GitHub returns thousands of files that contain FHIR, but after forming a set from the repositories, only 277 repositories were left in the end. C# shows similar behavior because code search found about 50,000 files on GitHub but there are only 257 repositories. Python and JavaScript seem to be more popular in project development using FHIR. There are more than 500 public repositories that have Python and JavaScript as the main languages. Java is the most popular with 1,679 different repositories. Figure 3 shows the ratio of used main programming languages in FHIR repositories. There were 3,392 repositories found using FHIR and the 5 selected languages. It is also found that using techniques, frameworks and components like search terms can produce more focused and more valuable results in such data mining.

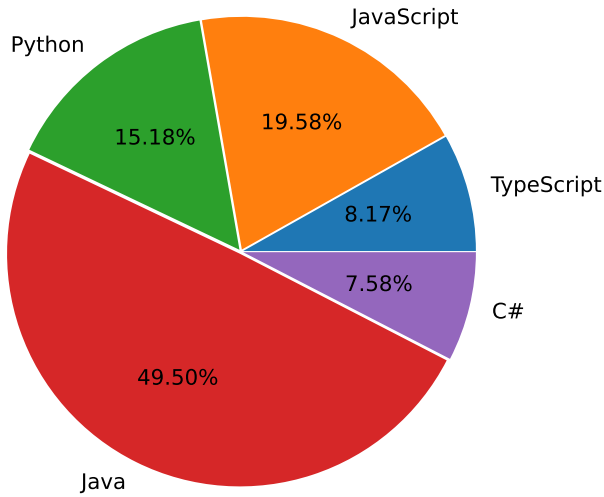

Figure 3: Presence of FHIR on GitHub by programming language categories

It is observed that FHIR is popular in application development but it is not clear how many telemedicine projects use FHIR. Telemedicine repositories were selected by using code search with the 8 search terms listed in Table 2 and filtered by the 5 chosen languages. Figure 4 depicts that the expression "healthcare" is present in most of the repositories. We found projects from specific areas of telemedicine too but they do not exceed 1% of the total together.

We have also collected the repositories that use the most common products listed in Table 3. After seeing that most of the telemedicine-related projects use Java, it

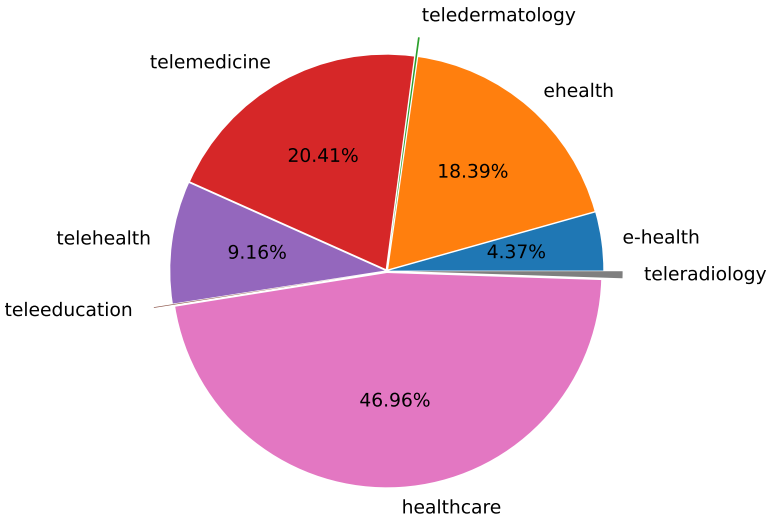

Figure 4: Presence of telemedicine on GitHub based on search terms

seemed to be obvious that the search term "HAPI" will return the most records. After HAPI, the official Python package is the most commonly used resource. Firely SDK showed a surprisingly significant popularity with 19.46%. Compared to the number of available public repositories, only 62 (4.68% of) repositories use the ts-fhir-types package written in TypeScript. The total number of repositories using the 4 packages was 1,326. Figure 5 shows the ratios.

To see how many telemedicine-related projects use helper packages, libraries and official solutions, we have inspected the intersections of these sets. We found that 35% of TypeScript projects using FHIR rely on the ts-fhir-types package. However, this typed version is not so popular in JavaScript-based projects. Unsurprisingly, HAPI and fhir.resources are used in more than 50% of projects in which Java or Python are the main programming languages and FHIR is present, too. The results are presented in Figure 6.

Finally, we found it is important to see how FHIR affects the lifetime of projects. We measured the freshness of the projects by applying a threshold for the last commit date. We filtered out repositories with a last commit date older than 3 months and an average size below 68,852 KB. This approach helped us identify repositories with a high maturity level. Naturally, in order not to distort the statistics, we applied the language filters, too. It came out that in mature projects FHIR is really popular, almost 50% of the projects use it as a standard (Figure 7). Here, we found only 326 repositories. Based on this experience, we evaluated the intersection of the FHIR set and all other telemedicine sets. It is seen that in the retrieved telemedicine projects with high levels of maturity, FHIR is commonly

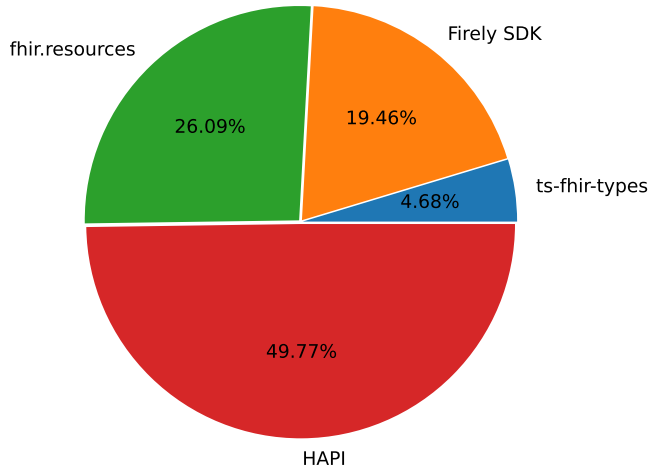

Figure 5: Presence of most common FHIR products on GitHub

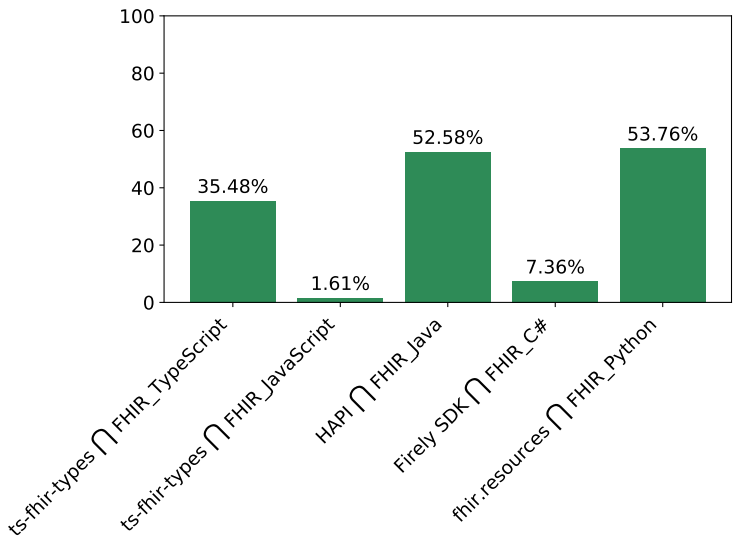

Figure 6: Intersection of most common products and projects using FHIR with main programming languages

applied (Figure 8), but some projects have unique data models while others show similarities to the FHIR model.

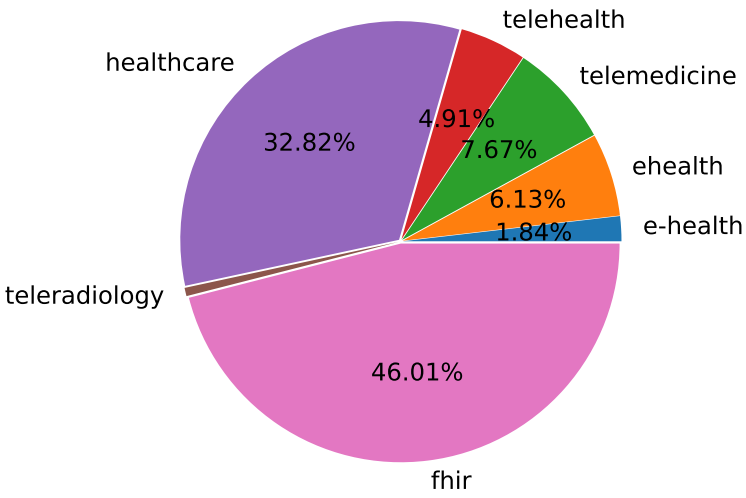

Figure 7: Presence of telemedicine repositories filtered by threshold values

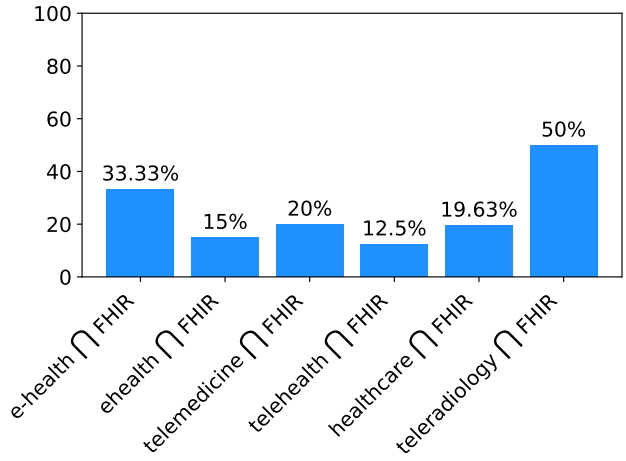

Figure 8: Ratio of telemedicine projects using FHIR filtered by threshold values

4 Challenges in telemedicine application development

Here, we collected the challenges that we were facing during the development of telemedicine applications and the Inclouded platform. To ease the development processes, we have elaborated and implemented an SDK that supports new technologies, provides extra features that help developers and solves complex problems as well. In this section, we list the challenges that our SDK provides solutions for. We will present our solutions in detail later.

- Inclouded supported FHIR since the foundation of the platform but only relational database systems were used prior. As public cloud solutions became more and more popular, we found that Not-only Structured Query Language (NoSQL) databases can perform better in many situations. FHIR offers a relational data model for handling healthcare resources, so it is a challenge to have a compatible NoSQL solution, too.
- Using a pre-created domain model, it is not trivial to find the proper entities and fields to store all necessary data. Standards are sometimes too generic even if they are practical. FHIR provides a lot of healthcare resources with a well-defined data model, but in many real telemedicine cases, it is hard to find the place to store the data. FHIR offers an extension mechanism for such cases but the extensions must contain a precise description of what they contain. Our SDK was extended with a Natural Language Toolkit-based (NLTK) recommendation system that helps to find the best matching extension for the data to be stored.
- FHIR provides a well-defined domain model but there is no recommendation on what technologies to use. Since a telemedicine application can contain not only metadata about healthcare records but binary files too, we decided to pursue the idea of polyglot persistence where we use different database systems, but each of them is used for what they are best at.
- In 2016, Google introduced Angular 2 framework that brought a big change after AngularJS [22]. As they recommended using TypeScript programming language, everyone felt the lack of a typed version of FHIR client library. `fhir.js`[14] offers an official solution for using FHIR in JavaScript but it was not extended to handle interfaces and classes.
- In many telemedicine projects it is limited where data can be stored and what path data can be transferred through. These limitations can be stated by owners, organizations or a project. To meet these requirements and run into less legal issues, a private cloud is recommended to establish. Inclouded SDK supports hybrid cloud solutions, so developer can choose to use public or private cloud to store the data.

[14] URL: https://github.com/FHIR/fhir.js/

5 Our solution

This paper introduces our telemedicine SDK called Inclouded SDK, its importance in telemedicine application development and all of its features. The basic concept of our SDK is to provide the capabilities of WebDAO for telemedicine application developments. Using WebDAO analogy, SDK offers Data Transfer Objects (DTOs) in form of classes to apply the entire DAO design pattern. Since it is shown that serverless development increases productivity, we decided to start telemedicine developments in a public cloud. Google Cloud Firebase platform and its services were used as a basis, so we built an SDK that can handle FHIR and can conform to the solutions of Firebase. Here, we used Google Cloud Firestore for storing metadata, Google Cloud Storage for storing binary files and the Google Authentication service for managing users. The SDK is publicly available and installable via Node Package Manager (NPM).

5.1 SDK structure

As it is shown in Figure 9, Inclouded SDK consists of resource-based application programming interfaces (APIs) that provide fully FHIR-compatible typed document classes, so-called DTOs and a list of queries that implements the necessary FHIR search parameters in form of independent functions. The basic Create, Read, Update, Delete (CRUD) operations are implemented in the FhirApi class and all the FHIR resources are inherited from this class. Besides the CRUD operations there is an additional id-based query function that is suitable for all resources. Hence, Inclouded SDK fully implements the DAO design pattern for FHIR and can be used as a part of the data layer of Clean Architecture [16]. To make Figure 9 more clear, we have only drawn a part of the whole library but the rest of it uses the same idea.

During the design of SDK, we took into account that FHIR has never had NoSQL support, however, Firebase provides only NoSQL database systems. FHIR was designed for RDBMS, but today NoSQL is gaining more and more space. FHIR defines search parameters that describe the necessary filters if one uses the standard. Using Google Cloud Firestore we were facing issues that made it hard to filter data that FHIR supports. Four main problems were:

1. finding a value of a field if it is in an object inside an array,
2. filtering by substrings of a field,
3. extending queries based on access-control rules,
4. and obtaining results that are gathered from multiple collections.

In Section 6, we show our algorithms for these problems and present a logical model for their specifications and verifications.

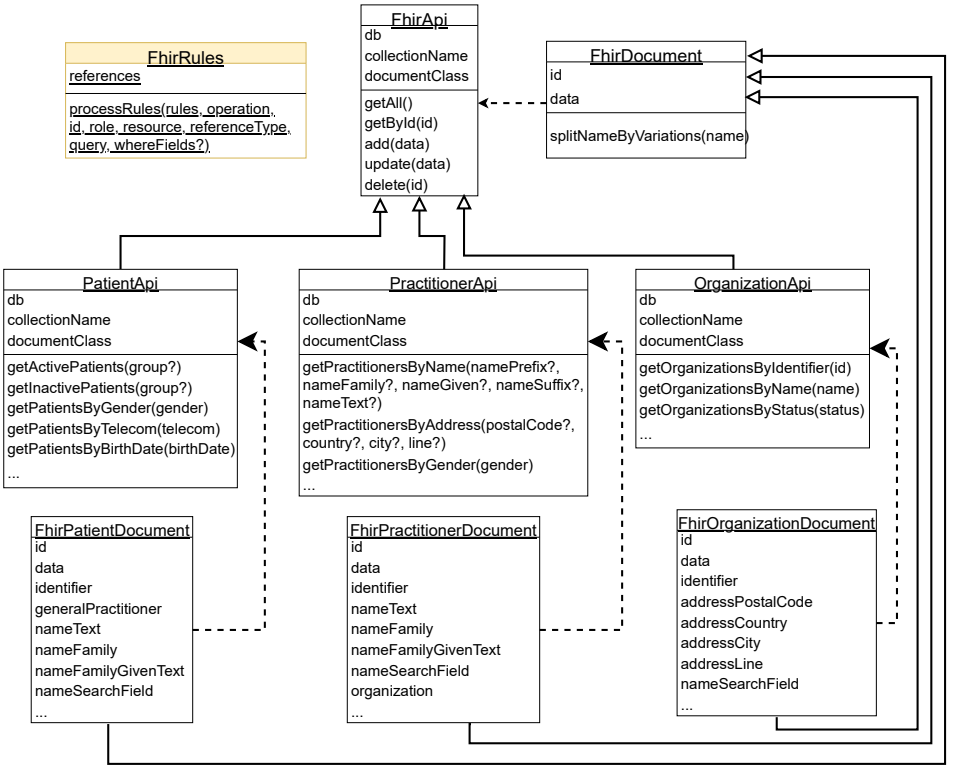

Figure 9: SDK structure

5.2 SDK architecture

Since FHIR requires endpoints that return back standard data, there is no restriction on the format in which the data is stored. So, the data is stored in a format compatible with NoSQL and Firestore concepts, but the data that is retrieved by the client is fully FHIR compliant. The above mentioned intelligent capabilities are implemented in the document classes. FhirDocument is the base class of all resource entities. Here, we have two attributes, one is id, the other is data. To solve the 4 main challenges, document classes use helper functions that convert data to a format that makes it searchable in any database system.

Figure 10 shows an architectural map about how SDK takes place in the data path. It is installed on the client side and processes the data sent by the client and forwards it to the cloud. If the client operation is an insertion, SDK waits for an FHIR-compatible object and creates an extended object that makes the original data NoSQL compatible. If the request is a query, then SDK waits for search parameters and adds the necessary filters to the query object. Since access control techniques are different in different database systems, SDK also has an optional rule

processing function that checks the grants based on the database settings. Since the rule system of Firestore does not work as a filter, it is required to add extra filters to the query to retrieve the needed objects.

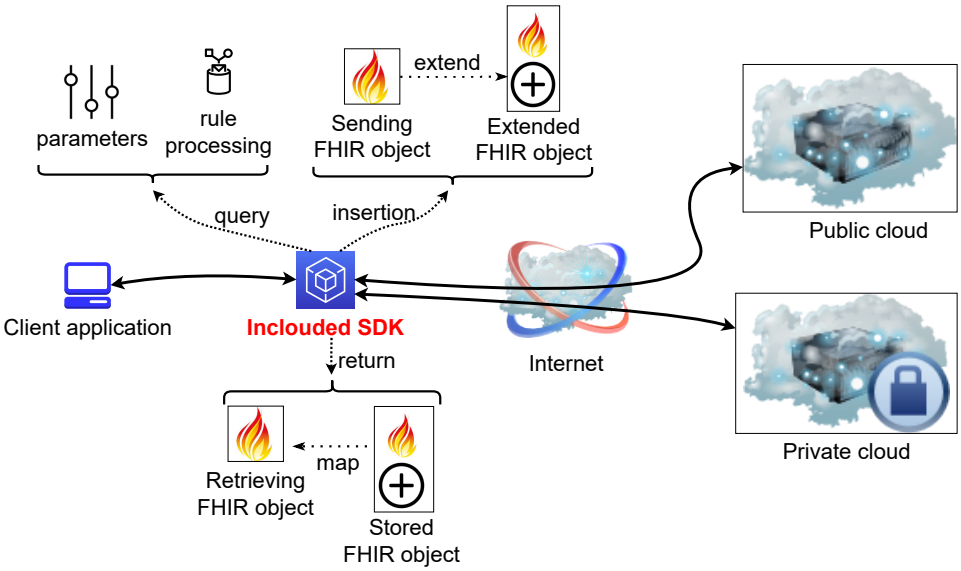

Figure 10: SDK architecture

6 Results

In this section, we will present the main components of our SDK, the formal descriptions of the algorithmic solutions, and the productivity results achieved.

6.1 Main components

Inclouded SDK consists of six core components that support developers in telemedicine application development. Here, we provide a detailed description about these solutions.

6.1.1 Outsourcing non-searchable fields

It is shown that in Google Cloud Firestore, if a field can be found in an object inside an array, it cannot be filtered. It is also problematic in other NoSQL systems if only a field value is known, not the whole object. SDK manages such cases by outsourcing these values into new so-called search-fields created at the top level. The standard form of the data is also kept, so REST endpoints conform to the FHIR standard.

6.1.2 Rule processor

Access control mechanism of Google Cloud Firestore is so simple that only requested collections or one document of a collection can be controlled by them. Rules can use the requester's sent data, requested data or a preset date to check if the resource can be given to the client. Since a rule controls the whole request, it is not an option to retrieve only those documents for which we have permission. To do so, requests must contain additional filters to start requests only for those documents that we have permission. If we construct our queries using this structure, we can execute queries in any NoSQL system. Thus, we elaborated a rule processor algorithm. Since it is known which FHIR resources can contain data referring to users, groups or permissions, we made a built-in resource descriptor object that contains the resources and their fields that may contain references to such entities. The rule processor waits for a query object, the currently active rule set and the logged in user's id, roles and groups. Optionally, the FHIR resource fields that may refer to permissions can be explicitly set up. By default, SDK uses the basic FHIR knowledge. After starting a request, – calling an SDK function, – SDK will extend the query based on the active rules, the user data, the requested resource and the FHIR resource references. So, developers do not have to take care about how data can be retrieved under given access control settings because SDK will resolve this issue and build up a perfect query. Naturally, the rule processor functionality can be turned off if there are no rules set up in the project. This solution can be applied in private cloud solutions, too, e.g. in RESTHeart[15] with MongoDB.

6.1.3 Collected system codes

FHIR stores quantitative values and values from enumerations in coding systems. SDK collects the most important and most common codes from the LOINC database that helps to describe stored data. This component not only provides the codes and their descriptions, but also creates the necessary FHIR format of the object that will contain the value. Developers can waste a lot of time by searching for these codes and finding the best description.

6.1.4 Extension finder

FHIR has an extension mechanism to give the opportunity to place data at a resource if there is no given field for the data. However, it is not easy to use because extensions must be well-defined using a FHIR profile that describes the stored data. We have extended our SDK with a public REST endpoint that can return a suggestion for data that developers could not find a field in the standard. The idea came after analyzing open-source projects using FHIR and applying extensions in the wrong way. Our approach uses NLTK to find the best matching extension that can define the data. In [10], we have shown our component in detail and demonstrated that our solution can achieve an 89% success rate.

[15] SoftInstigate. Restheart - ready to use backend for web and mobile apps. https://restheart.org/

6.1.5 Support for offline capability

In [8] and [9], we have presented a taxonomy to help design distributed telemedicine systems. Based on our elaborated taxonomy, we showed how consistency and data quality changes if the data path is complex and how systems can be configured to maintain consistency, availability and partition-tolerance at a high level. Inclouded SDK took a part in that model, and it has the option to easily tune a telemedicine system so that it remains offline capable without significant data staleness. It was measured that by allowing data up to 1 version older to be used in the cache, the system can still provide 83% consistency.

6.1.6 Hybrid cloud support

One of the biggest advantages of using Inclouded SDK is the hybrid cloud capability. It is a requirement in many projects to store data in a private cloud. However, public clouds can perform better. The development of Inclouded SDK started in 2016 and was introduced first in our former paper [7]. That study examined the concept of WebDAO and highlighted the importance of DAO in telemedicine application development. Based on our experiences in using Google Cloud, we have implemented a DAO layer that can substitute Google's document classes. It is a modern implementation of the classic DAO layer. Since Google Cloud Firestore is a document-oriented NoSQL database system, we found MongoDB as the closest open-source alternative to build a private cloud. Comparing their features, they operate very similarly, only technical differences can be found. Google is a bit more limited in filtering and setting up rules to access resources.

RESTHeart is an open-source cloud platform that provides REST API for MongoDB but cannot notify clients about data changes in real time via REST API. Since RESTHeart HyperText Transfer Protocol (HTTP) endpoints close the connection between the client and the server after responding, a WebSocket connection is needed to keep the connection alive. We have integrated a WebSocket module into Inclouded SDK that can establish WebSocket connection to a server. To follow up real time changes of a MongoDB collection, a Change Stream must be opened but Change Stream requires a MongoDB Replica Set that is not part of the basic RESTHeart platform. Hence, we have extended the original RESTHeart project with a MongoDB Replica Set and added a WebSocket server that can open Change Streams to collections. The WebSocket connection initiated by the SDK is used only for notifying the subscribed clients about changes. Every request goes through the original RESTHeart API.

Inclouded SDK can transform all type of queries that Google Cloud Firestore can handle including CRUD operations, filterings, ordering and paging. To have an interchangeable solution, we have created a MongoDBCollection, a MongoDBDocument and a MongoDBQuery classes that have the same functions as Firestore's TypeScript classes have, with the same input arguments and return values. Thus, a configured Firestore database object and a configured MongoDB DAO object can be interchangeably used. The database object is an input of FHIR API classes, so

developers can decide if a resource should be stored in a private cloud or in a public cloud.

6.2 Formal definitions of algorithms

In this section, we present our algorithmic solutions for the four main issues that we were facing by using NoSQL database systems. For three of them, we provided an algorithmic solution. In the fourth case, if data can be queried only from multiple collections, developers can start multiple queries to get the needed data but it can produce a huge load on the client side. A better approach is to collect the data based on use-cases in result tables. SDK supports two types, these are used for creating result tables and charts. These are implemented in independent functions, so here we do not provide an algorithmic solution.

Since FHIR was designed for relational database systems, its applicability in public cloud databases is limited due to their predominant use of NoSQL database systems. Glenn Pepito collected the challenges and strategies of RDBMS to NoSQL migration in [17]. In a recent ScyllaDB guide [21], it is detailed what trade-offs must be taken when changing from relational to non-relational database system. Alachisoft[16] collected the key steps for adapting an existing schema to non-relational databases. All in all, it is commonly recommended that during the data model transformation process, denormalization and embedding of referenced objects into the reference location should be employed in most cases. However, in some instances, a hybrid model may be more effective. Similarly, our algorithmic approach also follows a hybrid principle in structuring data by preserving the standard part intact but outsourcing specific fields due to limitations in filtering capabilities.

Three algorithmic solutions were modeled in Temporary Logic of Actions (TLA) using its TLA+ language [14] to provide formal definitions as well. We have also verified the correctness of algorithms with Temporary Logic of Components (TLC) model checker. The algorithms and their formal definitions for the mentioned issues are as follows:

- If a field that must be searchable by the standard is hidden in an object that is in an array, the field is outsourced to an independent field at the top level (Algorithm 1).
- If a field containing a string must be filtered by substrings, SDK generates all the possible variations of the string that may occur and place them in an independent array field at the top level (Algorithm 2).
- If a rule exists for a given resource, it must be extended in the query to return back data (Algorithm 3).

[16] Alachisoft. Migration from sql to nosql databases. https://www.alachisoft.com/resources/whitepapers/sql-to-nosql-migration.html#json-collections

Algorithm 1 Formal definition of "checking if object is in an array" algorithm

$CheckObjInArray(x)$

1: **if** $num\_op[x] < Len$(INPUT_OBJECT_FOR_OBJ_IN_ARRAY) **then**
2: $num\_op' = [num\_op$ EXCEPT $![x] = num\_op[x] + 1]$
3: $head' = Head$(check_arr)
4: **if** $head'! =$ "elementary" and $Head(head')! =$"object" **then**
5:
6: **if** $Len(head') > 0$ and $Head(Head(head')) =$"object" **then**
7: $found\_arrays' =$ TRUE
8: $array\_counter' = array\_counter + 1$
9: UNCHANGED $substr\_filter\_vars$
10: **else**
11: UNCHANGED $<<$ $array\_counter$, $substr\_filter\_counter$, $found\_arrays$, $substr\_filter\_needed$, $check\_substr >>$
12: **end if**
13: **else**
14: UNCHANGED $<<$ $array\_counter$, $substr\_filter\_counter$, $found\_arrays$, $substr\_filter\_needed$, $check\_substr >>$
15: **end if**
16: $check\_arr' = Tail$(check_arr)
17: **else**
18: UNCHANGED $vars$
19: **end if**

Algorithm models were verified if they work properly. We have developed a Generator API to all the FHIR Resource APIs and these generators produced inputs that were passed to TLC Model Checker. We have evaluated the state graph of the algorithms but none of them produced error or deadlock, and returned the expected values, so we can conclude that algorithms are working as expected.

6.3 Productivity

In addition to many features that Inclouded SDK carries, we have also taken measurements on how it influences the development productivity. We have seen that there are code and project metrics that can be used to measure productivity. Here, we introduce another technique that measures specifically the coding and its progress. After testing various tools, we found an open-source, cross-platform time tracker for operating systems that can profile to output only the time used for development. Automatic, rule-based time tracker (ARBTT[17]) is a completely automatic time tracker that can collect statistics about how users spend their time. It runs in the background and monitors the computer and saves statistics about

[17] Breitner, Joachim and et al. arbtt: the automatic, rule-based time tracker. https://arbtt.nomeata.de/#what

Algorithm 2 Formal definition of "substring filtering needed" algorithm

$CheckIfSubStrFilterNeeded(x)$

if $num\_op[x] < Len$(INPUT_OBJECT_FOR_SUBSTR_FILTER) **then**
  $num\_op' = [num\_op$ EXCEPT $![x] = num\_op[x] + 1]$
  $head' = Head$(check_substr)
  if $head'! =$ NEEDED and $Head(head') > 0$ **then**

    if $Head(head') =$NEEDED **then**
      $substr\_filter\_needed' =$ TRUE
      $substr\_filter\_counter' = substr\_filter\_counter + 1$
      UNCHANGED $obj\_in\_array\_vars$
    else
      UNCHANGED $<<$ $array\_counter$, $substr\_filter\_counter$, $found\_arrays$, $substr\_filter\_needed$, $check\_arr >>$
    end if
  else
    UNCHANGED $obj\_in\_array\_vars$
  end if
  $check\_substr' = Tail$(check_substr)
else
  UNCHANGED $vars$
end if

Algorithm 3 Formal definition of "rule processor needed" algorithm

$CheckRules(x)$

if $num\_op[x] < rules\_length$ **then**
  $num\_op' = [num\_op$ EXCEPT $![x] = num\_op[x] + 1]$
  $check_rules' =$RULES[RESOURCE]
  $head' = Head$(check_rules')
  if $Len(head') > 0$ **then**
    $query' = query$ o $<< head' >>$
  else
    FALSE
  end if
  UNCHANGED $obj\_in\_array\_vars$
  UNCHANGED $substr\_filter\_vars$
  UNCHANGED $rules\_length$
else
  UNCHANGED $vars$
end if

what windows were open, which one was the most active one in a given interval. The interval can be configured before starting the tracker.

In our study, we involved 10 university students who have not met FHIR yet, but completed a Web-development frameworks course where they learnt about Angular 2+ framework and Google Cloud Firebase platform and its services. With this study, our goal was to measure how productivity can be increased if developers use Inclouded SDK instead of start using the documentation of FHIR and the original Firestore SDK. The development phases were the followings:

1. Create an example TypeScript object for the selected FHIR resource.

2. Implement a list of Firestore queries without using Inclouded SDK (CRUD operations and other queries taking into account the FHIR search parameters).

3. Implement a list of Firestore queries using Inclouded SDK (same list of functions).

ARBTT was started with *-r 10* argument, so after every 10 seconds a log was created in the log file containing the opened windows and puts a flag to the most active one. Every developer used Visual Studio Code as IDE and installed the Angular 13 Snippets extension in advance. In the developer's ticket, it was specified what name they have to use by creating the file for the functions, so after analyzing the logs it was easy to determine how much time they spent editing a given file in the project. Everyone started to work on the same Angular 13 project that contained the necessary packages with fixed version numbers. The task list and the order of the tasks were identical, only the operating system was permitted to choose after the preferences. To verify the accuracy of the ARBTT capture logs, we analyzed the work logs added to the tickets as part of our quality control measurements.

In Figure 11, it can be seen that in all 4 scenarios the development time is reduced if developers used SDK. The development time in hours is presented as an average for each FHIR resource. The time tracker puts a flag to the window with the highest activity within the last 10 seconds, so it is not obvious how long the development time really took. We have also validated the time tracker results with logged work hours, and we found similar ratios between the two types of development form. After the final analysis of ARBTT logs, we found that the average activity time of a window in a 10 seconds long interval is 3.51 seconds. Since Patient was the first resource that developers had to work with, it needed the most time. Moreover, Patient has the most search parameters as well, so it needs the longest development time period. Comparing the development times, we can say that the development with SDK can be at least 2 times better than using only official documentation with no helper functions. Thus, we found that the Inclouded SDK can be an important key element not only in ours, but also in other telemedicine architectures. Moreover, these measurements validate the importance of DAO pattern from the point of view of productivity as well.

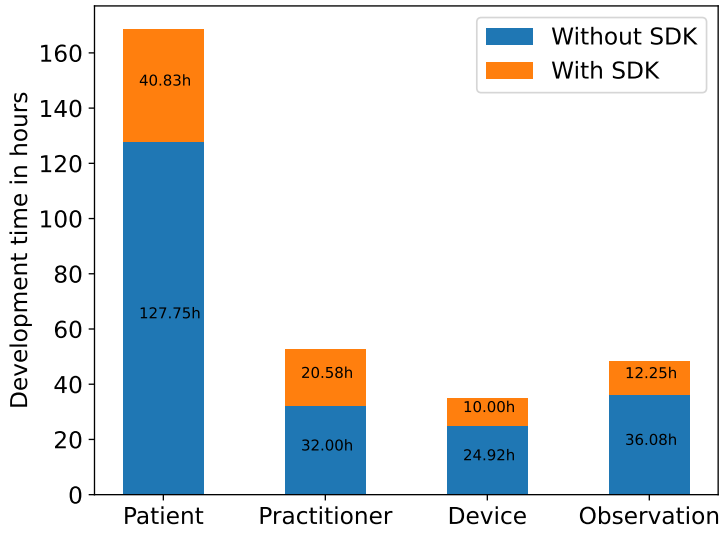

Figure 11: Average development time measured using SDK and without SDK

7 Future plans

Inclouded SDK is constantly updated and it is following the innovations of the dependencies. It is planned to integrate more public cloud solutions to support various systems. With these integrations, more novelties can be added to the package. After comparing our Google-based solution to Amazon Web Services and Azure, we found that all three platforms have common key points that make it possible for our SDK to be compatible with all public cloud platforms. Naturally, we would like to provide further support for private clouds as well. Regarding private cloud solutions, our solution primarily supports NoSQL databases. Since the filtering capabilities are more limited compared to a relational database, our solution can be clearly adapted to support relational databases as well. Nevertheless, FHIR defines a relational domain model, so such a solution can be implemented without the algorithmic solutions we proposed. Our solution and its significance came up with the idea to support other standards and may focus on other areas as well, not only on telemedicine. We have already started to develop a SDK with similar capabilities supporting telecommunication projects and using an acknowledged standard called TM Forum. In summary, there is a planned effort to assess the productivity of GitHub projects, supporting the importance of the WebDAO pattern as presented in [3].

8 Conclusions

This paper presented Inclouded SDK that can be a key component of any telemedicine system. It acts as a link between client-side and server-side in a way that the backend system can be easily changed. Both private and public clouds are supported, furthermore it contains several functionalities that help developers to work efficiently. Here, we support the most popular telemedicine standard, and we made various statistics to show its actuality. We presented the five main components of the SDK, in which the algorithmic solutions were formally defined and verified as well. The significance of Inclouded SDK in telemedicine application development is proved with different measurements. In terms of productivity, we have shown that the required development time can be at least two times less with SDK than without using SDK. Our results based on GitHub analysis and productivity showed that it is a promising solution. It is open-source and publicly available in NPM, and the increasing number of downloads denotes that there is a growing demand on packages and libraries like this.

References

[1] Abbas, A. and Khan, S. A review on the state-of-the-art privacy preserving approaches in e-health clouds. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1431–1441, 2014. DOI: 10.1109/JBHI.2014.2300846.

[2] Baudin, G. Handle FHIR objects with TypeScript (and JavaScript). URL: https://medium.com/@ahryman40k/handle-fhir-objects-in-typescript-and-javascript-7110f5a0686f. [Accessed: 2022-09-29].

[3] Choudhary, S., Bogart, C., Rosé, C. P., and Herbsleb, J. D. Modeling coordination and productivity in open-source GitHub projects, 2018. URL: http://reports-archive.adm.cs.cmu.edu/anon/isr2018/CMU-ISR-18-101.pdf.

[4] Ferrer-Roca, O. *Standards in Telemedicine*. In *E-Health Systems Quality and Reliability: Models and Standards*, pages 220–243. Medical Information Science Reference, 2011. DOI: 10.4018/978-1-61692-843-8.ch017.

[5] FHIR version history and maturity. Technical report, The Office of the National Coordinator for Health Information Technology. URL: https://www.healthit.gov/sites/default/files/page/2021-04/FHIR%20Version%20History%20Fact%20Sheet.pdf.

[6] Forsgren, N., Tremblay, M., Vander Meer, D., and Humble, J. DORA platform: DevOps assessment and benchmarking. In *Proceedings of the International Conference on Design Science Research in Information System and Technology*, pages 436–440, 2017. DOI: 10.1007/978-3-319-59144-5_27.

[7] Jánki, Z. R. and Bilicki, V. Full-stack FHIR-based MBaaS with server- and client-side caching capable WebDAO. In *Proceedings of the 11th Conference*

of PhD Students in Computer Science, pages 179–183, 2018. URL: https://www.inf.u-szeged.hu/~cscs/cscs2018/pdf/cscs2018.pdf.

[8] Jánki, Z. R. and Bilicki, V. Crosslayer cache for Telemedicine. In *Proceedings of the 12th Conference of PhD Students in Computer Science*, pages 159–163, 2020. URL: https://www.inf.u-szeged.hu/~cscs/cscs2020/proceedings.php.

[9] Jánki, Z. R. and Bilicki, V. Taxonomy for trade-off problem in distributed Telemedicine systems. *Acta Cybernetica*, 25(2):285–306, 2021. DOI: 10.14232/actacyb.290352.

[10] Jánki, Z. R. and Bilicki, V. Domain specific semantic data model integration. In *Proceedings of the 13th Conference of PhD Students in Computer Science*, pages 197–201, 2022. URL: https://www.inf.u-szeged.hu/~cscs/cscs2022/pdf/cscs2022.pdf.

[11] Kazulkin, V. Measure and increase developer productivity with help of Severless. URL: https://www.slideshare.net/VadymKazulkin/measure-and-increase-developer-productivity-with-help-of-severless-by-kazulkin-and-bannes-sla-the-hague-2020-238115659. [Accessed: 2022-09-29].

[12] Kruse, C. S., Smith, B., Vanderlinden, H., and Nealand, A. Security techniques for the electronic health records. *Journal of Medical Systems*, 41(8):127–136, 2017. DOI: 10.1007/s10916-017-0778-4.

[13] Kulakiewicz, A., Parkin, E., and Powell, T. Patient health records: Access, sharing and confidentiality. Technical report, House of Commons Library, UK Parliament, 2022. URL: https://researchbriefings.files.parliament.uk/documents/SN07103/SN07103.pdf.

[14] Lamport, L., Matthews, J., Tuttle, M., and Yu, Y. Specifying and verifying systems with TLA+. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, pages 45–48, 2002. DOI: 10.1145/1133373.1133382.

[15] Maia, R., Von Wangenheim, A., and Nobre, L. A statewide telemedicine network for public health in Brazil. In *Proceedings of the IEEE Symposium on Computer-Based Medical Systems*, Volume 2006, pages 495–500, 2006. DOI: 10.1109/CBMS.2006.29.

[16] Martin, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall Press, USA, 1st edition, 2017. ISBN: 0134494164.

[17] Pepito, G. RDBMS to NoSQL migration: Challenges and strategies, 2018. URL: https://www.researchgate.net/publication/341294540_RDBMS_to_NoSQL_Migration_Challenges_and_Strategies.

[18] Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., and Stumm, M. Continuous deployment at Facebook and OANDA. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 21–30. ACM, 2016. DOI: 10.1145/2889160.2889223.

[19] Shake Technologies, I. Metrics for measuring the productivity of your development team. URL: https://www.shakebugs.com/blog/measuring-developer-productivity/. [Accessed: 2022-09-29].

[20] Solanke, V., Kulkarni, G., Vishnu, M., and Kumbharkar, P. Private vs public cloud, 2013. URL: https://www.researchgate.net/publication/258253155_Private_Vs_Public_Cloud.

[21] SQL to NoSQL: Architecture differences and considerations for migration. Technical report, ScyllaDB, 2020. URL: https://www.scylladb.com/wp-content/uploads/wp-sql-to-nosql-architectur-differences-considerations-migration-1.pdf.

[22] Sultan, M. Angular and the trending frameworks of mobile and web-based platform technologies: A comparative analysis. In *Proceedings of the Future Technologies Conference*, pages 928–936, 2018. https://saiconference.com/Downloads/FTC2017/Proceedings/128_Paper_264-Angular_and_the_Trending_Frameworks_of_Mobile.pdf.