Acta Cybernetica **26** (2024) 543–562.

# The Influence of the Nonfunctional Requirements on the Data Model*

Grácián Kokrehel[ab] and Vilmos Bilicki[ac]

**Abstract**

During the design and development of real-world telemedicine applications, the data model evolves significantly along the datapath. The model itself, the storage technique, and the user interface are the most common contributors. This relates to non-functional requirements. The size and complexity of the domain model may also be significantly influenced by standards. This phenomenon is distinct from data model erosion, which occurs when the data model changes due to a software developer's fault and non-properly defined interfaces. This is occurring by design. We are unaware of any technique, including OMG's Unified Modeling Language (UML), that focuses on this aspect of complex systems: the change of the data model along the datapath. In this article, we investigate this phenomenon and, in addition to identifying the locations where this change may occur, we classify the modifications depending on the possible influence a specific model change may have on the system's overall properties. This paper presents a novel methodology for complex system datapath analysis and demonstrates its application to a selection of telemedicine-related applications. This technique illustrates the possible effect of non-functional requirements on the datapath and the potential consequences of these modifications.

**Keywords:** FHIR, telemedicine, GUI, Firebase, Angular, modeling

*This work 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. The research 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: kokrehel@inf.u-szeged.hu, ORCID: 0000-0002-5074-6033
[c]E-mail: bilickiv@inf.u-szeged.hu, ORCID: 0000-0002-7793-2661

DOI: 10.14232/actacyb.299253

# 1 Introduction

In the late 1950s, renting an IBM 704 digital mainframe computer cost hundreds of dollars per minute. Recently, cloud computing as a service with on-demand pay-per-use is a widely used Information Technology (IT) phenomenon that offers great economies of scale. In order to make the platform as a service more accessible and affordable, serverless computing has attracted the interest of both industry and academia.

Another important trend is the widespread use of Internet of Things (IoT) devices. The Function as a Service (FaaS) and Platform as a Service (PaaS) solutions provide the de facto backend for IoT solutions. The integration of the IoT to the cloud/edge node is governed in most cases by the traditional Representational state transfer (REST) paradigm, implemented on top of the Hyper Text Transfer Protocol (HTTP). The data is typically serialized in JavaScript Object Notation (JSON). On the datapath, data travels through a variety of technology stacks.

Domain model erosion is a phenomenon in which the information model of an application becomes separated from its actual implementation. When applications shift from one technology stack to another, causing the information model to change, or when software engineers contribute changes that are not accurately reflected in the information model, this might occur. When the data model of an application is expressed in JSON format, but the actual implementation of the data model changes without corresponding modifications to the JSON representation, domain model erosion can occur. This might result in incompatibilities between the data model and its representation, leading to unanticipated application behavior.

Consider, for instance, a web application that employs a JSON representation to store user information. If a software engineer adds a new field to the user data model, such as a new email address, but does not update the JSON representation to incorporate this information, the program may continue to work but will not keep the new email address for users.

In addition to the data erosion, which may be viewed as a design flaw (lack of strong, typed interfaces), the data model change along the datapath may be a real but little understood phenomenon. A widespread system integration approach is based on the REST architectural style, therefore there is no absolute domain model, but rather a given representation on a given portion of the data path. If we consider the MVVM (Model-ViewModel) architectural pattern, we can observe that the domain model in a given location/layer may differ dramatically from other locations. All of these alterations are possible in software systems; a comprehensive system integration is not required to meet these issues.

Non-functional requirements may also influence this domain model. When an application needs to grow to accommodate greater traffic, this is a classic example of non-functional requirements resulting in data model modifications. To accommodate the increasing load, it may be necessary to modify the data model for performance by adding additional indices, denormalizing the data, or sharding it across different servers. Another illustration is when privacy and security needs change, resulting in data model modifications. For instance, a new rule may man-

date that sensitive data be encrypted at rest, which may necessitate alterations to the manner in which it is kept within the data model.

In order to conform to non-functional requirements such as standard interfaces or system extensibility, the data format is governed by the standards of a given domain. E.g.: In the field of telemedicine the Fast Healthcare Interoperability Resources (FHIR) [4] standard is widely used. When implementing a system that is intended to conform to a standard, such as FHIR, the domain model may need to be extended or updated to comply with the standard. To describe the numerous healthcare concepts, the domain model may need to incorporate FHIR resources, such as patients, medications, conditions, and procedures, in the case of FHIR. Additionally, the domain model may need to be extended to include data elements.

## 2 Research questions

As stated in the introduction, both functional and non-functional requirements influence the evolution of data along the datapath. While functional requirements determine the necessary data transformations, non-functional requirements also play a significant role in shaping the data evolution process. This is a frequent practice among software developers, although it has not been properly investigated. This is a gray area from both a qualitative and quantitative sense. In addition to defining our study, we posed research questions. The first question discusses the potential ramifications of the modification. As observed, change occurs, but what are its consequences? The second research topic concerns the FHIR standard's effects on the domain model. FHIR is one of the few practical standards, and as a result, it is frequently used, making it an excellent candidate for study. The third study topic focuses on the technical environment's effects on the data model. In our situation, we chose Google's serverless Firebase solution, which is also a good contender due to its popularity. Within Firebase, the impact of using Firestore as a database is studied.

- RQ1: What are the dimensions which enable us to measure the impact of non-functional requirements?
- RQ2: Based on the defined dimensions, what are the impacts of the FHIR standard and using the Firebase API

## 3 State-of-the-art

There are numerous articles about software architecture and FaaS best practices. Wen et al. presented the first comprehensive study on understanding the challenges in developing serverless-based applications from the developers' perspective. They mine and analyze 22,731 relevant questions from Stack Overflow (a popular Q&A website for developers), and show the increasing popularity trend and the high difficulty level of serverless computing for developers. Through manual inspection of

619 sampled questions, they constructed a taxonomy of challenges that developers encounter, and report a series of findings and actionable implications [1].

In a different publication, Wen et al. presents a comprehensive study on characterizing mainstream commodity serverless computing platforms, including AWS Lambda, Google Cloud Functions, Azure Functions, and Alibaba Cloud Function Compute. Specifically, they conduct both qualitative analysis and quantitative analysis. Based on the results of both qualitative and quantitative analysis, they derive a series of findings and provide insightful implications for both developers and cloud vendors [2].

Grogan et al. showed the impact of the FaaS on the software architecture. The analysis of the data path is missing from these articles [3].

On another hand, there is a community focusing on cloud modeling e.g.: Bergmayr et al. investigated the diverse features currently provided by existing Cloud Modeling Languages (CMLs). They classified and compared them according to a common framework with the goal to support Cloud Service Customers (CSCs) in selecting the CML which fits the needs of their application scenario and setting. As a result, not only features of existing CMLs are pointed out for which extensive support is already provided but also in which existing CMLs are deficient, thereby suggesting a research agenda [4].

Software architectures allow identifying confidentiality issues early and in a cost-efficient way. Information Flow (IF) and Access Control (AC) are established confidentiality mechanisms, so modeling and analysis approaches should support them. Because confidentiality issues often trace back to data usage, data-oriented approaches are promising. However, Seifermann et al. could not identify a data-oriented approach to handling both, IF and AC. Therefore, they present a unified data-oriented modeling and analysis approach supporting both, IF and AC. They demonstrated the integration into an existing architectural description language and evaluated the resulting expressiveness and accuracy by a case study considering 22 cases [5].

The main theoretical results of Stunkel et al. are proofs of the facts that comprehensive systems are an admissible environment for (i) applying formal means of consistency verification (diagrammatic predicate framework), (ii) performing algebraic graph transformation (weak adhesive HLR category), and (iii) that they generalize the underlying set of graph diagrams and triple graph grammars [6]. The data path aspect and the evolution of the data are not studied.

From a data modeling perspective there are articles about the effect of the different storage formats that have been studied [7] or about different binary serialization formats [8]. There are also studies focusing on single system persistence issues [9] but the impact of the standards and the technical environment has not been studied.

Ulrich et al. presents a Metadata Repository (MDR) prototype that allows for the linking and mapping of data elements, enabling relations to be defined semi-automatically. The system enables the management of all registered data elements and metadata, allowing for comfortable queries within classified data elements. The architecture has technical advantages such as fast response times and the

ability to search across clinical coding systems. The MDR allows for the reuse of data elements, simplifying cooperation among research groups. The system was evaluated with positive results using two methods: usability testing and cross-validation [10].

# 4 Typical software in telemedicine stack

A typical software stack consists of an IoT or mobile client and a FaaS backend. In our case, Firebase is used as a FaaS service while the client side is implemented in an Angular environment. The data model is implemented in a FHIR conformant way.

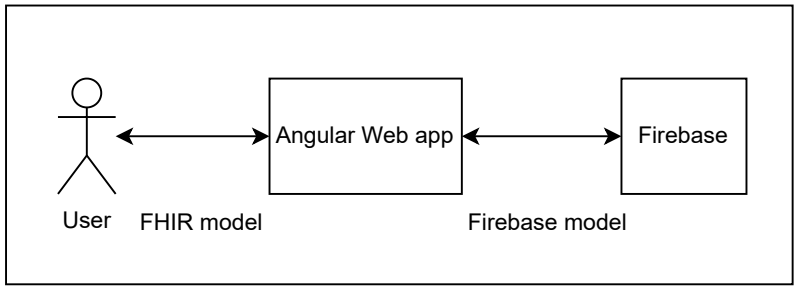


Figure 1: Cloud software stack

## 4.1 FHIR

The philosophy behind FHIR is to build a base set of resources that, either by themselves or when combined, satisfy the majority of common use cases. FHIR resources aim to define the information contents and structure for the core information set that is shared by most implementations. There is a built-in extension mechanism to cover the remaining content as needed [11].

## 4.2 Firebase

Cloud Functions is another integration of the existing Google product into Firebase. It is a tool for running back-end code from the cloud on an event-driven basis. The way Cloud Functions suggests running our app is what is usually called a serverless architecture. This type of architecture means building applications as a set of separate functions, isolated in the cloud, and connected between each other via APIs [12].

Usually, we will use the Realtime Database as our main storage. The main problem is limited querying capabilities. We cannot query for more than one key at a time and the service does not provide a way to filter our data. The format

also excludes the option to model the data. We do not host the data, all data is hosted on Firebase and it is a major problem of using BaaS platforms as our app backend. Unless Firebase provides a migration tool to enable easy transfer of user's data, it strongly limits data migration. It makes users dependent on the platform and there is no easy way to transfer the app to another source.

# 5 Methodology

The CAP theorem was utilized as a starting point for defining the dimensions of impact. We expanded it with security; defining the taxonomy where security should be associated is not straightforward. The "C" (Consistency) component is the first aspect of security to consider within the context of the CAP theorem. Access restrictions, encryption, and authentication are examples of security controls and protocols that can be used to maintain data consistency and ensure that only authorized users have access to critical information. Next, security considerations can be included into the "A" (Availability) component by designing redundant systems and disaster recovery methods that assure the ongoing availability of vital systems and data in the event of a security breach or other calamity. This may involve backing up important data to secure off-site locations and deploying network segmentation to mitigate the impact of any security compromise. Last but not least, security can be integrated into the "P" (Partition Tolerance) component by providing security controls that can detect and respond to network partitions and other disruptions that may jeopardize the system's security. Therefore, we decided to treat it as a distinct dimension. As a consequence, we settled on four major dimensions or categories for measuring the impact:

- Consistency
- Availability
- Partition tolerance
- Security

We examined the source code of the selected systems in order to determine the patterns in which a domain model change could affect a certain dimension. We chose ten sample pilot projects from our telemedicine portfolio as the focus of our research. From a functional standpoint, these projects adequately address the requirements of a certain medical field of interest. The target field might be deemed representative since, from both a modality (e.g., imaging, CT, vital signals, lifestyle, etc.) and a health sciences perspective, it encompasses a vast array of topics (e.g.: dermatology, Otology and rhino-laryngology, diabetes care, cohrea surgical planner, etc).

This was the qualitative portion of our analysis, in which we identified patterns within the source code that influence system properties along a specific dimension (CAP + Security).

To examine the influence of the technology stack and FHIR (RQ2), we conducted an analysis centered on the domain model. We took the source code and, beginning with the user interface, traced all affected source code sections until they reached the backend. This allowed us to extract the data pathways. Then, we extracted entity-specific subpaths from these datapaths (instances of a given part of the domain model). We integrated the data paths along the entities along the dimension, incorporating the discovered pattern-based concerns. Thus, a summary of the impact of technical infrastructure and FHIR on a given dimension at the entity level was obtained.

In the case of availability, we extended this analysis to include the average influence of the standard and the technological environment by collecting the data (from the databases of the running applications) at the entity level and deleting each subsequent layer until just the core information remained. The average size of each entity was then determined for each domain model level.

In the section under "Threats to Validity," the statistical importance of our data collection methods will be examined in detail. Here, we would like to emphasize that the analyses presented represent simply the first "sampling approach." There is a continuing effort to crawl and automatically analyze a subset of GitHub projects. The patterns detected by manually analyzing the code will likely be extended, but the current patterns will remain valid. The results on the data volume inflation may also be slightly impacted, but we are certain that our results will accurately reflect the magnitude of change.

# 6 Results

In this section, we define the dimensions and then show what the overall impact is based on real world examples. Following that, we will analyze the size problems and then we will present the structure of the data at a given analysis point.

## 6.1 Dimensions

RQ1: What are the dimensions which enable us to measure the impact of nonfunctional requirements?

As indicated in the methodology section, we decided to choose CAP+ Security as the primary dimensions. We gathered code patterns that have an effect on one or more of these dimensions. The patterns are classified into categories, which are then linked to dimensions. The categories were identified using a pattern-based grouping strategy. As indicated in the methodology section, we opted to choose CAP+ Security as the primary dimension. We gathered coding patterns that influence at least one of these dimensions. The patterns are divided into categories, and the categories are then linked to dimensions. As an appropriate grouping technique, the categories were determined based on the patterns. The following categories have been identified:

1. T(transformation): refers to situations where the domain model transforms online. Typically, this is the result of employing the MVVM design pattern or a comparable one. However, this may also occur on the backend, where some aggregation is required.

2. S(security): this indicates a data leak may occur (e.g.: for a given screen there could be an aggregated screen specific data containing sensitive information)

3. C(consistency): in this instance, consistency may be at risk (e.g.: aggregation of data and if the trigger is not executed then the data becomes inconsistent)

4. Pe(performance): data manipulation or access with increased overhead (e.g: deep data structures).

We can link these categories tha CAP in the following way:

- Consistency: C (Consistency), T (Transformation)
- Availability: Pe (Performance)
- Security: S (Security)

Due to Firebase, there is strong partition tolerance and availability, but in exchange, there are significant issues with consistency. During the analysis, we did not encounter any partition tolerance errors that would affect the data. We developed a tool for identifying the portion of the data path affected by the given effect. Table 1 displays the places and categories, with the names of the most significant patterns identified in each cell.

Table 1: Dimension with patterns

| T | sr/mr (single/multi row) | c(client) | s(server) | |
|---|---|---|---|---|
| T | 1 - string | 2 - number | 3 - date | 4 - others |
| S | 1 - gdpr | | | |
| C | 1 - id reference | 2 - embedded data | 3 - many extra data | 4 - data after delete |
| Pe | 1 - deep data | 2 - multiple promise | 3 - handle null/und | |

Then, we developed a coding method to enable the efficient and compact coding of all relevant information. The processed projects were developed using the Angular framework, therefore the code snippets below will be presented with JavaScript or TypeScript syntax. Table 1 helps in interpreting the following code snippets. These are the identified patterns:

***T SM/MR C/S 1*** String transformations happen including the modification of one or more records on the client or server side. In Firebase there is no way to order by based on the value of the object that is in the array, so it must be outsourced to a separate field. In Firebase there is no %like% search option, so another method can be used to achieve the same effect, which results in having to organize the text into a string array. The complex FHIR data structure can be simplified by merging it into one text.

```
function splitNameByVariations(nameString: string): Array<string> {
   const allSubstrings = new Set<string>();
   const start = 0;
   const splittedName = nameString.split(' ');

   for (let j = 0; j < splittedName.length; j++) {
       for (let i = start + 1; i <= splittedName[j].length; i++) {
           allSubstrings.add(splittedName[j].substring(start, i));
       }
   }

   let a = 0;
   while (a < splittedName.length) {
       let possibleSubStrs = '';
       for (let i = a; i < splittedName.length; i++) {
           possibleSubStrs += splittedName[i];
           if (i < splittedName.length - 1) {
               possibleSubStrs += ' ';
           }
       }
       for (let i = start + 1; i <= possibleSubStrs.length; i++) {
           allSubstrings.add(possibleSubStrs.substring(start, i));
       }
       a++;
   }

   for (let i = start + 1; i <= nameString.length; i++) {
       allSubstrings.add(nameString.substring(start, i));
   }

   return Array.from(allSubstrings.values());
}
```

***T SM/MR S 2*** Often happens that the smart device sends the data in the form of a string or with an incomplete value (undefined/null). Firebase cannot handle undefined data. Number transformations happen including the modification of one or more records on the server side.

```
getPulse(notFhirFormatData: any) {
    // for(data in [pulse, hearthrate, bp])
    let pulse = notFhirFormatData.pulse;
    if (pulse === undefined) {
        return null;
    }
    pulse = pulse.trim() as unknown as number;
    pulse = pulse.toFixed(2);
    return this.convertToFHIRFormat(pulse);
}
```

***T SM /MR C/S 3*** Date transformations happen including the modification of one or more records on the client or server side. Date transformation must be performed before insertion and modification, because Firebase used a unique date solution. timestamp = nanoseconds: 0, seconds: 0. If this transformation does not take place and we try to use the data, the client side will fail with an error. The localization required by the user also takes place here.

```
function formatPeriodLocal(start: Date, end?: Date) {
 const localStartDate = DateTime.fromISO(start.toISOString(),
   { zone: 'Europe/Budapest'});

 let formattedLocalDate = localStartDate.year + '. ' +
   localStartDate.month.toString().padStart(2, '0') + '. ' +
   localStartDate.day.toString().padStart(2, '0') + '. ' +
   localStartDate.hour.toString().padStart(2, '0') + ':' +
   localStartDate.minute.toString().padStart(2, '0');
 if (end) {
   const localEndDate = DateTime.fromISO(end.toISOString(),
     { zone: 'Europe/Budapest'});
   formattedLocalDate += ' - ' + localEndDate.hour.toString()
     .padStart(2, '0') + ':' +
     localEndDate.minute.toString().padStart(2, '0');
  }

 return formattedLocalDate;
}
```

***C 2/3/4*** Due to Firebase, numerous attributes and objects must be stored in the data; if the original data is modified, Cloud Function must be used to update/delete all embedded data, otherwise the data becomes inconsistent.

```
export const onEntity1UpdateOrDelete =
 functions.firestore.document('/Entity1/{cpid}').onUpdate/onDelete(
    async (snap: any) => {
        if (snap.after.exists === true) {
          const entity1: any = snap.after.data();
          const entity2: any = this.get(entity1.referenceToEntity2);
          // update/delete the embedded data in entity2        }
    });
```

***Pe13*** The deep data structure and their non-mandatory attributes place additional burdens on the client side. So that the application does not stop, we handle all non-mandatory data elements with the optional chain operator (?).

```
// reading the patient name
{{ patient?.name?.[0]?.given?.[0] }}  
{{ patient?.name?.[0]?.family }}
```

***Pe2*** Occasionally, a piece of data contains many references that must all be accessed on the client side. It can have many sources of error, and it must be treated as a transaction, if it stops during any step, it must be started again. This can be extremely burdensome for the client side.

```
async function getPatient(): Promise<IPatient>
{ /* Handle error, reset on fail, process data... */ }

async function getDevice(): Promise<IDevice> { /* ... */ }

async function getParent(): Promise<IPractitioner> { /* ... */ }

addApointment(){
    const [patient, device, parent] =
    await Promise.all([getPatient(), getDevice(), getParent()]);
    // Do something, create an appointment .....
}
```

As a summary, based on the patterns discovered in the source code files along the datapath, we divided the effects into four groups and, within each category, determined the exact code patterns responsible for the observed effect. Dimensions were also tied to the categories.

## 6.2 Overall impact

RQ2: Based on the defined dimensions, what are the impacts of the FHIR standard and using the Firebase API?

In order to assess the impact among the dimensions we collected the handled entities. We analyzed the datapath for each entity in different applications and created an abstract datapath based on the metrics. The table below contains the result of this process.

In Table 2, all the models that have been used till now were subjected to the evaluation points defined in Table 1. In the other columns, three main problem sources were defined: FHIR, Firebase(CRUD) and GUI. Then the representation of common errors at given points. Based on the above results, it can be noticed that the FHIR model alone is not enough, but several transformations have to be applied during development to get the proper results. If we observe carefully we can see that the incorrect combinations are repeated column by column, so for e.g. under Firebase Create we get the same errors, which means that the problems can be well delimited and thus they can be solved as a group, there is no need to deal with it specifically by model. The development of solutions to individual problems can be defined and reused for other models as well. Explanation for Patient row:

1. String transformation must be performed on the server before insertion and modification, because in Firebase there is no %like% search option, so another method can be used to achieve the same effect, which results in having to organize the text into a string array.

2. When reading data, consistency problems arise because of the embedded data. Because of Firebase, many attributes and objects must be stored in the data, if the original data is changed, all embed data must also be updated using Cloud Function.

3. It is a security problem if the patient's sensitive data (name, social institute number, birthdate) are also displayed when the view tables are created.

4. It is an extra task to ensure that no reference to the entity remains anywhere after deletion.

For example, in the Appointment row:

1. Date transformation must be performed before insertion and modification, because Firebase used a unique date solution. timestamp = nanoseconds: 0, seconds: 0

2. Because of more than three references and embed data, extra data has to be retrieved and displayed, which burdens the performance of the GUI.

Table 2: Analysis of FHIR models

| Model | FHIR | Firebase | | | | GUI |
|---|---|---|---|---|---|---|
| | | C | R | U | D | |
| Device | C2 C3 Pe1 | T4s | C2 S1 | T4s | C4 | |
| Patient | C3 Pe1 | T1srs | C2 S1 | T1srs T1mrs | C4 | C4 Pe1 |
| Practitioner | C3 Pe1 | T1srs | C2 S1 | T1srs T1mrs | C4 | C4 Pe1 |
| Appointment | C1 Pe1 | T3s | C2 | T3s | | Pe2 |
| Questionnaire | C1 Pe1 | | | | | Pe1 |
| Questionnaire Response | C2 Pe1 | | | | | Pe1 |
| Group | Pe1 | T1s | C2 | T1s | | C4 Pe2 |
| CarePlan | Pe1 | | C2 | | | Pe1 |
| Condition | Pe1 | | C2 | | | Pe2 |
| Communication | Pe1 | | C1 | | | Pe1 |
| Goal | Pe1 | | C2 | | | Pe1 |
| Medication Request | C2 Pe1 | | C2 | | | Pe1 |
| ServiceRequest | C2 C3 Pe1 | | C2 | | | |

## 6.3 Data size evolution

RQ2: Based on the defined dimensions, what are the impacts of the FHIR standard and using the Firebase API?

The last dimension of the analysis is the data overhead caused by the elements on the datapath. Out of the 25 deployed telemedicine solutions, we extracted data of the selected entities. The starting point was the fully extended data stored in the Firebase, with the help of scripts we were able to remove the firebase specific field in order to get the simple FHIR conform data, with the help of another script we were able to extract the core data in order to remove the FHIR overhead, now we have what is called the basic data. Figure 2 shows the size difference of 16.67 times between the simple and the final state.

Table 3 shows the data overhead for both the clear text and compressed data. We can conclude that FHIR adds a very significant overhead, while the Firebase specific data fields also doubles the FHIR data volume. So it is important to remove this overhead before sending it to the IoT client, the DAO layer has its place in this case.

In Table 3, we compared the 3 forms of data (Simple, after FHIR conversion and after transformations required by Firebase) in JSON file size (bytes) and zipped (bytes) file size. We can observe is that the size of the json files without repetition does not change as much as expected after compression. Furthermore, it can be observed that the ratio between the columns is the same or very similar for each row.

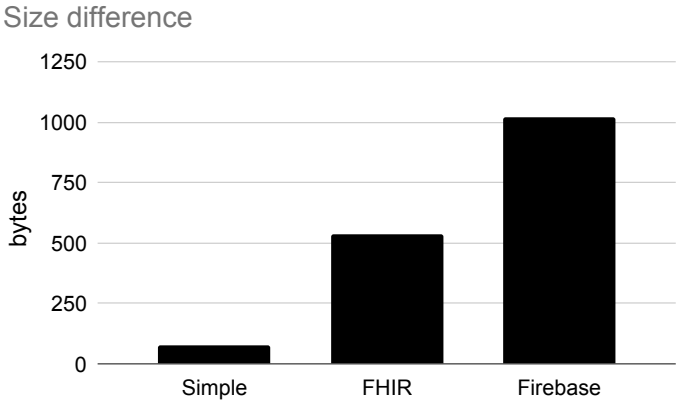


Figure 2: Size difference

Table 3: Size difference(bytes)

| Model | Basic | Basic zip | FHIR | FHIR zip | Firebase | Firebase zip |
|---|---|---|---|---|---|---|
| Device | 94 | 36 | 980 | 499 | 1560 | 602 |
| Patient | 220 | 56 | 1920 | 700 | 3666 | 937 |
| Practitioner | 130 | 42 | 1430 | 628 | 2080 | 708 |
| Appointment | 84 | 35 | 992 | 518 | 1410 | 574 |
| Questionnaire | 72 | 31 | 985 | 503 | 1160 | 523 |
| Questionnaire Response | 145 | 46 | 1653 | 526 | 2400 | 805 |
| Group | 47 | 28 | 700 | 412 | 799 | 440 |
| CarePlan | 49 | 31 | 809 | 426 | 865 | 459 |
| Condition | 39 | 23 | 589 | 375 | 659 | 398 |
| Communication | 37 | 22 | 581 | 369 | 643 | 388 |
| Goal | 44 | 26 | 646 | 388 | 705 | 411 |
| Medication Request | 43 | 25 | 623 | 384 | 716 | 430 |
| Service Request | 74 | 37 | 1180 | 586 | 1320 | 623 |

### 6.4 Details of the data expansion

In the previous section, we presented the effect of certain parts of the system, now we will explain step by step how the models look. The comparison is based on JSON file sizes. In this section, the different stations are introduced and explained, based on a simple example. Listings 1, 2, and 3 can be found in the Appendix section. The four analysis condition are as follows:

1. Simple (file size: 75 bytes) [Listing 1]
2. Effects of FHIR (file size: 536 bytes) [Listing 2]
3. Effects of Firebase (file size: 1022 bytes) [Listing 3]

### 6.5 Effects of Firebase

Due to the use of Firebase, the model went through extra changes in order to be able to implement certain filtering interfaces. Description of data fields:

- `data` — The data must be stored according to the FHIR standard and returned to the user in this form.
- `email, name` — In Firebase there is no way to order by based on the value of the object that is in the array, so it must be outsourced to a separate field.
- `nameText, emailText` — In Firebase there is no `%like%` search option, so another method can be used to achieve the same effect, which results in having to organize the text into a string array.

## 7 Discussion

As indicated in the introduction, our motivation was to observe/study the life/evolution of the domain model along the datapath as influenced by non-functional requirements such as technology, standards, and even the use of predefined design patterns. As we were unable to find appropriate measures in the literature, we established both the dimension along which we wanted to assess the impact and, within that dimension, the exact metrics that assist us understand the nature of the change and its impact. Instead of using a theoretical approach, we began to examine the source code for patterns that could be utilized as metrics. We were able to discover categories and groupings of patterns. We uncovered thirteen patterns, which we presented in the findings section. We supplied a fundamental datapath map indicating where and why these patterns could be discovered (backend vs. frontend, which data subpath e.g. CRUD). With this method, it is possible to discover potential points that are not "bad smells" from a software quality viewpoint, but rather key portions of the datapath that require special attention. We believe that this datapath-oriented perspective could provide important insight into a system's inherent features. Table 2 provides a summary of this map's typical entity

level pathways. The true potential of an approach will become apparent when defined patterns could be detected automatically. Our ongoing effort is now centered on this issue. We have also demonstrated how implementing a standard or utilizing a technology stack affects the data size of the domain model. We agree that we cannot alter the standard itself, but we would like to provide a suggestion for future standard's data structure architecture. If one consults the FHIR documentation, it is evident that modularity/extensibility, not simplicity, was the driving force behind the domain model. One could argue that with 4G, 5G, and XG technologies, the amount of data to be transmitted across the line is insignificant due to the large bandwidth. Here, we would like to emphasize the delay caused by the transfer of substantially more data. As end users are not patient, delay is a crucial part of the design of actual user-facing technologies.

# 8 Threats to Validity

The objective of our research was to determine the domain model's response to non-functional needs. We adopted an analytical strategy by identifying and assessing a code basis. We selected 10 telemedicine initiatives (consisting of more than 200 screens, and 300 modules) from our portfolio. Functionally, these projects satisfy the needs of the medical industry. The target field is representative from both modality (e.g., imaging, CT, vital signals, lifestyle) and health sciences vantage points (e.g., imaging, CT, vital signals, lifestyle) (e.g.: dermatology, Otology and rhino-laryngology, diabetes care, cohrea surgical planner, etc). We agree that further examples from the open-source community should be included. This is likely to increase the number of sample patterns and create a more complex depiction of the problem, but our conclusion and fundamental patterns will remain unchanged. Consequently, it may be anticipated as the initial result in this field. Concerning the size-related findings, it is difficult to construct a valid database for a particular open-source application; thus, we believe that our results are statistically significant because our system is utilized in routine medical work.

During our research, we made the following decisions:

- The FHIR was used as an example for analyzing the impact. As one of the most prevalent criteria, we believe this to be a suitable option.

- We chose the Angular - Firebase technology stack to investigate the impact of technology on the domain model. In this case, we consider that the selection of technological stacks does not reduce the statistical significance of the study. In the case of web application frameworks (e.g., React, Vue, etc.), general design patterns (e.g., MV*) may have similar effects. From a backend perspective, Firestore has the same constraints as other serverless document stores, therefore it was also a strong contender. We agree that it would be interesting in the future to categorize the various persistence options and evaluate the impact of each category separately.

# 9 Conclusions

It is evident from the literature review that there are numerous techniques to investigate complex software stacks, but there are very few articles that account for the entire data lifecycle. By evaluating significant issues encountered in the creation of Telemedicine applications utilizing the FHIR standard, we identified key evaluation criteria for modern systems. With our methodology, we can determine how architectural and component-level design patterns are applied. This strategy will demonstrate its effectiveness if it is accompanied by an automated data life cycle analysis tool. In our ongoing effort, we have already located more than 9k GitHub projects, and we are currently creating an NLP-based method for identifying the patterns that match to the requirements. Massive standard objects present issues when a small IoT device transmits data; in this instance, it is important to consider a simpler model than FHIR. FHIR adds a large amount of overhead, and the addition of Firebase-specific data fields increases the FHIR data capacity. Before providing data to the IoT client, it is crucial to eliminate this overhead; the DAO layer has a place in this scenario. Even if the system must be standardized, the DAO layer makes it possible for devices to provide a minimal quantity of data while the ultimate outcome is still standardized. In the subsequent essay, we will describe this concept's capabilities.

# References

[1] Altexsoft. What is Firebase: Review, pros and cons, alternatives. URL: https://www.altexsoft.com/blog/firebase-review-pros-cons-alternatives/. Accessed: 2022-09-26.

[2] Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., and Leymann, F. A systematic review of cloud modeling languages. *ACM Computing Surveys*, 51(1):1–38, 2018. DOI: 10.1145/3150227.

[3] Grogan, J., e. a. A multivocal literature review of Function-as-a-Service (FaaS) infrastructures and implications for software developers. In *European Conference on Software Process Improvement*, pages 58–75. Springer, 2020. URL: https://link.springer.com/chapter/10.1007/978-3-030-56441-4_5.

[4] HL7. Overview — FHIR v4.0.1. URL: https://www.hl7.org/fhir/overview.html. Accessed: 2022-03-26.

[5] Petković, D. SQL/JSON standard: Properties and deficiencies. *Datenbank Spektrum*, 17(3):277–287, 2017. DOI: https://doi.org/10.1007/s13222-017-0267-4.

[6] Seifermann, S., Heinrich, R., Werle, D., and Reussner, R. A unified model to detect information flow and access control violations in software architectures.

In *Proceedings of the 18th International Conference on Security and Cryptography*, Volume 1, pages 26–37. SCITEPRESS — Science and Technology Publications, 2021. DOI: 10.5220/0010515300260037.

[7] Stünkel, P., König, H., Lamo, Y., and Rutle, A. Comprehensive systems: A formal foundation for multi-model consistency management. *Formal Aspects of Computing*, 33:1067–1114, 2021. DOI: 10.1007/s00165-021-00555-2.

[8] Swami, D. and Sahoo, B. Storage size estimation for schemaless big data applications: A JSON-based overview. In *Intelligent Communication and Computational Technologies*, Volume 19 of *Lecture Notes in Networks and Systems*, pages 315–323. Springer, Singapore, 2018. DOI: 10.1007/978-981-10-5523-2_29.

[9] Ulrich, H., Kock, A.-K., Duhm-Harbeck, P., Habermann, J. K., and Ingenerf, J. Metadata repository for improved data sharing and reuse based on HL7 FHIR. *Studies in Health Technology and Informatics*, 281:160–164, 2021. DOI: 10.3233/978-1-61499-678-1-162.

[10] Viotti, J. C. and Kinderkhedia, M. A survey of JSON-compatible binary serialization specifications, 2022. DOI: 10.48550/arXiv.2201.02089.

[11] Wen, J., Chen, Z., Liu, Y., Lou, Y., Ma, Y., Huang, G., Jin, X., and Liu, X. An empirical study on challenges of application development in serverless computing. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 416–428. Wiley, 2021. DOI: 10.1145/3468264.3468558.

[12] Wen, J., Liu, Y., Chen, Z., Chen, J., and Ma, Y. Characterizing commodity serverless computing platforms. *Journal of Software: Evolution and Process*, 35(10), 2021. DOI: 10.1002/smr.2394.

# Appendix

Listing 1: `Simple.json`

```
{
    "id": "1",
    "email": "jon@doe.mr",
    "name": "Mr. Jon Doe"
}
```

Listing 2: `Fhir.json`

```
{
    "id": "1",
    "telecom": [
        {
            "system": "email",
            "value": "jon@doe.mr",
            "use": "home",
            "rank": 1,
            "period": ""
        }
    ],
    "name": [
        {
            "use": "official",
            "text": "Mr. Jon Doe",
            "family": "Doe",
            "given": [
                "Jon"
            ],
            "prefix": [
                "Mr."
            ],
            "suffix": [],
           "period": ""
        }
    ]
}
```

Listing 3: `Firebase.json`

```
{
    "id": "1",
    "data": {
        "id": "1",
```

```
        "telecom": [{
                "system": "email",
                "value": "jon@doe.mr",
                "use": "home",
                "rank": 1,
                "period": ""
            }],
        "name": [{
                "use": "official",
                "text": "Mr. Jon Doe",
                "family": "Doe",
                "given": ["Jon"],
                "prefix": ["Mr."],
                "suffix": [],
                "period": ""
            }]
},
    "email": "jon@doe.mr",
    "name": "Mr. Jon Doe",
    "nameText": [
        "m", "mr", "mr.", "j", "jo", "jon", "d", "do", "
           doe", "mr. j",
        "mr. jo", "mr. jon", "mr. jon ", "mr. jon d", "mr
           . jon do", "mr. jon doe"
    ],
    "emailText": [ "j", "jo", "jon", "jon@", "jon@d",
        "jon@do", "jon@doe", "jon@doe.", "jon@doe.m", "
           jon@doe.mr"] }
```