Received February 17, 2022, accepted March 30, 2022, date of publication April 7, 2022, date of current version April 26, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3165548

SFTSDH: Applying Spring Security Framework With TSD-Based OAuth2 to Protect Microservice Architecture APIs

AYAN CHATTERJEE[1], MARTIN W. GERDES[1], PANKAJ KHATIWADA[2], AND ANDREAS PRINZ[1]

[1]Center for e-Health, Department of Information and Communication Technology, University of Agder, 4630 Kristiansand, Norway
[2]Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU), 7034 Trondheim, Norway

Corresponding author: Ayan Chatterjee (ayan.chatterjee@uia.no)

This work was supported in part by the University of Agder.

ABSTRACT The Internet of Medical Things (IoMT) combines medical devices and applications that use network technologies to connect healthcare information systems (HIS). IoMT is reforming the medical industry by adopting information and communication technologies (ICTs). Identity verification, secure collection, and exchange of medical data are essential in health applications. In this study, we implemented a hybrid security solution to secure the collection and management of personal health data using Spring Framework (SF), Services for Sensitive Data (TSD) as a service platform, and Hyper-Text-Transfer-Protocol (HTTP (H)) security methods. The adopted solution (SFTSDH = SF + TSD + H) instigated the following security features: identity brokering, OAuth2, multifactor authentication, and access control to protect the Microservices Architecture Application Programming Interfaces (APIs), following the General Data Protection Regulation (GDPR). Moreover, we extended the adopted security solution to develop a digital infrastructure to facilitate the research and innovation work in the electronic health (eHealth) section, focusing on solution validation with theoretical evaluation and experimental testing. We used a web engineering security methodology to achieve and explain the adopted security solution. As a case study, we designed and implemented electronic coaching (eCoaching) prototype system and deployed the same in the developed infrastructure to securely record and share personal health data. Furthermore, we compared the test results with related studies qualitatively for the efficient evaluation of the implemented security solution. The SFTSDH implementation and configuration in the prototype system have effectively secured the eCoach APIs from an attack in all the considered scenarios. The eCoach prototype with the SFTSDH solution effectively sustained a load of ($\approx$) 1000 concurrent users in the developed digital health infrastructure. In addition, we performed a qualitative comparison among the following security solutions: SF security, third-party security, and SFTSDH, where SFTSDH showed a promising outcome.

INDEX TERMS API Security, TSD, spring framework, HTTP, OAuth2, eCoach.

I. INTRODUCTION

IoMT has been an emerging ecosystem of wireless and connected medical-grade devices. It provides exciting opportunities in the healthcare sector to collect, transfer, and store personal health data over a network without necessitating human-to-human or human-to-computer interaction [1], [2]. IoMT is a combination of Internet-connected medical devices (including wearable devices and standalone devices for remote patient monitoring) and patient information [3]–[6]. The connectivity between medical devices simplifies clinical workflow management and improves patient care in medical institutions and remote areas [5]–[8]. In the IoMT ecosystem, IoMT devices collect or perceive data and information about health and personal well-being and then send them to databases or other IoMT devices or clouds or nodes (for example, fog or edge computing nodes) over the Internet [7]–[11].

The lack of security awareness may promote the following attacks on the IoMT ecosystem: illegal penetration attempts (e.g., cross-site forgery (CSRF), cross-site scripting (XSS), cross-source resource sharing (CORS) and Clickjacking),

The associate editor coordinating the review of this manuscript and approving it for publication was Mansoor Ahmed.

asset destruction, record theft, denial of service (DoS), distribution DoS (DDoS), man-in-the-middle attack (MITM), content sniffing (CS), brute force attack (BF), Internet protocol (IP) spoofing and therapeutic manipulation (TM) [5]–[7]. The Open Web Application Security Project (OWASP) discovered that IoMT security vulnerabilities include the following [5]–[7]:

- Inadequate authentication and authorization,
- Exposed Application Programming Interfaces or APIs (e.g., mobile, web, cloud),
- Inefficient data transmission encryption,
- Vulnerable network services,
- Privacy issues, and
- Deficient security configurability

Therefore, guaranteeing the safety of IoMT is an acute problem to be resolved. Due to the rapid growth of IoMT solutions and the continuous development of IoMT technology, security assurance has caused issues for IoMT users and organizations adopting IoMT technology. These problems exist, especially when choosing appropriate and reliable safety measures for the situation. Thus, authorized, and authenticated users or devices should only access the IoMT healthcare system. Insufficient authentication may cause an attacker to enter the system and access the user's private healthcare data. More and more malicious attackers target medical servers and digital medical systems because personal medical data and information are precious in the illegal market [3], [5]–[7]. Therefore, medical service providers require even more robust security measures, which inevitably increase the cost of creating, operating, and maintaining these medical services. Due to security breaches, retrieving and eliminating stolen data is both challenging and critical. The government and medical institutions must formulate strict regulations and severe penalties to protect patients' health records.

The reliability and credibility of eHealth scientific research and associated services rely on the health data protection plans and guidelines regarding security, privacy, and confidentiality. The Health Insurance Portability and Accountability Act of 1996 (HIPAA or the Kennedy–Kassebaum Act) of the United States (USA) [12] and the EU GDPR [13] are two worldwide accepted standards (or Privacy Rule) for the protection of individually identifiable health information. The principles of the privacy law cover the use and disclosure of health information from individuals - called "protected health information". The primary purpose of the privacy policy is to ensure that individuals' health information is adequately protected while facilitating the flow of health information required to provide and promote high-quality health care and protect the health and well-being of the public. Personal data is the information that can identify a person, directly or indirectly [12]–[14]. It primarily includes online identifiers such as IP addresses, cookies, digital fingerprinting, and location information that may identify people [12]–[14]. The GDPR [13] has the following six general data security standards:

- Fairness and lawfulness
- Limitation of purpose
- Minimization of data
- Precision
- Limitation of storage
- Honesty and confidentiality

Norwegian data protection policies (NORMEN) [14], another Norwegian health and care provider standard for information protection and privacy, is an agreed collection of information security standards based on legislation. In the modern digital data era, it is no longer possible to run research on sensitive data on local facilities, with limited capacities and no possibility to share results with collaborators. Research environments at universities and university hospitals have expressed a need for ample storage space for sensitive research data to handle.

This research aims to develop a secure digital health infrastructure with SFTSDH security solution for research and innovation services, providing secure hosting and operation of application services, collection, storage, processing, and provisioning of data, and test the security solution with the deployment of an eCoach prototype system as a case study. We adopted Representational State Transfer (REST)-based microservices architecture (MSA) with Spring Framework to develop lightweight eCoach APIs with independently deployable services to increase granularity and agility. Limited studies focused on applying security solutions at the MSA level, which has been addressed in this research. The different gaps between this study and the existing studies have been captured in Section III. This research addresses the following research questions –

1. How to protect REST APIs from external vulnerabilities with an integrated security solution approach (SFTSDH)?

2. How to extend the SFTSDH with VPN, Bcrypt hash, API key, network firewall, and SSL protocol to build a digital health infrastructure?

3. How scalable and effective is the adopted security solution approach?

SFTSDH implements security features, such as identity brokering, OAuth2, multi-factor authentication, CORS, and user management to protect the REST APIs from illegitimate access and external attacks, such as CSRF, XSS, Clickjacking, content sniffing, and brute force. An eCoach prototype system has been deployed in the developed digital health infrastructure to conduct a formal security analysis of the integrated security solution scheme as a case study. In addition, performance analysis and qualitative analysis have been performed to show how scalable and effective is the adopted security solution approach.

The rest of this paper is designed as follows. Section II describes the essential preliminaries required for the adopted security solution. Section III describes related work and emphasizes the difference between existing and our work, focusing on the architectural aspects besides different security features. In Section IV, we present our developed digital health system architecture.

Section V describes the adopted security solution. In Section VI, we describe experimental results. The paper is concluded in Section VII.

II. ESSENTIAL PRELIMINARIES

The needed technical background to realize the SFTSDH security solution has been described in this Section. In healthcare systems (e.g., eCoach [18]–[21]), personal health data are collected from various sources (e.g., wearables, fitness trackers, medical devices, assessment tools, self-reports, and user applications). According to the GDPR and Norwegian law, personal research data and information are sensitive and must be stored and processed under strict regulations [13], [14]. There are different security methods described in Section III to protect REST APIs. However, in combination with SF [15] and TSD platform [16], the HTTP security paradigm [17] may offer a parallel approach to prevail security functionalities, their features, session management, secure communication, and access management solutions for REST APIs. The reasons behind adopting SF, TSD platform, and HTTP security paradigm in our security solution are:

(a.) Spring Framework [15], [22] provides a comprehensive programming and configuration model for modern Java-based enterprise applications on any deployment platform. A key element of Spring is application-level infrastructure support. Spring focuses on the ''pipeline'' of enterprise applications so that developers can focus on application-level business logic without unnecessary contact with a specific deployment environment.

(b.) TSD (service for sensitive data) [16] is a GDPR supported platform for collecting, examining, and offering sensitive information consistent with the NORMEN. The TSD is an IT platform for research, regardless of whether it is sometimes utilized for clinical exploration and business research. TSD has been created and worked by the University of Oslo (UiO) and is a piece of NorStore, the public foundation for dealing with logical information. All the services and data are protected inside of TSD from illegitimate access. The TSD offers services such as client registration (signup, confirmation, getting an API key, and password reset), authentication and authorization with access tokens, file import and export in JavaScript Object Notation (JSON) (simple file upload and download, resumable file upload and download, resuming uploads and download, four different types of access tokens for both basic authorization and TSD authorization services (survey_import, survey_export, survey_admin, and survey_member), queries for filtering, encryption (Base64 encoded) of JSON and file data. In this research, TSD has been considered a third-party Identity and Access Management (IAM) platform.

(c.) HTTP [17], [23] is a ubiquitous protocol and one of the foundations of the network. HTTP is a stateless protocol based on messages (requests, responses), headers (key-value pairs), and optional bodies. The HTTP protocol works on the Transmission Control Protocol (TCP) [23]. TCP is one of the core protocols in the Internet protocol stack. It provides reliable, orderly, and error-checking data stream transmission, making it an ideal choice for HTTP [23]. Some essential features of HTTP in a web security context are [17], [23]: query string, URL encoding, cookies, built-in authentication mechanism (e.g., Basic and Digest), and Headers. Basic and Digest are two built-in methods of HTTP. However, other authentication methods are Windows NT LAN Manager (NTLM), IWA (Integrated Windows Authentication or Kerberos), Transport Layer Security (TLS), or Secure Socket Layer (SSL) client certificates. In addition, forms authentication, OAuth2, Security Assertion Markup Language (SAML), JWT (JSON Web Token), and many other types of authentication options reuse features in HTTP (such as form data or headers) to authenticate the client [23]. HTTP headers are used to pass additional information in requests and responses. Usually, custom headers start with X- (for example, X-Content-Type-Options header, X-Frame-Options header, etc.), but this is just a widely adopted convention, not by the HTTP protocol [23].

The MSA [24] helped to avoid the following drawbacks associated with traditional monolithic approaches to software development [25], [26]: bundled deployment as a single stack, limited scalability, DevOps challenges, and high resource cost. Subsequently, we deployed the eCoach prototype in the intended secure digital infrastructure and performed functional testing for acceptability. REST endpoint security includes the following methods: HTTP-based authentication scheme with basic or bearer token, API keys, OAuth2 with the access token and refresh token, and OpenID Connect (e.g., OpenID, BankID). OpenID Connect (OIDC) [27] is a thin layer on top of OAuth2 that adds insights concerning the client signing into the client profile. TSD supports the following four types of authentication schemes: basic, two-factor, ID-porten, and dataporten. We used TSD's two-factor authentication scheme and access token-based authorization method for REST API security. eHealth scientific research and related services' reliability and believability depend on the health data security plans and rules regarding security, and confidentiality. For this research, we got an ethical endorsement from The Norwegian Center for Research Data (NSD) and Regional Ethical Committee (REK) for overseeing information for our eHealth research in Norway.

III. RELATED WORK

A. eHEALTH SECURITY AND PRIVACY REQUIREMENTS

Articles associated with eHealth research reveal the subsequent security and privacy requirements in healthcare systems for both on-premises and cloud-based [28], [29]: EHR security, user authentication, governing amenability, authorized access, secrecy, ethical consent, legal issues, the relevance of data access, data ownership, data uniformity, data separation, security audits, archiving, requirements for third-party certificates (such as SAS70 Type II, PCI DSS Level 1, ISO 27001, and FISMA), protection against external security threats (such as DoS, DDoS, MITM, IP spoofing), security policies,

TABLE 1. **Security solutions with existing MSSA with respect to the key attributes of a secure web application architecture.**

Research	Inter-Tier Authentication	Server-Side Validation	Secure Communication	Data Encryption	Logging
Chatterjee et al. (our work)	Yes	Yes	Yes	Yes	Yes
Salibindla et al. [40]	No	No	No	No	No
Xie et al. [41]	Yes	No	No	No	No
Nguyen et al. [25]	Yes	Yes	No	No	No
Dikanski et al. [26]	Yes	No	No	No	No
Aloufi et al. [42]	No	No	No	No	No
Beer et al. [43]	No	Yes	No	Yes	No
Serme et al. [44]	No	No	No	Yes	No
Backere et al. [45]	No	No	Yes	No	No

TABLE 2. **Security solutions with existing MSSA with respect to the implemented security features.**

Research	Multi-factor Authentication	OAuth 2.0	SSL/TLS	Bcrypt hash	API Key	Third party Service Providers	Protection against CSRF, XSS, Clickjacking, content sniffing, and BF	CORS
Chatterjee et al. (our work)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Salibindla et al. [40]	No	No	No	No	No	No	No	No
Xie et al. [41]	No	Yes	No	No	No	No	No	No
Nguyen et al. [25]	No	Yes	No	No	No	No	CSRF, BF, XSS	No
Dikanski et al. [26]	No	Yes	No	No	No	No	No	No
Aloufi et al. [42]	No	No	No	No	No	No	No	No
Beer et al. [43]	No	No	No	No	No	No	DoS, DDoS, BF	No
Serme et al. [44]	No	No	No	No	No	No	No	No
Backere et al. [45]	No	No	TLS	No	No	No	No	No

security protocols, and database access management. The identified vital security terms can be disseminated in the following four categories: authentication (multi-factor, form-based, access token, and API key), authorization (OAuth2, OpenID, Dataporten, and CORS), encryption (digital certificate, SSL, RSA, Bcrypt, SHA-256, Base64, and MD5) and external security threats (CSRF, DDoS, MITM, XSS, brute force, and IP spoofing). Authentication is used to verify personal identity; identity verification is related to verifying credentials. The plan determines whether the person is a legitimate user. Authorization is a mechanism used to determine whether a particular service is available to authenticated users. It verifies that users have the right to personally access resources such as records, databases, and files. Usually, authorization is performed after authentication to verify the user privileges. Encryption uses an algorithm to encrypt, encrypt data and then use a key to decrypt, which is the information of the receiving group. External threats are the possibility that people outside the system may use malicious software, hacking, disruption, or social engineering to exploit system vulnerabilities. Healthcare research shows various studies [24], [30]–[39] associated with security and protection in electronic health records (EHRs), secure health monitoring framework, security conventions, and endorsement plans, security contemplations in medical services applications, strategies for medical services security, and interoperability, healthcare cloud, big data security, medical services security consistence, security execution, security difficulties, and success factors. However, studies related to security solutions at the Microservice Solution Architecture Level (MSSA) are limited.

B. MSSA

Salibindla [40] surveyed MSSA and Microservice API security methods (e.g., OAuth, HTTPS), focusing on the security of communication protocols. A security architecture [54]–[59] consists of security principles, methods, and models designed to align with business goals and help protect organizations from cyber threats. The key attributes of a secure web application architecture are [54]–[59]: Inter-tier authentication (e.g., LDAP, Kerberos, Mutual SSL, IP Validation, ID Portal), Server-side validation (e.g., secure Proxy, protection against external attacks), Secure communication (e.g., HTTPS), data encryption and logging. The microservice open security architecture framework provides a comprehensive overview of the vital security issues, principles, components, and concepts that are the basis for the architectural decision-making involved in designing an effective security architecture. Xie *et al.* [41] published a result on Spring API Security architecture and implementation without verifying the effectiveness of the Spring Security Framework (SSF) and OAuth2 when these technologies were used to enforce Microservice API endpoint authentication and authorization. Nguyen *et al.* [25] conducted a proof of concept (PoC) of a Microservice application using SSF and OAuth2 to reduce the knowledge gap on Microservice and API security. However, they did not test how the solution would work after integrating with a third-party IAM platform. Dikanski *et al.* [26] completed a conceptual study to identify and execute authentication and authorization patterns in the SSF to reduce the gap between the design and implementation of pattern-based protection to include high-quality security characteristics in software systems. Nevertheless, their study suffered from the SSF's actual performance to protect Microservice at the API endpoint level with integration with IAM platform. Alofi *et al.* [42] proposed a secure and cost-effective model based on the Message Queuing Telemetry Transmission (MQTT) protocol to protect IoT resources through access control to RESTful Web services. Beer *et al.* [43] proposed an adaptive security architecture based on neural networks to protect RESTful Web services in enterprise computing environments through three functional principles-intelligent methods for predicting, preventing, and learning to detect future threats. The core of their security solution is to protect HTTP transactions based on Public Key Infrastructure (PKI) and related encryption technologies. The interpretation results show that compared with the supported network/transport layer security, the proposed security solution is suitable for protecting REST and is better than SOAP-based Web services. Since RESTful Web services are stateless, they usually do not have any session to perform the challenge-response mechanism. TLS/SSL provides secure peer authentication.

Nevertheless, when the authentication request is based on delegation, this mechanism is not sufficient to allow the site to authenticate on behalf of its users. HTTP security (HTTPS) is widely used confidentiality, but it only provides hop-by-hop protection. An excellent RESTful solution follows a token-based approach. Serme *et al.* [44] proposed a security model based on confidentiality and digital signatures to protect RESTful messages. These messages carry undeniable tokens and provide data secretly by encrypting their contents. They proposed a protocol to ensure the communication security of RESTful services. They provide encryption, signatures, and their combination. They do not intend to offer an equivalent secure session for RESTful services because it is related to the transport layer security of HTTP, which has been resolved in protocols such as SSL and TLS. Backere *et al.* [45] Designed a security solution for RESTful Web services using non-RESTful elements to outperform TLS.

The summary of the security solutions concerning essential attributes of a secure web application architecture and security features are described in Table 1 and Table 2 and are further compared with our adopted SFTSDH security approach. Table 1 and Table 2 show how our SFTSDH solution is different from the existing studies.

C. STATE-OF-THE-ART

This study aims to reduce the MSA and API security knowledge gap and protect REST APIs by creating an MSA application prototype using SFTSDH (see Table 2). In conjunction with Spring Framework, TSD platform, and HTTP methods, we implemented a parallel API protection solution (SFTSDH) with pre-existing security features such as identity brokering, OAuth2, multi-factor authentication, and CORS. We performed unit testing to validate the security solution at the modular level. Moreover, we created a self-signed SSL certificate with Keytool to protect sensitive information on the web using public key (RSA) encryption [46]. On top of the SSL protocol, we used "eduVPN" to provide a safer E-2-E connection with service encryption and changed IP addresses. It did not only help data traffic to pass all the mid-points safely but also provided access control by allowing legitimate VPN users to access the REST endpoints of the eCoach prototype system. Finally, we validated the scalability of the adopted approach as a performance metric. In our solution approach, all the implemented security features are described in Table 3.

IV. SYSTEM MODELING

Personal health and wellness data are usually collected over time through wearable sensors, offline and online interactions, smartphone apps, self-reporting questionnaires, and feedback forms [18]–[21]. To collect contextual weather data over time, both free and paid weather APIs and various weather sensors are available in the AppStore or marketplace. For ubiquitous tracking and distinct levels of physical activity measures, high-end, time-dependent activity data collection with wearable BLE-enabled devices has become known and achievable. Wearable activity sensors are connected via Bluetooth communication technology (BLE) to a smartphone. The eCoach smartphone application can seamlessly track and transfer high-resolution raw acceleration data to safe storage for further processing with a scheduler module. Some activity data are questionnaire-dependent (self-reporting), such as non-wear time and intense activity information. Physiological data can be collected (e.g., weight, blood pressure, heart rate, SPO2, body assessment data) either invasively (e.g., glucose level, lipid profile) or non-invasively. Based on the self-reported questionnaire, behavior data (dietary and habit) are collected either regularly or irregularly (alternating days or weekly). Baseline data (demographic data, medical history, personal preferences, initial weight and height, initial

TABLE 3. Well-established security features implemented in the SFTSDH security solution.

Security Features	Description
Multi-factor authentication	Multi-factor authentication is a form of electronic authentication in which access to a website or application is given to a computer user only after two or more pieces of evidence are successfully submitted to an authentication mechanism: information, possession, and inherence.
Bearer Token	Bearer Tokens are the prevalent type of OAuth 2.0 access token. A Bearer Token is an invisible string that is not supposed to have any significance for the external user using it. Some servers will issue tokens that are hexadecimal characters in a short string, whereas others use standardized tokens like JSON Web Tokens.
API Key	A unique identifier used to authenticate a user, creator, or calling program to an API is an application programming interface key (API key). However, they are usually used to authenticate a project instead of a human user with the API. API keys can be introduced and used by various platforms in various ways.
OAuth2.0	OAuth is an open-standard authorization protocol or mechanism that explains how, without sharing the original, linked, single logon credential, unrelated servers, and services can safely allow authenticated access to their properties. The Access Token and Refresh Token are the two token forms used in OAuth2 authentication. The access token is used to get access to resources from the resource server for authentication and authorization. The refresh token is usually sent in combination with the access token.
CORS	Cross-origin resource sharing (CORS) is a process that enables another domain outside the domain from which the first resource was served to request limited resources on a web page.
Self-signed certificate	A self-signed certificate is a security certificate not signed by the certificate authority (CA) for cryptography and computer security purposes. It is easy to make these certificates, and they do not cost money. However, they do not have all the security features that certificates signed by a CA are meant to provide. They are certificates for SSL, code signing, and S/MIME.
HTTPS	Hypertext Transfer Protocol Secure is a Hypertext Transfer Protocol extension. It is used over a computer network for encrypted communication and is commonly used on the Internet. In HTTPS, Transport Layer Security (TLS) or formerly Protected Sockets Layer (SSL) is used to encrypt the communication protocol.
RSA (Rivest, Shamir, and Adleman)	RSA is a public-key cryptosystem commonly used for the transmission of secure data. It is an asymmetric cryptography algorithm. It is one of the oldest as well. It has key sizes of 1,024 to 4,096 bit typical. A pair of keys must be created by each person or party who wishes to engage in communication using encryption, namely public key, and private key. The public key is used for encryption, and the private key is used for decryption.
Bcrypt (Blowfish-based crypt)	To store a password that is a one-way password hash, Bcrypt uses an adaptive hash algorithm. When encoding passwords and storing the salt and the encrypted password, Bcrypt internally produces a random salt. It is also apparent for the same string to get different encoded results. It is not possible to decrypt the password. A Bcrypt hash string is of the form: $2b$[cost]$[22-character salt] [31-character hash].

blood pressure, initial lipid profile, initial glycemic response, and initial body assessment data) are being collected for demographic statistics or population clustering, or individual target assessment during the participant's initial recruitment or monthly basis.

In this context, we have developed a digital health infrastructure after extending the SFTSDH features with VPN, Bcrypt hash, API key, firewall, and SSL as depicted in Figure 1. Moreover, we deployed a health eCoach prototype system in the developed infrastructure to execute a formal security testing of the adopted SFTSDH solution and it is depicted in Figure 2 and Figure 3 for a smartphone application (app.) version and a web version, respectively. We maintained a modular structure for our eCoach prototype system with the following modules [21]: activity (for device provision, collection and visualization of activity intensity, activity pattern, activity classification, posture detection, and step count over time), scheduler (for periodic notification generation and activity data transfer), eCoachUI (user interface for forms, dashboard, and basic information), eCoachBusiness (for the management of user login, complaint, performance, and data), weather (for the collection of weather data from OpenWeather using API-Key authentication and visualization of weather trends), and fhir (for data interoperability with HL7 protocol and medical ontology [47]). FHIR stands for Fast Healthcare Interoperability Resources for providing semantic and structural interoperability in personal and person generated health data with FHIR resources and clinical vocabularies.

A project p-1075 was created on 11th March 2020 under the name "diferi" on behalf of UiA on TSD side. The TSD infrastructure (see Figure 1) was established to support researchers, digital service operators, healthcare service providers, and third-party system and solution vendors to carry out research and innovation work jointly with the Centre for e-Health at UiA, the Centre for eHealth Research I4Helse, and other research groups, partners, and customers. The TSD infrastructure collects, stores, and exchanges different

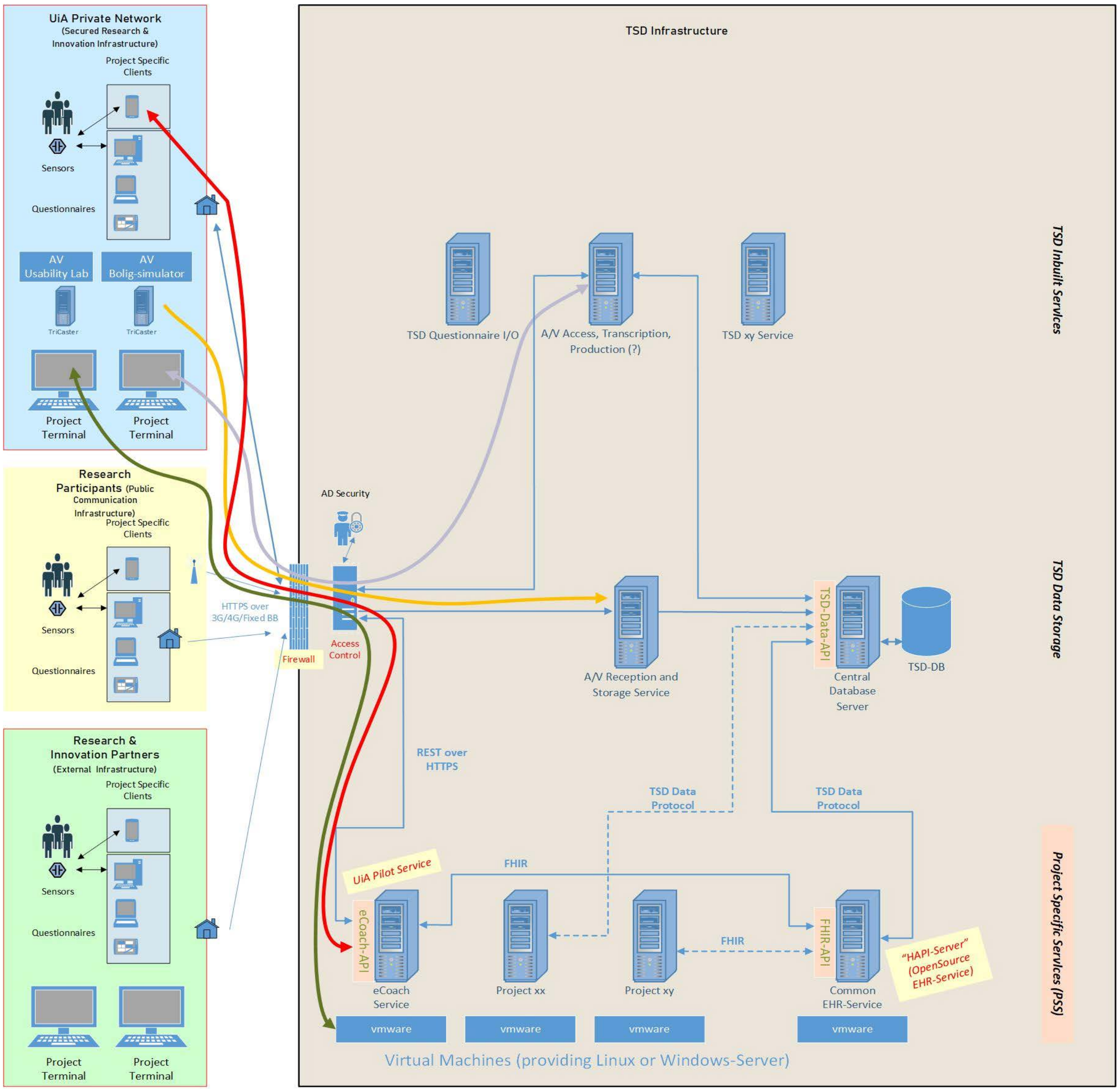


FIGURE 1. The developed digital health infrastructure with extended SFTSDH features.

health-related data. It supports incoming data generated from the following dissimilar sources:

- UiA Private Network or Secured Research and Innovation Infrastructure that collects data from sensors, questionnaire, usability lab and the living lab (Boligsimulator) and project terminals.
- Research participants or Public Communication Infrastructure that collects data from wearable sensors, non-wearable sensors, and questionnaire.
- Research and Innovation Partners or External Infrastructure that collects data from sensors, questionnaire, and project terminals.

TSD infrastructure supports project specific servlets (PSS), data storage, and inbuilt servlets. TSD infrastructure can host multiple services in respective VMware. All the collected data are sent to specific services (e.g., eCoach) deployed in the TSD virtual machines (VMware) using HTTPS, following the TSD authentication and authorization rules, and further stored in the central TSD database. TSD infrastructure additionally supports standard-based EHR or patient-journal-system (PJS), secure access to the data for project-specific services (e.g., for processing and evaluation of the data for decision-support services), and the secure provisioning of the data to facilitate smooth exchange and interoperability

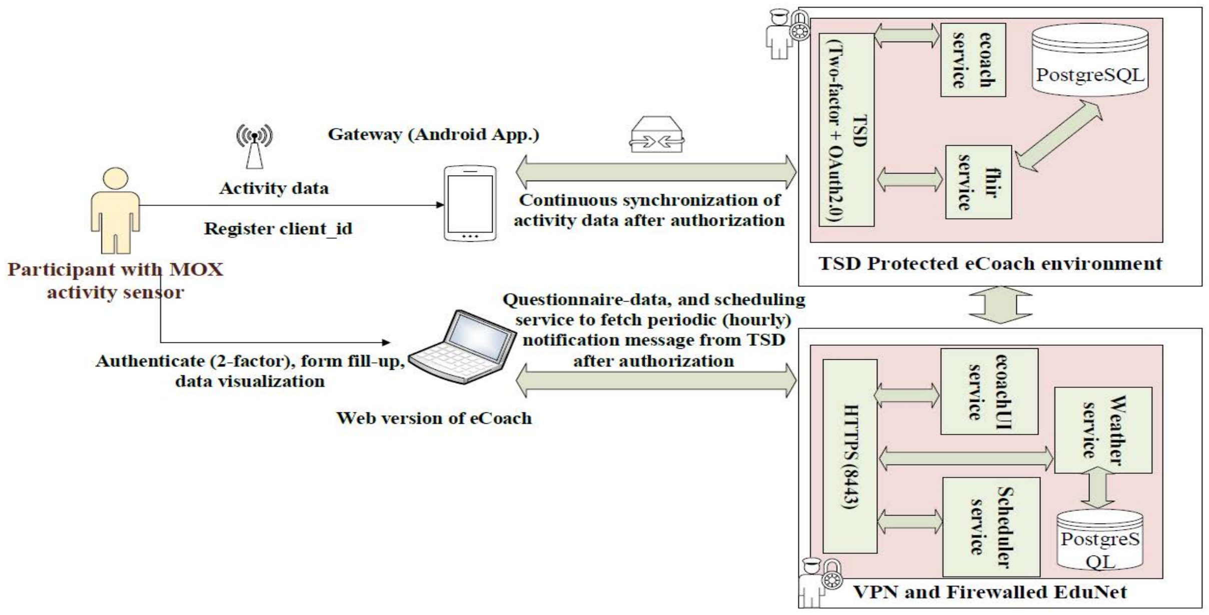

Gateway (Android App.)
Activity data
Register client_id
Participant with MOX activity sensor
Continuous synchronization of activity data after authorization
TSD (Two-factor + OAuth2.0)
ecoach service
fhir service
PostgreSQL
TSD Protected eCoach environment
Questionnaire-data, and scheduling service to fetch periodic (hourly) notification message from TSD after authorization
Authenticate (2-factor), form fill-up, data visualization
Web version of eCoach
HTTPS (8443)
ecoachUI service
Weather service
Scheduler service
PostgreSQL
VPN and Firewalled EduNet

FIGURE 2. The proposed security solution for our health eCoach prototype system for web version.

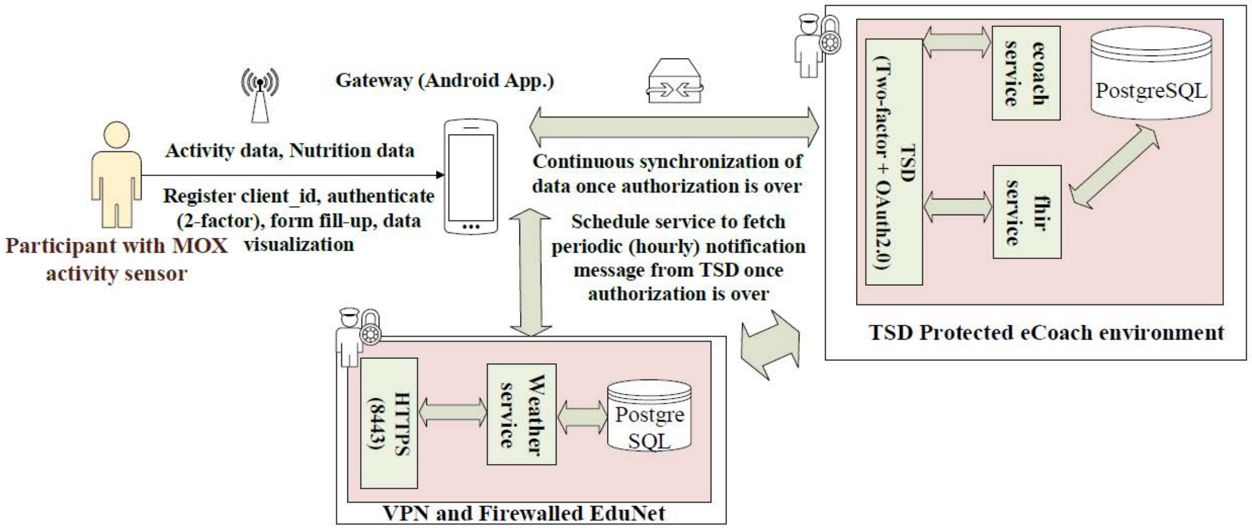

Gateway (Android App.)
Activity data, Nutrition data
Register client_id, authenticate (2-factor), form fill-up, data visualization
Participant with MOX activity sensor
Continuous synchronization of data once authorization is over
Schedule service to fetch periodic (hourly) notification message from TSD once authorization is over
TSD (Two-factor + OAuth2.0)
ecoach service
fhir service
PostgreSQL
TSD Protected eCoach environment
HTTPS (8443)
Weather service
PostgreSQL
VPN and Firewalled EduNet

FIGURE 3. The proposed security solution for our health eCoach prototype system for smartphone app. version.

between infrastructure components and project partners. The project aims at the following outcomes:

- Components in the digital health infrastructure for secure collection and storing of health data.
- Establishment of a pilot service for project-specific data collection and processing.
- The environment to store Audio & Video (A/V) recordings securely.
- Providing seamless, secure, and full access for UiA users and external partners to the digital infrastructure.

Following a general decision by UiA/IT, the TSD infrastructure within this pilot project will be established within the TSD platform at the University of Oslo. In line with this, an EHR or PJS for standard-compliant reception and provisioning of health data has been set up within TSD. A secure communication mechanism and access control have been established between the TSD proxy gateway and the EHR/PJ server. As a pilot study to test the established infrastructure with a project-specific prototype service for research, eCoach services have been deployed and integrated into the secure digital infrastructure. That pilot services receive project-specific person-generated data (PGDs) from external user devices and applications and store the data in the TSD database following semantic and structural interoperability. The pilot services boost project-specific data processing, decision support, real-time communication with user devices

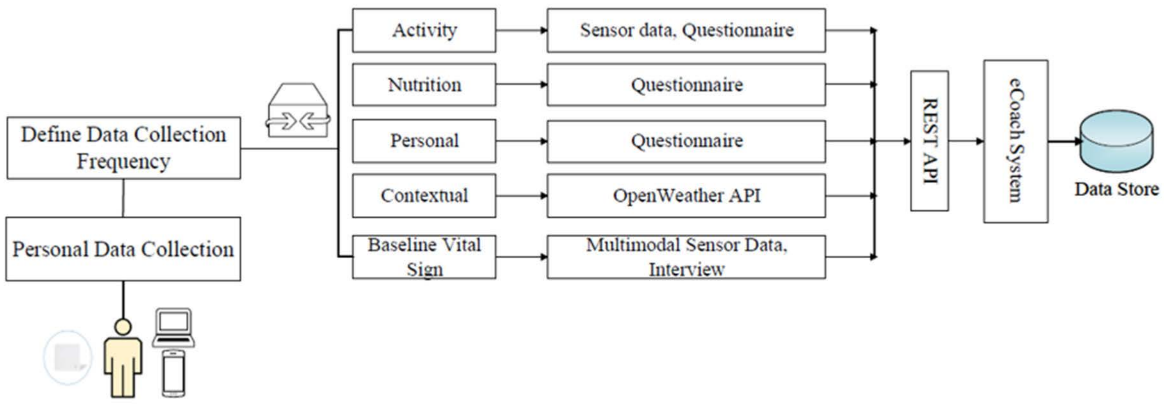


FIGURE 4. **Data collection modules of the health eCoach prototype system.**

and applications in a public communication infrastructure (i.e., outside the secure digital infrastructure).

The Usability Lab and the Living Lab (Boligsimulator) of the I4Helse environment at UiA support the audio-visual observation, monitoring and recording of test scenarios for research and innovation purposes. Audio-Visual recordings from human test participants must be treated as personal, privacy-relevant data, and must be handled securely. For this, the secure digital infrastructure within the TSD system will be connected to the audio/video lab facilities at UiA/I4Helse, to allow secure transmission, storage, analysis, and access to the recorded content. Shared data access requires strong security and access control. At the same time, established access control solutions at UiA and external partner institutions, such as Feide, BankID will be supported. These access control mechanisms will be integrated with the access control solution of the TSD system to extend the robustness of the solution. In this study, we have focused on project aims (a) and (b). The remaining project aims (c) and (d) are of the future research focus.

TSD does not allow outgoing API calls inside the projects and is open for the registered networks only. TSD's internal IP addresses are not published to external DNS services. Therefore, we set up a project-specific proxy at UiA. Each service deployed inside the TSD infrastructure is accessible from outside TSD with exposed REST-APIs using OAuth2. TSD can provide an infrastructure to host multiple disjoint projects and connect services to a secure, centralized database server. TSD PostgreSQL database is protected by its access control list (ACL) rules and authentication mechanism. Further details related to the service deployment inside TSD as a docker image and its access outside TSD using the necessary security tokens are discussed in Section V.

A. DATA COLLECTION

The activity module is responsible for activity device registration, device allocation, seamless collection of sensor observations, and sending it back to the eCoach business module for storing data in a PostgreSQL database. We used a wearable MOX-2 [48] activity monitor to collect personal activity data for the following measurement parameters – physical activity classification (low intensity, medium intensity, and high intensity), posture detection (sedentary, standing, and weight-bearing), physical activity intensity (counts per minutes), and steps. It is an activity monitor based on a BLE accelerometer embedded. Consuming low power, the device can seamlessly measure and transmit high-resolution raw acceleration data and multiple activity parameters per second for seven consecutive days (up to 60 days). Activity data collection is a two-step process: a. activity data collection from MOX-2 activity sensor to personal mobile in CSV format using BLE protocol, and b. periodic uploading of CSV data from mobile location to activity module for persistent storage and further analysis using a scheduler service using HTTP-POST (see Figure 4). The weather module periodically contains weather updates from OpenWeather REST APIs (latest and hourly) with API-Key authentication and sends them back to eCoach business logic for stable storage. The questionnaire module consists of six question sets – daily, alternative day, weekly, interview, baseline (monthly), and feedback form. The participant submits the questionnaire, which is stored back in the database through eCoach business logic.

B. eCOACH SYSTEM (APP. VERSION VS WEB VERSION)

The eCoach endpoints are exposed as REST APIs. All the modules are connected to a PostgreSQL relational database system for persistent data storage using object-relational mapping (ORM) via the HAPI-FHIR module for semantic representation of PGDs using medical terminologies in JSON. The activity, scheduler, and eCoachUI modules are written using a Spring-Framework, and their exposed REST APIs are tested with Spring-Boot Swagger. To monitor these modules' health, the Spring-Boot Actuator provides secure endpoints, such as /metrics, /env, /beans, /trace, /health, and /info, which are protected by a role-based authorization scheme. The eCoach system can be accessed by both versions: a smartphone mobile app. (android) and a web. The "/ecoachui/home" and "/ecoachui/" APIs are exposed to the external user (but protected with VPN access, a firewall, and SSL). Other APIs are protected with access-role to the TSD platform. The user interface module is responsible for app view, web view, and data visualization based on individual access roles. We focus only on establishing a secure digital health infrastructure, eCoach security implementation, and security verification in this study. Concepts such as sensor description, processing of time-dependent (activity, nutrition, habit) and time-invariant (demographic) data, data visualization, and the implementation of HAPI-FHIR (HL7 V. 4) are beyond the scope of this paper.

Figure 2 and Figure 3 depict how the security solution for the eCoach system has been developed with the TSD platform for the smartphone app. version and web version, respectively. In smartphones, all the personal and person-generated health data (e.g., activity sensor data, questionnaire, and interview) are collected with an android eCoach app. and transferred to the eCoach services hosted in the TSD infrastructure for storing in the TSD central database. Using web version, only questionnaire and interview data can be captured and recorded in the TSD database. From TSD, access to the external server is prohibited. Therefore, the weather module is deployed outside of TSD infrastructure; however, inside of a VPN (EduVPN) and firewall-protected ubuntu infrastructure (EduNet), provided by UiA to access external resources as Weather APIs. Weather data collected from Weather APIs are stored in a PostgreSQL relational database inside EduNet. Networks inside EduNet are accessible (e.g., SMTP-mail.outlook.com); however, they must go through a proxy for external access. We deployed the following three variants of the Scheduler module: (the first one or v.1) inside of the TSD (for scheduled notification generation and storing the result in the TSD database as a notification message), (the second one or v.2) inside of the EduNet (for periodic notification message collection from TSD, combining it with the weather forecasting for contextual recommendation generation, and store the personalized notification message in TSD), and (the third one or v.3) inside of the eCoach mobile application (for periodic activity data transfer to TSD and periodic notification message collection from EduNet to generate notification pop-up alerts). Personal and person-generated health data inside the EduNet is protected and free from identity disclosure. Moreover, our eCoachUI module has two deployed versions: (v.1) inside of the EduNet (user interface for eCoach Web version) and (v.2) inside of the mobile application as a user interface.

Only activity, scheduler (v.1), eCoachBusiness, and the fhir sub-modules are deployed inside TSD. TSD exposes their endpoints under the following three services: /v1/p1075/ecoach/activity, /v1/p1075/ecoach/scheduler, and /v1/p1075/ecoach/fhir. These services are protected with OAuth2. The FHIR service is responsible for giving access to HAPI-FHIR REST APIs for semantic and structural interoperability. On the first installation of the eCoach mobile app, participants must register a system-generated user-identifier that will be stored in the smartphone's in-memory storage (or SQLite database) with the Bcrypt encryption algorithm. The user-identifier will help in session management and the transfer of personalized activity data. With the de-registration of the eCoach app., it will be auto removed from the mobile. SQLite database will be further reused to store valid short-lived (30 days) access tokens for TSD authorization. Inside EduNet, we created a self-signed SSL certificate with Keytool (keytool -genkey -alias apache-tomcat -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore eCoach.p12 -validity 3650) to secure confidential web information using public key (RSA) encryption. The Keystore file path was then inserted into the configuration file of the Apache-tomcat webserver to alter the application start-up port from 8080 to 8443.

The eCoach prototype system consists of five user types: researcher, developer, system admin, health professional, and participants. These users are grouped into ADMIN (researcher, developer, system admin) and USER (health professional or nurse, and participants) for role-based access management. Researchers and developers are responsible for eCoach design, development, test, and validation study. An ADMIN is responsible for infrastructure support. They have no access to participant's PGDs and dashboard. Trained health professionals (e.g., professional nurses) are responsible for offline interviewing to facilitate participant screening and collecting initial and baseline data. Furthermore, they can view the dashboard to monitor the participant's health status and health progress. Participants have access to their self-reporting questionnaire, feedback forms, and health monitoring dashboard through the eCoachUI module.

C. METHODS FOR SECURITY IMPLEMENTATION AND PERFORMANCE EVALUATION

There is no single protection method to meet all the security and design requirements for our modular and distributed eCoach prototype system. To realize and justify the adopted security solution, we applied a web engineering security methodology proposed by Aljawarneh *et al.* [49]. The software engineering principles encourage the method built up on top of the standard waterfall system development life cycle (SDLC). The applied approach reduced substantial threat exposures during all the SDLC phases by integrating security and assessment factors at each SDLC phase which software engineers and security professionals verified.

To determine the performance of the adopted security approach, we evaluated the API scalability. Throughput and latency were considered to measure the API scalability [53], [60]–[62]. Network throughput refers to the average data rate at which data or messages are successfully delivered on a specific communication link. It is measured in bits per second (bps). The maximum network throughput equals the TCP window size divided by the communication packet's round-trip time (RTT). The method does not consider communication overhead, such as network receiver window size, machine limitations, or network latency [53], [60], [61]. Network latency is the time taken for a packet to be captured, transmitted, processed through multiple devices, and then received and decoded at the destination [53], [60], [61]. We generated HTTP request loads to check API scalability as a "Thread Group" with Apache open-source software JMeter (V 5.4.1) and captured corresponding throughput and latency. For the load testing with JMeter [53], the following three properties have been considered critical: the number of threads or users, the ramp-up period in seconds, and the loop count to set the test count. We repeated the experiment multiple times with a loop count value of five for individual load and took a mean throughput and latency. Low latency and high throughput are good performance indicators to support real-time critical applications.

V. SECURITY IMPLEMENTATION SCHEME

This section describes our security solution implementation and then its validation. We demonstrate security configuration, sequence diagrams for user registration, authentication, authorization, application deployment, and the login process to access web resources from the TSD as IaaS (Infrastructure as a Service). We tested the security performance of the system in a real-time environment.

A. HYBRID SECURITY SCHEME AND ITS DEPLOYMENT IN THE ARCHITECTURE

For Linux VMs, TSD uses *Thinlinc* remote access protocol based on HTML5, with a *Nginx* proxy for two-factor authentication. *Thinlinc* is a remote desktop framework that supports HTML5 to connect to their Linux machines using their browser instead of a *VNC* client. Thinlinc uses *TigerVNC* and provides an additional layer of user and agent VM administration, allowing the automatic assignment of project user-groups to Thinlinc agents installed on the Linux VMs. *Thinlinc* infrastructure consists of a *Thinlinc* proxy and a *Thinlinc* master. The proxy runs *Nginx* and a customized log-in protocol. The *Thinlinc* master server is the machine where the users are redirected to after authenticating through the proxy. The master acts as the broker, keeping track of all the *Thinlinc* agent machines, and redirecting users to the correct agent machine. Users need to do an extra login on the actual project VMs. Each service deployed inside of TSD platform (e.g., */v1/p1075/ecoach, /v1/p1075/scheduler* and */v1/p1075/fhir*), is associated with a unique long-lived API Key ({"api_key" = <key>}).

The keys are valid for one year and expire automatically. It is the responsibility of the system administrator to order a new key with an application-specific (e.g., eCoach) client_id and password (superuser). The key is just a JWT token, therefore, it can be decoded to verify if it has expired or not. The API key is used as a bearer token in the HTTP request header to generate short-lived access token (valid for 30 days). The admin can request five long-lived API keys in total. If anyone abuses the API, then long-lived API token is revoked. The complete API Key generation, sign-up and login processes are as depicted in Figure 5 and Figure 6.

TSD has pre-defined access tokens to investigate which access tokens are theoretically available to give clients access

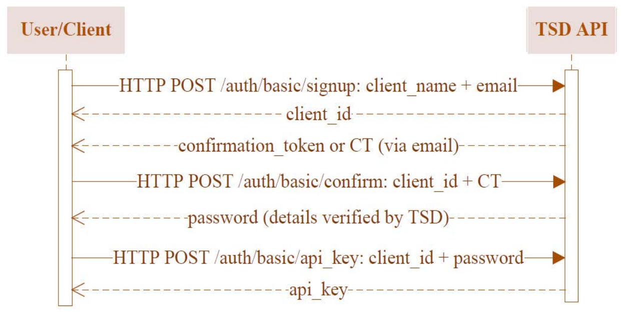

User/Client
TSD API
HTTP POST /auth/basic/signup: client_name + email
client_id
confirmation_token or CT (via email)
HTTP POST /auth/basic/confirm: client_id + CT
password (details verified by TSD)
HTTP POST /auth/basic/api_key: client_id + password
api_key

FIGURE 5. Admin sign-up and API key generation for TSD exposed services.

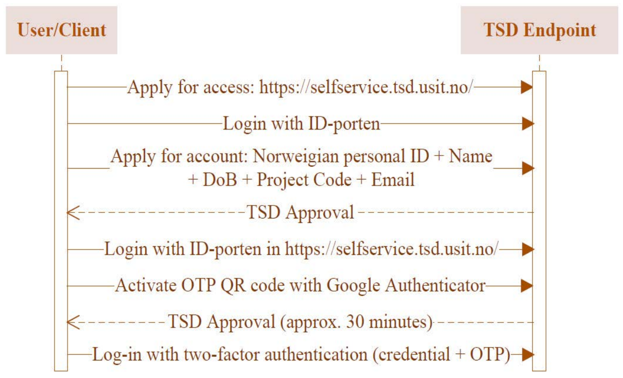
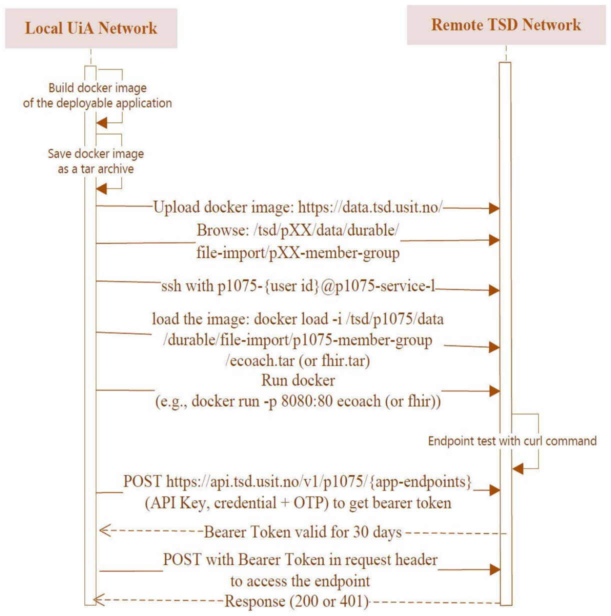

User/Client
TSD Endpoint
Apply for access: https://selfservice.tsd.usit.no/
Login with ID-porten
Apply for account: Norweigian personal ID + Name + DoB + Project Code + Email
TSD Approval
Login with ID-porten in https://selfservice.tsd.usit.no/
Activate OTP QR code with Google Authenticator
TSD Approval (approx. 30 minutes)
Log-in with two-factor authentication (credential + OTP)

FIGURE 6. Sign-up and login for project members.

to the specific API (*GET /v1/p1075/auth/tokens/info*). TSD API supports the following two authentication methods: basic and two-factor authentication. In the former, using just the API key, users can get the short-lived access token to their application to import data to TSD for the project to they have consent. Applications using basic authentication cannot export any data. In the latter, a google authenticator-based OTP is required to import data to TSD and export data from TSD. Both the authentication headers are explained in Textbox I.

Textbox I The Authentication Headers in the Request Header

```
POST
https://api.tsd.usit.no/v1/p1075/auth/basic/token
Authorization: Bearer $api_key
{
   "User_name":"p1075-test",
   "password": "*********"
}
OR
POST
https://api.tsd.usit.no/v1/p1075/auth/tsd/token?type=<type>
Authorization: Bearer $api_key
{
   "User_name":"p1075-test",
   "otp":"******",
   "password": "*********"
}
```



Local UiA Network
Remote TSD Network
Build docker image of the deployable application
Save docker image as a tar archive
Upload docker image: https://data.tsd.usit.no/
Browse: /tsd/pXX/data/durable/ file-import/pXX-member-group
ssh with p1075-{user id}@p1075-service-l
load the image: docker load -i /tsd/p1075/data /durable/file-import/p1075-member-group /ecoach.tar (or fhir.tar)
Run docker (e.g., docker run -p 8080:80 ecoach (or fhir))
Endpoint test with curl command
POST https://api.tsd.usit.no/v1/p1075/{app-endpoints} (API Key, credential + OTP) to get bearer token
Bearer Token valid for 30 days
POST with Bearer Token in request header to access the endpoint
Response (200 or 401)

FIGURE 7. Application deployment in TSD and accessing exposed service APIs.

TSD's network is strictly firewalled, and internal IP addresses are not published to external DNS services. TSD has a hierarchy of three DNS servers, two UNBOUND (the resolves for the addresses inside TSD) and one BIND being the source of authority and responds to queries from TSD and UiO resolvers outside TSD (having only the IPv6 addresses). TSD servers will not be able to ping or do DNS lookups on addresses outside the primary firewall by TSD. There is no Internet access to and from the project VMs, but we can communicate between hosts within the project's network, either from the login node or the service host itself. We can perform HTTP requests to the services once they are up and running through a given proxy. All the applications hosted on the TSD project VM are secured automatically and accessed within the TSD network and cannot be accessed from the external network. To access the APIs of our hosted services within TSD, we need to follow the steps depicted in Figure 7.

Our eCoach modules were first deployed into a test server (see Textbox II), and then it was hosted on the docker container in the VM in TSD (see Textbox III). The direct connection between the TSD environment and the outside internet and vice-versa is not possible due to the security policy of TSD. Thus, we are unable to use the internet functionalities from inside the VM and cannot access the docker hub to pull the docker images of the developed application. Hence, we need to follow an utterly offline procedure to build a docker image, upload it on TSD using their proprietary file upload procedure, load the image in the Docker container, and run it as depicted in Figure 7. TSD provides PostgreSQL and MSSQL as the DB services. TSD has mirrored the UiO setup for DBs inside TSD, and every project in need of a database (DB) will get its VM with a DB on it. The DB uses block device storage from the HUS-VM, and there are four DB administration computers. All DB traffic is encrypted and uses SSL certificates from Uninett. For this project, p1075-dbpg01.tsd.usit.no PostgreSQL DB host is created and opened from the p1075-service-l.tsd.usit.no service host. According to the security policies from the TSD on the PostgreSQL server, we do not have superuser privileges. TSD admin team can create application-specific databases and DB-owners. This restriction is done to guarantee a proper backup of PGDs. These DBs are only accessible from inside the VM from the service host and the applications hosted in the TSD but cannot be accessed from the public internet.

Textbox II Prerequisites to Deploy eCoach Modules Into the Test Server

- ✓ Setup the working environment with JDK 8.0+ version and Apache-Maven build tool (V 3.X)
- ✓ Download a module (e.g., activity) from GitHub with clone or if its existing in the local codebase then perform pull operation.
- ✓ Install the latest version of Apache-tomcat webserver (e.g., V9.0.3).
- ✓ Create a database namely "ecoach" in PostgreSQL (e.g., V13.0)
- ✓ Create a table "TBL_ACTIVITIES" under eCoach database
- ✓ Configure activity module to PostgreSQL database in the environment property file.
- ✓ Compiling and installing activity module → generate activty.war file.
- ✓ Deploy the web archive file (war) into the tomcat webserver.

Textbox III Deploying eCoach Modules Into the Developed Digital Infrastructure With TSD Integration

- ✓ Install docker desktop in the local PC or laptop.
- ✓ Download a module (e.g., activity) from GitHub with clone or if its existing in the local codebase then perform pull operation.
- ✓ Create a database namely "ecoach" in PostgreSQL (e.g., V13.0) at TSD side.
- ✓ Create a table "TBL_ACTIVITIES" under eCoach database
- ✓ Configure activity module to PostgreSQL database in the environment property file.
- ✓ Compiling and installing activity module → create Dockerfile with OpenJDK:12-alpine → generate activity.tar docker image file.
- ✓ Upload the container image tarball (ecoach:activity) into TSD project via the TSD Data Portal (https://data.tsd.usit.no/).
- ✓ Load the docker image fhir.tar file (docker load -i activity.tar) in TSD.
- ✓ Run the image file (docker run -d 8080:8080 ecoach:activity) in TSD.

TSD acts as an authorization server (AS) in the OAuth2 workflow. An authorization point works on the AS, allowing our applications and HTTP endpoints to define our system's features. In our eCoach system, two actors who interact with

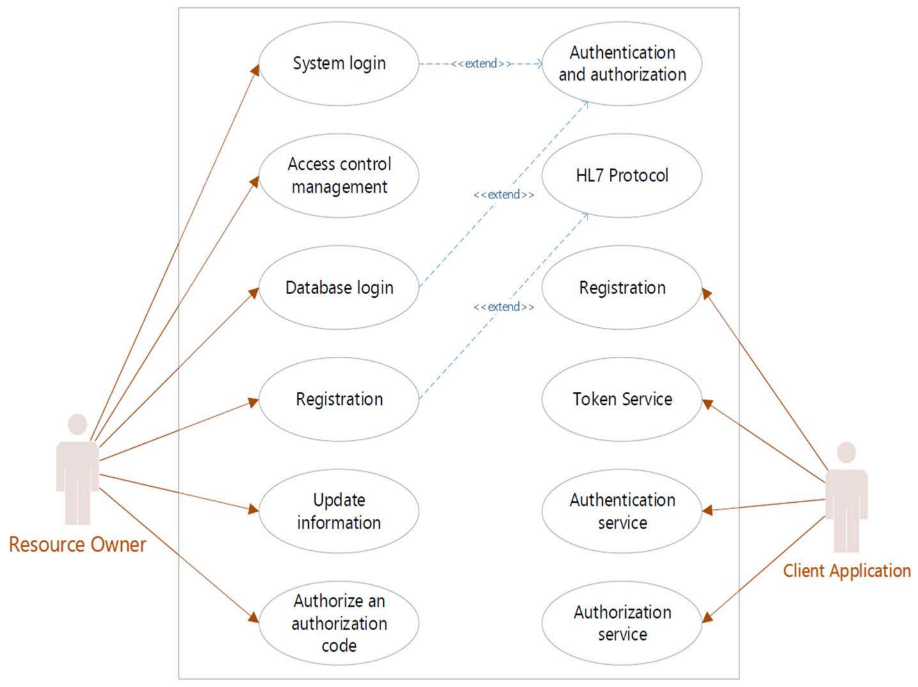


FIGURE 8. Authorization Server (AS) use-case for eCoach prototype system.

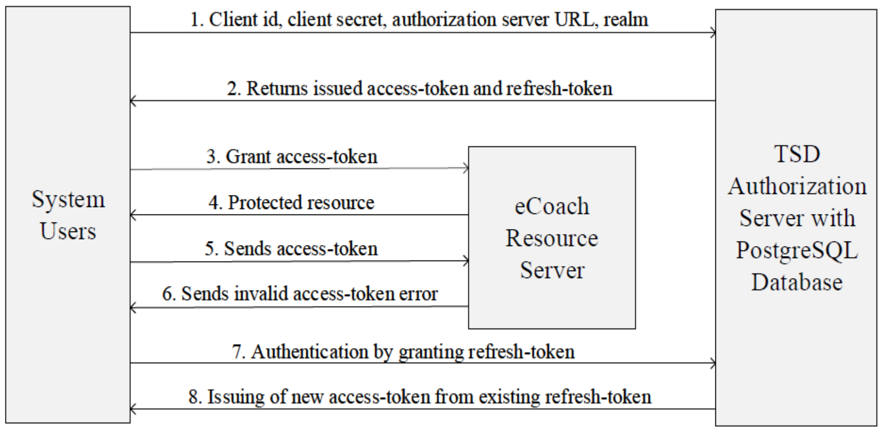

1. Client id, client secret, authorization server URL, realm
2. Returns issued access-token and refresh-token
3. Grant access-token
4. Protected resource
5. Sends access-token
6. Sends invalid access-token error
7. Authentication by granting refresh-token
8. Issuing of new access-token from existing refresh-token
System Users
eCoach Resource Server
TSD Authorization Server with PostgreSQL Database

FIGURE 9. Access-token generation from the existing refresh-token.

the AS are – ADMIN (resource owner) and USER (client registered with AS) as the fundamental use cases depicted in Figure 8. The resource server is an application that provides clients with an access token to access the HTTP endpoint resource server. It is a library set that includes HTTP end-points, static tools, and interactive web pages. OAuth2 is a mechanism for authorization to allow access to the client resources. We focused on the grant form (authorization code), client ID, and client secret to create an OAuth2 application.

JWT Token representing the claims between two parties is a JSON Web Token [25]–[27]. Such tokens are of two types: identity token (part of the OpenID Connect specification that is a client-dedicated function namespace) and access token (part of the OpenID Connect and OAuth2 specification and allows HTTP request that grants access to the service being invoked on). We used *asymmetric key encryption ES256*

TABLE 4. Specification of the experimental environment.

Specification	Details
Memory	15 GB
Operating System	GNU/Linux
Disk (HDD)	1023.9 GB
Socket endpoint	10.225.147.186:8443 (Class A private IPV4)

TABLE 5. Specification and format of the used HTTP headers for API testing.

Header	Specification	Usage in the HTTP request
Authorization	To authorize API access based on access token	Authorization: Basic <<token>> Authorization: Bearer <<token>>
Content-type	To define MIME type	Content-Type: application/json; charset=UTF-8
Cache-Control: no-cache, no-store, max-age=0, must-revalidate	Security header to ensures safe browsing history	Cache-Control: max-age=3600 Cache-Control: no-cache Cache-Control: no-store Cache-Control: no-transform Cache-Control: only-if-cached
Pragma: no-cache, Expires: 0	Security header to ensure safe browsing history	Pragma: no-cache
Strict-Transport-Security: max-age= <time in seconds>	Security header to ensure HTTP Strict Transport Security (HSTS)	Strict-Transport-Security: max-age=3600
X-Frame-Options: DENY	Security header to prohibit Clickjacking	X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block	Security header to prohibit Clickjacking	X-XSS-Protection: 1; mode=block

(*SHA256 with ECDSA algorithm*) to create a JWT signature. Access tokens are typically short-lived and frequently expire after only minutes. The additional refresh token sent by the login protocol allows a new access token to be accessed by the application after it expires (see Figure 9).

VI. EXPERIMENTAL RESULTS AND DISCUSSION

A. EXPERIMENTAL SETUP

Security testing method is intended to show vulnerabilities in an information system's security mechanisms that protect data and retain functionality as expected. Security assessments are carried out in many ways to verify security features. Here, we used unit testing as a security testing method. Unit testing was performed with Spring-Boot Swagger and Mock MVC framework (Mockito) to validate the security functionalities of different eCoach modules [50], [51]. In addition, we set up Apache JMeter to perform scalability testing of the adopted security solution in a digital health infrastructure. We executed the security solution in a Linux environment (see Table 4), protected with a VPN and network firewall. All the essential HTTP headers for the API testing are specified in Table 5.

B. EXPERIMENTAL RESULTS

This section discusses adopted security considerations, scenarios, and experimental outcomes. Experiments related to CSRF, XSS, Clickjacking, content sniffing, and brute force were performed with Mockito framework (v3.9.0), Swagger (v2.2.1), and curl system command (v7.76.1). In our Spring codebase (v2.4.5), we created five functional test cases with Mockito for TSD basic (client_id + password) user authentication, TSD two-factor user authentication (client_id + password + OTP), and role-based authorization (OAuth2) with an access token. We further implemented a negative test case where the user received an expected error response code HTTP 401 for an unauthorized resource API endpoint. Table 6 defines the combined result of Mockito test performance in a vanilla test setting where we compared our API response time with preferred, acceptable, and delayed response time in seconds (sec). The response metrics can be classified into the following categories – mean response time, peak response time, and error rate. We considered a preferred response time of 0.1 sec, an acceptable response time of <1 sec, and a delayed response time of <=10 sec. We received a mean response time in an acceptable time range for all the API responses.

The spring-boot application extends the security class *WebSecurityConfig* to protect against CSRF attacks. The CSRF tokens are powerful, changeable, and created as session tokens with specific properties that cannot be calculated or predicted by the intruder. Both HTTP POST forms in the "JSP" or template files need to introduce the CSRF token. If it is a JSON call, the token must be added to the HTTP request header. Initially, we disabled TSD token-based security setup and extended Spring's default web security configuration. Next, we executed the following four test cases for CSRF attack with Swagger: CSRF disabled (*valid credential, invalid credential*) and CSRF enabled (*valid credential and valid _csrf token, valid credential, and invalid _csrf token*). Successful authentication resulted in an *HTTP status code 200 or 201*. However, we disabled the CSRF token generation in actual security solution implementation. The reason for disabling CSRF was that our developed spring-boot application would be exposed to the public in the future. Therefore, we replicated similar and more robust web security standards with TSD's two-factor authentication and access token-based authorization. The successful test results are captured in Table 6. To ensure protection against

TABLE 6. Performance of unit-testing with Mockito framework.

Scenario	Input	Mean response	Category		
			Preferred response time (0.1 sec)	Acceptable response time (< 1 sec)	Delayed response time (10 sec)
Retrieval of short-lived access token with long-lived API Key	API Key, user_name, OTP, password	0.04 – 1 sec.	Yes	Yes	No
TSD basic authentication	client_id, password	<0.05 sec.	Yes	Yes	No
TSD two-factor authentication	client_id, password, OTP	0.04 – 1 sec.	Yes	Yes	No
Authorized access	Valid access token	0.05 – 1 sec.	Yes	Yes	No
Unauthorized access	Invalid access token	<0.07 sec.	Yes	Yes	No
Retrieval of short-lived access token with long-lived API Key	API Key, user_name, OTP, password	0.04 – 1 sec.	Yes	Yes	No

TABLE 7. Determination of protection against XSS, Clickjacking, and content sniffing with response headers.

Scenario	Parameters	Value
Input Parameters for HTTP POST	Endpoint	/ecoachui/authorize
	Port	8443
	HTTP Verb	POST
	Information	API Key, user_name, OTP, password
Data collection # 1	Response Header: X-XSS-Protection	1
	Response Header: mode	block
Data collection # 2	Response Header: X-Content-Type-Options	nosniff
Data collection # 3	Response Header: X-Frame-Options	DENY

XSS, content sniffing, and Clickjacking we explored TSD's security defense. We tested the attacks with a client's HTTP POST request with a Swagger API call and investigated whether XSS, Clickjacking, and content sniffing securities are allowed in the response header: setup data and recorded response headers are shown in Table 7. XSS is a category of security vulnerability found in web applications as cross-site scripting. The XSS attack was blocked by setting the value as "block" to the "X-XSS-Protection" parameter in the HTTP response header. Clickjacking is a method of tricking a user into clicking on something other than what the user perceives, thus potentially exposing sensitive information or allowing others to gain control of their device while clicking on harmless items, such as web pages. We blocked the Clickjacking by setting the "nosniff" value to the "X-Content-Type-Options" parameter in the HTTP response header. Content sniffing (or media type sniffing or MIME sniffing) is a method of tracking and inspecting the content of a "byte stream" to determine the file format of the data within it. We blocked the content sniffing by setting the "DENY" value to the "X-Frame-Options" parameter in the HTTP response header.

TABLE 8. Testing for the brute-force attack.

Scenario(s)	Comment
Normal successful login	No brute force detected
Normal login failure (<3 times continuously)	No brute force detected
Normal login failure (3 times continuously)	Exceeded maximum attempts allowed. Brute force attack detected, and account locked

We enabled configurable security defense to ensure data protection against brute force attacks. According to our coding, the user will get a chance of a maximum of three login failures before their account gets locked. The account will be activated based on the request ticket. Our analysis uses Spring-Boot, a counter to increasing the number of attempts with TSD to model the case attacker to execute the brute force attack. We set up the data in Swagger and recorded the response headers. Table 8 describes the unit test result of the brute force attack against three test cases.

TABLE 9. Scalability testing results with Y = 1, Z = 5, and variable loads (X).

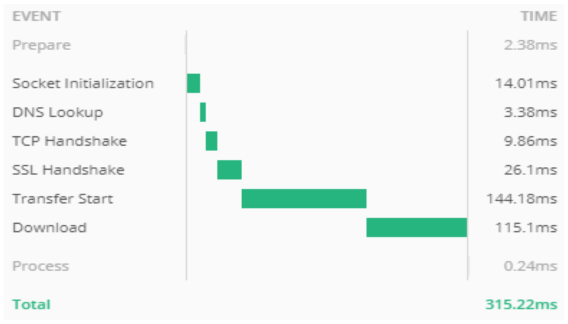
Y = 1, Z = 5	Mean Throughput	Error %	Received KB / sec	Delivered KB / sec	Mean Latency
Load (X)	(/sec)				(sec)
1	3.2	0	1426.43	3.45	275
10	15.1	0	6787	16.44	532
25	22.5	0	10079	24.41	962
50	24	0	10753.2	26.04	1751
75	24.9	0	11178.8	27.07	2563
100	25.2	0	11305.18	27.38	3344
200	26.5	0	11282.41	28.79	6302
300	21.1	0	3198.9	22.93	13102

TABLE 10. Scalability testing results with Y = 5, Z = 5, and variable loads (X).

Y = 5, Z = 5	Mean Throughput	Error %	Received KB / sec	Delivered KB / sec	Mean Latency
Load (X)	(/sec)				(sec)
1	4.4	0	1964.72	4.76	228
10	8.5	0	3801.19	9.21	187
25	19.9	0	8918.65	21.6	289
50	24.3	0	10890.75	26.38	989
75	24.9	0	11179.57	27.08	1863
100	24.9	0	11185.12	27.09	2773
200	25.3	0	11358	27.51	6392
300	22	0	3990	23.88	11716

TABLE 11. Scalability testing results with Y = 10, Z = 5, and variable loads (X).

Y = 10, Z = 5	Mean Throughput	Error %	Received KB / sec	Delivered KB / sec	Mean Latency
Load (X)	(/sec)				(sec)
1	4.5	0	2036.16	4.93	220
10	4.6	0	2067.12	5.01	179
25	11.2	0	5030.67	12.18	186
50	22.3	0	10012.17	24.25	305
75	24.6	0	11056.07	26.78	1000
100	24.9	0	11192.21	27.11	1869
200	25.4	0	11401.32	27.61	5515
300	25.7	0	11399	27.96	8938



EVENT	TIME
Prepare	2.38ms
Socket Initialization	14.01ms
DNS Lookup	3.38ms
TCP Handshake	9.86ms
SSL Handshake	26.1ms
Transfer Start	144.18ms
Download	115.1ms
Process	0.24ms
Total	315.22ms

FIGURE 10. Break-up of a response time in our SFTSDH solution for a single authorized HTTP request.

To perform scalability testing in JMeter, we selected an eCoach REST service with an approximated 1.19 KB of a request body, 448.3 KB of the response body, and a response time of 316 msec. (see Figure 10). Using JMeter "Thread Group" feature, concurrent threads or loads (X) were created with three different values of ramp-up seconds (Y) and a loop count value of five (Z). At each iteration, X*Z number of loads were created to capture mean throughput and mean latency time. The results are described in Table 9 – Table 11. A load is a numeric value representing a total number of concurrent requests or threads. The pseudo-code used for scalability testing is described in Textbox IV. We have considered the following values for scalability testing; however, the range can be increased for the future studies:

$$X = \{1, 10, 25, 50, 75, 100, 200, 300, 500\}$$
$$Y = \{1, 5, 10\}$$
$$Z = \{5\}$$

TABLE 12. Summary of the adopted functional and non-functional testing.

Test Case Scenario	Test Case	Passed (Yes/No)
Basic authentication	Access with a valid credential	Yes
	Access with an invalid credential	Yes
Two-factor authentication	Access with a valid credential + OTP	Yes
	Access with an invalid credential / an incorrect OTP	Yes
New user creation and role assignment	Creation request with a valid role assignment	Yes
Role-based API access	Access with a valid access key	Yes
	Access with an invalid access key	Yes
CSRF disabled	Access with a valid credential	Yes
CSRF Enabled	Access with a valid credential and valid "_csrf" token	Yes
	Access with a valid credential and invalid "_csrf" token	Yes
CSRF Disabled but Access Token enabled	TSD-based authentication and authorization	Yes
XSS Attack	Validate with response header if XSS attack protection is enabled	Yes
Brute Force Attack	General login failure	Yes
	Multiple (=6) login failure	Yes
Clickjacking Attack	Validate with response header if attack protection is enabled	Yes
Content sniffing	Validate with response header if attack protection is enabled	Yes
Total Test Pass Rate	**-**	**100%**

TABLE 13. Qualitative analysis on the effectiveness of SFTSDH with three flags – No (0), limited (1), and yes (2).

Features	Spring Security	Third-party Platform (e.g., Keycloak, Okta)	SFTSDH
OAuth2.0	Yes	Yes	Yes
SAML2.0	No	Yes	Yes
OpenID	No	Yes	Yes
WebFlux	Yes	Yes	Yes
Access control	Yes	Yes	Yes
Identity and Access Management - Single-Sign-On (SSO), Identity Brokering and Login, User Federation, Client Adapters	No	Yes	Yes
Entire process of seamless calling Authorization Server from Spring-boot	Yes	No	Yes
Robustness	Limited	Limited	Yes
Powerful and customizable	Limited	Limited	Yes
Handling of Java EE security constraints	Limited	Limited	Yes
Multi-factor authentication	No	Yes	Yes
Easy to use	Yes	No	Yes
JSON web token	No	Yes	Yes

The result shows a direct proportion between throughput and load, and latency time and load. However, reaching a certain threshold, the throughput drops with increased load.

C. DISCUSSION

API endpoint security implicitly guarantees personal health data privacy inside our implemented secure digital

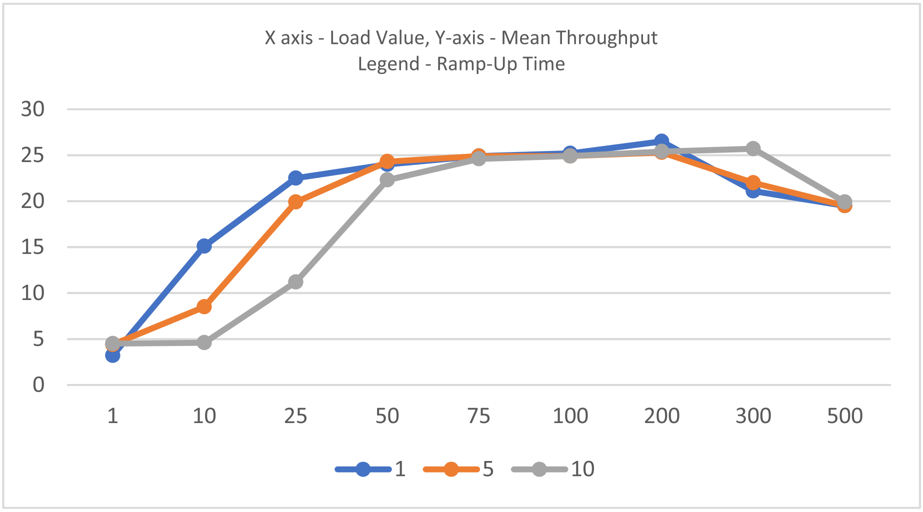


FIGURE 11. Mean throughput (/sec.) with increasing load for different ramp-up time in seconds.

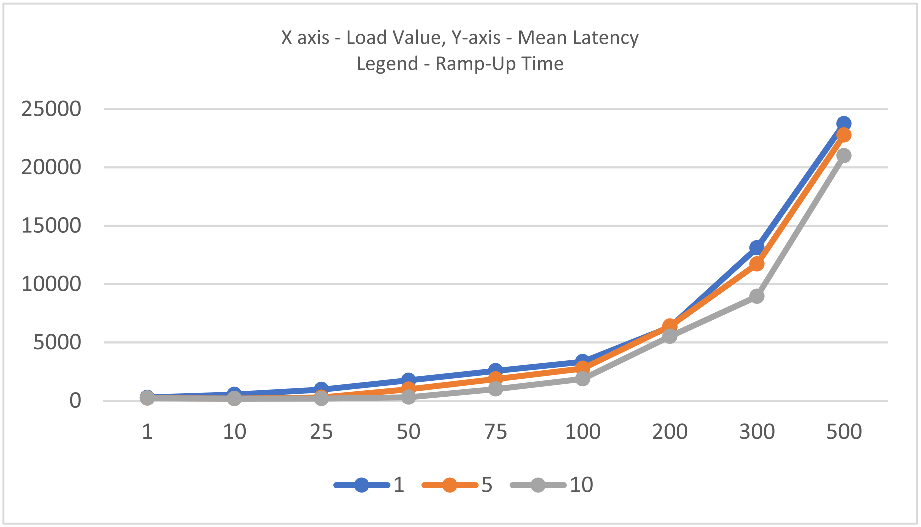


FIGURE 12. Mean latency (sec.) with increasing load for different ramp-up time in seconds.

infrastructure. It is three-fold research. First, we conceptualize a secure solution with SFTSDH, VPN, network firewall, and SSL. Second, we implemented the solution for developing a digital health infrastructure where we deployed an eCoach prototype system. Third, we perform testing of the eCoach prototype system's REST API endpoints against the common functional security testing (see Table 12) and scalability testing. We created 17 test cases for 11 test scenarios to evaluate the effectiveness of our adopted security solution. The SFTSDH implementation and configuration in the eCoach system have effectively secured the eCoach APIs from an attack in all the scenarios. Moreover, we performed a qualitative analysis on the effectiveness of SFTSDH in Table 13, after comparing SFTSDH with Spring Security and other Third-party Platform (e.g., Keycloak, Okta). Personal health data governance using the SFTSDH security solution has fulfilled the GDPR compliance checklist as specified in Table 14. Therefore, the SFTSDH solution is safe from the typical external illegitimate flooding requests as the external or exposed eCoach services are protected by a VPN and a firewall.

Due to licensing and subscription constraints, we could not create a similar environment to the TSD infrastructure and deploy other solutions (e.g., SF + Okta, SF + Keycloak, or solutions identified in literature) to test scalability against throughput and latency. Therefore, we only performed scalability testing for our work under defined settings and obtained a promising result, as depicted in Figures 11 and 12. The result shows an optimal throughput and latency at X = 200.

Textbox IV Pseudo-Code for Scalability Testing

```
Step 1: /*Initialization*/
            $L = \{n \mid n$ is an integer, and $n \in Z^{+}\}$
            $R = \{n \mid n$ is an integer, and $n \in Z^{+}\}$
            $C = \{n \mid n$ is an integer, and $n \in Z^{+}\}$
            $X \in L$
            $Y \in R$
            $Z \in C$
Step 2: /*Iteration and value capturing*/
            For i = 1 to size(Y) do
                y = Y(i)
                For j = 1 to size(X) do
                    x = X(j)
                    For k = 1 to size(Z) do
                        z = Z(k)
                        load = x * z
                        /*calculate metrices*/
                        Calculate throughput (requests/sec), error%,
            received data (KB/sec), delivered data (KB/sec), mean
            latency (sec) and add the results to a list (List)
                    End
                End
            End
Step 3: /*Iteration and metric value display*/
            For i = 1 to length (List) do
                display (List(i))
            End
```

TABLE 14. GDPR compliance checklist for SFTSDH.

GDPR checklist [53]	Addressed
Lawful basis and transparency	Yes
Data security	Yes
Accountability and governance	Yes
Privacy rights	Yes

Therefore, the digital health infrastructure with the SFTSDH solution can sustain a load of 200*5 = 1000 concurrent users efficiently.

Our future study will focus on the migration of the deployed services from *EduNet* to the TSD platform after resolving identified TSD constraints, such as accessing TSD endpoints from public network using wireless technologies, basic webpage rendering without authorization, calling third-party weather API from TSD, and generation of auto-notification inside TSD in a synchronous manner. Auto-notification support inside TSD will further reduce scheduling overhead in the eCoach mobile application.

VII. CONCLUSION

An API endpoint is an interface that helps in exchanging data between services. The service can access the expected resources from the API to perform the necessary operations. It plays a vital role in ensuring the regular and safe operation of the services and systems. Therefore, one of the leading security factors in MSA applications is API security. eCoach's REST API endpoint security solution with the TSD platform implicitly guarantees the privacy of personal health data within our adopted SFTSDH solution scheme following the GDPR and NORMEN regulations. This research accomplishes a prototype of an eCoach system that uses MSA with the TSD platform to evaluate technology and security integration. The research results show that the adopted digital solution effectively protects the APIs and personal health data (stored inside TSD). This study can be used as a TSD integration manual to protect personal health data in healthcare research.

ABBREVIATIONS

HIS:	Healthcare Information Systems
ICTs:	Information and Communication Technologies
IoMT:	Internet of Medical Things
SF:	Spring Framework
TSD:	Services for Sensitive Data
HTTP:	Hyper-Text-Transfer-Protocol
APIs:	Application Programming Interfaces
GDPR:	General Data Protection Regulation
eHealth:	Electronic Health
eCoach:	Electronic Coach
CSRF:	Cross-Site Forgery
XSS:	Cross-Site Scripting
CORS:	Cross-Source Resource Sharing
DoS:	Denial of Service
DDoS:	Distribution DoS
MITM:	Man-In-The-Middle Attack
CS:	Content Sniffing
BF:	Brute Force Attack
IP:	Internet Protocol
TM:	Therapeutic Manipulation
OWASP:	Open Web Application Security Project
HIPAA:	Health Insurance Portability and Accountability Act
NORMEN:	Norwegian Data Protection Policies
REST:	Representational State Transfer
JSON:	JavaScript Object Notation
IAM:	Identity and Access Management
TCP:	Transmission Control Protocol
IWA:	Integrated Windows Authentication
TLS:	Transport Layer Security
SSL:	Secure Socket Layer
SAML:	Security Assertion Markup Language
MSA:	Microservice Architecture
NSD:	Norwegian Center for Research Data
REK:	Regional Ethical Committee
EHRs:	Electronic Health Records
MSSA:	Microservice Solution Architecture Level
SSF:	Spring Security Framework
MQTT:	Message Queuing Telemetry Transmission
PoC:	Proof of Concept
PKI:	Public Key Infrastructure
BLE:	Bluetooth Short-Range Communication Technology
PSS:	Project Specific Servlets
PJS:	Patient-Journal-System

PGDs:	Person Generated Data
BPS:	Bits Per Second
RTT:	Round-Trip Time
IaaS:	Infrastructure as a Service
AS:	Authorization Server
ACL:	Access Control Rules

ACKNOWLEDGMENT

The authors would like to thank the University of Agder and coauthors for supporting them to perform this research.

REFERENCES

[1] J. T. Kelly, K. L. Campbell, E. Gong, and P. Scuffham, "The Internet of Things: Impact and implications for health care delivery," *J. Med. Internet Res.*, vol. 22, no. 11, Nov. 2020, Art. no. e20135, doi: 10.2196/20135.

[2] S. A. Parah, J. A. Kaw, P. Bellavista, N. A. Loan, G. M. Bhat, K. Muhammad, and V. H. C. de Albuquerque, "Efficient security and authentication for edge-based internet of medical things," *IEEE Internet Things J.*, vol. 8, no. 21, pp. 15652–15662, Nov. 2021, doi: 10.1109/JIOT.2020.3038009.

[3] *How the Internet of Medical Things is Impacting Healthcare*. Accessed: Jan. 25, 2022. [Online]. Available: https://healthtechmagazine.net/article/2020/01/how-internet-medical-things-impacting-healthcare-perfcon

[4] *Internet of Medical Things (IoMT) Market Size 2021*. Accessed: Jan. 25, 2022. [Online]. Available: https://www.marketwatch.com/press-release/internet-of-medical-things-iomt-market-size-2021-in-depth-analysis-market-dynamics-with-top-players-impact-of-covid-19-case-study-analysis-industry-impact-and-global-forecast-till-2026-2021-03-15

[5] A. Gatouillat, Y. Badr, B. Massot, and E. Sejdić, "Internet of medical things: A review of recent contributions dealing with cyber-physical systems in medicine," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3810–3822, Oct. 2018, doi: 10.1109/JIOT.2018.2849014.

[6] P. I. R. Grammatikis, P. G. Sarigiannidis, and I. D. Moscholiosb, "Securing the Internet of Things: Challenges, threats and solutions," *Internet Things*, vol. 5, pp. 41–70, Mar. 2019, doi: 10.1016/j.iot.2018.11.003.

[7] Y. Sun, F. P.-W. Lo, and B. Lo, "Security and privacy for the internet of medical things enabled healthcare systems: A survey," *IEEE Access*, vol. 7, pp. 183339–183355, 2019, doi: 10.1109/ACCESS.2019.2960617.

[8] F. Belqasmi, R. Glitho, and C. Fu, "RESTful web services for service provisioning in next-generation networks: A survey," *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 66–73, Dec. 2011, doi: 10.1109/MCOM.2011.6094008.

[9] J. N. S. Rubí and P. R. L. Gondim, "IoMT platform for pervasive healthcare data aggregation, processing, and sharing based on OneM2M and OpenEHR," *Sensors*, vol. 19, no. 19, p. 4283, 2019, doi: 10.3390/s19194283.

[10] S. Banerjee, V. Odelu, A. K. Das, J. Srinivas, N. Kumar, S. Chattopadhyay, and K.-K.-R. Choo, "A provably secure and lightweight anonymous user authenticated session key exchange scheme for Internet of Things deployment," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8739–8752, Oct. 2019, doi: 10.1109/JIOT.2019.2923373.

[11] A. H. M. Aman, W. H. Hassan, S. Sameen, Z. S. Attarbashi, M. Alizadeh, and L. A. Latiff, "IoMT amid COVID-19 pandemic: Application, architecture, technology, and security," *J. Netw. Comput. Appl.*, vol. 174, Jan. 2021, Art. no. 102886, doi: 10.1016/j.jnca.2020.102886.

[12] D. Lange-Kuitse, "The health of patient privacy: The patient's perspective on the HIPAA protected health information," Ph.D. dissertation, College Educ. Int. Services, Andrews Univ., Berrien Springs, MI, USA, 2007, doi: 10.32597/dissertations/1699.

[13] C. Kuner, L. A. Bygrave, C. Docksey, and L. Drechsler, *The EU General Data Protection Regulation (GDPR): A Commentary*, Mar. 2021, doi: 10.1093/oso/9780198826491.001.0001.

[14] *NORMEN*. Accessed: Jan. 25, 2022. [Online]. Available: https://www.ehelse.no/normen

[15] F. Gutierrez, "Spring with spring boot," in *Pro Spring Boot*. Berkeley, CA, USA: Apress, 2016, pp. 89–105. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-1431-2_5, doi: 10.1007/978-1-4842-1431-2_5.

[16] *TSD*. Accessed: Jan. 25, 2022. [Online]. Available: https://www.uio.no/english/services/it/research/sensitive-data/

[17] J. Hodges, C. Jackson, and A. Barth, *HTTP Strict Transport Security (HSTS)*, document RFC 6797, 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-04

[18] A. Chatterjee, M. Gerdes, A. Prinz, and S. Martinez, "Human coaching methodologies for automatic electronic coaching (eCoaching) as behavioral interventions with information and communication technology: Systematic review," *J. Med. Internet Res.*, vol. 23, no. 3, Mar. 2021, Art. no. e23533, doi: 10.2196/23533.

[19] A. Chatterjee, A. Prinz, M. Gerdes, and S. Martinez, "An automatic ontology-based approach to support logical representation of observable and measurable data for healthy lifestyle management: Proof-of-concept study," *J. Med. Internet Res.*, vol. 23, no. 4, Apr. 2021, Art. no. e24656, doi: 10.2196/24656.

[20] A. Chatterjee, M. W. Gerdes, and S. Martinez, "eHealth initiatives for the promotion of healthy lifestyle and allied implementation difficulties," in *Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2019, pp. 1–8, doi: 10.1109/WIMOB.2019.8923324.

[21] A. Chatterjee, M. W. Gerdes, A. Prinz, S. G. Martinez, and A. C. Medin, "Reference design model for a smart E-coach recommendation system for lifestyle support based on ICT technologies," in *Proc. 12th Int. Conf. eHealth, Telemed., Social Med. (eTELEMED)*, 2020, pp. 52–58.

[22] *Spring Framework*. Accessed: Jan. 25, 2022. [Online]. Available: https://spring.io/projects/spring-framework

[23] *HTTP Security: A Security-Focused Introduction to HTTP*. Accessed: Jan. 25, 2022. [Online]. Available: https://www.acunetix.com/blog/web-security-zone/http-security/

[24] W. Sun, Z. Cai, Y. Li, F. Liu, S. Fang, and G. Wang, "Security and privacy in the medical Internet of Things: A review," *Secur. Commun. Netw.*, vol. 2018, Mar. 2018, Art. no. 5978636.

[25] Q. Nguyen and O. Baker, "Applying spring security framework and OAuth2 to protect microservice architecture API," *J. Softw.*, vol. 14, no. 6, pp. 257–264, Jun. 2019.

[26] A. Dikanski, R. Steinegger, and S. Abeck, "Identification and implementation of authentication and authorization patterns in the spring security framework," in *Proc. 6th Int. Conf. Emerg. Secur. Inf., Syst. Technol. (SECURWARE)*, Oct. 2012, pp. 14–30.

[27] D. Recordon and D. Reed, "OpenID 2.0: A platform for user-centric identity management," in *Proc. 2nd ACM Workshop Digit. Identity Manage.*, 2006, pp. 11–16.

[28] J. J. Rodrigues, I. de la Torre, G. Fernández, and M. López-Coronado, "Analysis of the security and privacy requirements of cloud-based electronic health records systems," *J. Med. Internet Res.*, vol. 15, no. 8, p. e186, Aug. 2013.

[29] K. Bennett, A. J. Bennett, and K. M. Griffiths, "Security considerations for e-mental health interventions," *J. Med. Internet Res.*, vol. 12, no. 5, p. e61, Dec. 2010.

[30] H. Huang, T. Gong, N. Ye, R. Wang, and Y. Dou, "Private and secured medical data transmission and analysis for wireless sensing healthcare system," *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1227–1237, Jun. 2017.

[31] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. D. Xu, S. Kao-Walter, Q. Chen, and L.-R. Zheng, "A health-IoT platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2180–2191, Nov. 2014.

[32] P. Gope and T. Hwang, "BSN-care: A secure IoT-based modern healthcare system using body sensor network," *IEEE Sensors J.*, vol. 16, no. 5, pp. 1368–1376, Mar. 2016.

[33] A. Tejero and I. de la Torre, "Advances and current state of the security and privacy in electronic health records: Survey from a social perspective," *J. Med. Syst.*, vol. 36, no. 5, pp. 3019–3027, Oct. 2012.

[34] C. Papoutsi, J. E. Reed, C. Marston, R. Lewis, A. Majeed, and D. Bell, "Patient and public views about the security and privacy of electronic health records (EHRs) in the UK: Results from a mixed methods study," *BMC Med. Informat. Decis. Making*, vol. 15, no. 1, p. 86, Dec. 2015.

[35] M. A. Ameen, J. Liu, and K. Kwak, "Security and privacy issues in wireless sensor networks for healthcare applications," *J. Med. Syst.*, vol. 36, no. 1, pp. 93–101, Feb. 2012.

[36] T.-C. Hsiao, Y.-T. Liao, J.-Y. Huang, T.-S. Chen, and G.-B. Horng, "An authentication scheme to healthcare security under wireless sensor networks," *J. Med. Syst.*, vol. 36, no. 6, pp. 3649–3664, Dec. 2012.

[37] J. Kwon and M. E. Johnson, "Meaningful healthcare security: Does 'meaningful-use' attestation improve information security performance?" *MIS Quart.*, vol. 42, no. 4, pp. 1043–1067, Dec. 2018.

[38] K. Abouelmehdi, A. Beni-Hessane, and H. Khaloufi, "Big healthcare data: Preserving security and privacy," *J. Big Data*, vol. 5, no. 1, p. 1, Dec. 2018.
[39] C. S. Kruse, B. Smith, H. Vanderlinden, and A. Nealand, "Security techniques for the electronic health records," *J. Med. Syst.*, vol. 41, no. 8, p. 127, Aug. 2017.
[40] J. Salibindla, "Microservices API security," *Int. J. Eng. Res.*, vol. 7, no. 1, pp. 277–281, Jan. 2018.
[41] L. Xie, L. Han, M.-H. Li, and X.-L. Dong, "Design and implement of spring security-based T-RBAC," in *Proc. Int. Conf. Wireless Commun., Netw. Appl.*, 2017, pp. 183–188.
[42] K. Aloufi and O. Alhazmi, "Secure IoT resources with access control over restful web services," *Jordan J. Electr. Eng.*, vol. 6, no. 1, p. 64, 2020.
[43] M. I. Beer and M. F. Hassan, "Adaptive security architecture for protecting RESTful web services in enterprise computing environment," *Service Oriented Comput. Appl.*, vol. 12, no. 2, pp. 111–121, Jun. 2018.
[44] G. Serme, A. S. de Oliveira, J. Massiera, and Y. Roudier, "Enabling message security for RESTful services," in *Proc. IEEE 19th Int. Conf. Web Services*, Jun. 2012, pp. 114–121.
[45] F. D. Backere, B. Hanssens, R. Heynssens, R. Houthooft, A. Zuliani, S. Verstichel, B. Dhoedt, and F. D. Turck, "Design of a security mechanism for RESTful web service communication through mobile clients," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–6.
[46] A. S. Tanenbaum *et al.*, *Computer Networks*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996, pp. 1–17.
[47] M. A. Hussain, S. G. Langer, and M. Kohli, "Learning HL7 FHIR using the HAPI FHIR server and its use in medical imaging with the SIIM dataset," *J. Digit. Imag.*, vol. 31, no. 3, pp. 334–340, Jun. 2018.
[48] S. van der Weegen, H. Essers, M. Spreeuwenberg, R. Verwey, H. Tange, L. de Witte, and K. Meijer, "Concurrent validity of the MOX activity monitor compared to the ActiGraph GT3X," *Telemed. e-Health*, vol. 21, no. 4, pp. 259–266, 2015.
[49] S. Aljawarneh, "A web engineering security methodology for e-learning systems," *Netw. Secur.*, vol. 2011, no. 3, pp. 12–15, Mar. 2011.
[50] P. Khatiwada, H. Bhusal, A. Chatterjee, and M. W. Gerdes, "A proposed access control-based privacy preservation model to share healthcare data in cloud," in *Proc. 16th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2020, pp. 40–47.
[51] S. Acharya, *Mastering Unit Testing Using Mockito and JUnit*. Birmingham, U.K.: Packt Publishing, 2014.
[52] *GDPR Checklist for Data Controllers*. Accessed: Jan. 25, 2022. [Online]. Available: https://gdpr.eu/checklist/
[53] A. A. Ismail, H. S. Hamza, and A. M. Kotb, "Performance evaluation of open source IoT platforms," in *Proc. IEEE Global Conf. Internet Things (GCIoT)*, Dec. 2018, pp. 1–5.
[54] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed Internet of Things," *Comput. Netw.*, vol. 57, no. 10, pp. 2266–2279, 2013.
[55] E. Torroglosa-García, A. D. Pérez-Morales, P. Martinez-Julia, and D. R. Lopez, "Integration of the OAuth and web service family security standards," *Comput. Netw.*, vol. 57, no. 10, pp. 2233–2249, Jul. 2013.
[56] L. Malina, J. Hajny, R. Fujdiak, and J. Hosek, "On perspective of security and privacy-preserving solutions in the Internet of Things," *Comput. Netw.*, vol. 102, pp. 83–95, Jun. 2016.
[57] L. Babun, K. Denney, Z. B. Celik, P. McDaniel, and A. S. Uluagac, "A survey on IoT platforms: Communication, security, and privacy perspectives," *Comput. Netw.*, vol. 192, Jun. 2021, Art. no. 108040.
[58] L. Liu, D. Wang, J. Zhao, and M. Huang, "SA4WSs: A security architecture for web services," in *Information and Communication Technology*. Berlin, Germany: Springer, 2013, pp. 306–311.
[59] J. C. S. Santos, K. Tarrit, and M. Mirakhorli, "A catalog of security architecture weaknesses," in *Proc. IEEE Int. Conf. Softw. Architecture Workshops (ICSAW)*, Apr. 2017, pp. 220–223.
[60] M. Priyadarsini and P. Bera, "Software defined networking architecture, traffic management, security, and placement: A survey," *Comput. Netw.*, vol. 192, Jun. 2021, Art. no. 108047.
[61] D. Arkhipkin, J. Lauret, and P. V. Shanmuganathan, "Modular and scalable RESTful API to sustain STAR collaboration's record keeping," *J. Phys., Conf. Ser.*, vol. 664, no. 5, Dec. 2015, Art. no. 052021.
[62] A. Chatterjee and A. Prinz, "Applying spring security framework with keycloak-based OAuth2 to protect microservice architecture APIs: A case study," *Sensors*, vol. 22, no. 5, p. 1703, Feb. 2022, doi: 10.3390/s22051703.

AYAN CHATTERJEE received the B.Eng. degree in computer science and engineering (CSE) from the West Bengal University of Technology, India, in 2009, and the master's degree in information technology from Jadavpur University, India, in 2016. He worked as an Associate Consultant at Tata Consultancy Services Ltd., India, from 2009 to 2019, and was deputed to Denmark and the Netherlands for three to four years as a Java Architect and a Data Analyst. He is currently pursuing the Ph.D. degree with the University of Agder, Norway, with specialization in ICT-eHealth. His research interests include IoMT, AI, statistical analysis, persuasive computing, health data management, domain ontology, user-centered design, software engineering, and data mining.

MARTIN W. GERDES received the M.Sc. degree in electrical engineering (EE) from RWTH Aachen University, Aachen, in 1998, emphasis on (ICTs), and the Ph.D. degree in ICTs, with specialization in eHealth from the University of Agder (UiA), Grimstad. He has been an Associate Professor with the Department of Information and Communication Technology, UiA, with teaching and supervision experience, since 2014. His main research interests include eHealth solutions for remote monitoring and patient support, AI technologies, ICT, and the Internet of Things (IoT).

PANKAJ KHATIWADA received the B.Eng. degree in electronics and communication engineering from Kathmandu Engineering College, Nepal, in 2013, and the M.Eng. degree in information and communication technology from the University of Agder, Norway, in 2019. He is currently pursuing the Ph.D. degree with the Department of Security, Norwegian University of Science and Technology (NTNU), Norway, with a specialization in eHealth. He worked as a Project Engineer at Arya Nirman Sewa, Nepal, from 2013 to 2015, and responsible for handling different electrical and communications technologies projects. His research interests include health infrastructure, health data security, AI in healthcare, and blockchain in health data.

ANDREAS PRINZ received the M.Sc. degree in mathematics and the Ph.D. degree in computer science from Humboldt-University zu Berlin, Germany, in 1988 and 1990, respectively. From 2007 to 2015, he was the Head of the ICT Department, UiA, Grimstad, Norway, where he has been the Scientific Head of the eHealth Centre, since 2017. He is currently a Professor with the Department of ICT, UiA, with teaching and supervision responsibilities.