Future Generation Computer Systems 140 (2023) 209–224

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

The vendor-agnostic EMPAIA platform for integrating AI applications into digital pathology infrastructures

Christoph Jansen [a,*], Björn Lindequist [a], Klaus Strohmenger [a], Daniel Romberg [b], Tobias Küster [c], Nick Weiss [d], Michael Franz [a], Lars Ole Schwen [b], Theodore Evans [c], André Homeyer [b], Norman Zerbe [a]

[a] Charité – Universitätsmedizin Berlin, corporate member of Freie Universität Berlin and Humboldt-Universität zu Berlin, Institute of Pathology, Charitéplatz 1, 10117 Berlin, Germany
[b] Fraunhofer Institute for Digital Medicine MEVIS, Max-von-Laue-Straße 2, 28359 Bremen, Germany
[c] Technische Universität Berlin, DAI-Labor, Ernst-Reuter-Platz 7, 10587 Berlin, Germany
[d] Fraunhofer Institute for Digital Medicine MEVIS, Maria-Goeppert-Straße 3, 23562 Lübeck, Germany

ARTICLE INFO

Article history:
Received 1 July 2022
Received in revised form 5 October 2022
Accepted 23 October 2022
Available online 28 October 2022

Keywords:
Computational pathology
Artificial intelligence
Applications
API
Interoperability
Digital platform

ABSTRACT

Automated image analysis and artificial intelligence (AI) are becoming increasingly common in digital pathology software. While various proprietary pathology systems exist, there are no fully vendor-agnostic integration approaches for AI apps. This makes it difficult for vendors of AI solutions to integrate their products into the multitude of non-standard software systems in pathology.

The EMPAIA Consortium is developing an open and decentralized platform allowing AI-based apps of different vendors to be integrated with existing clinical IT infrastructures. For this purpose, we defined, analyzed, and prioritized relevant use cases and identified requirements for an open platform to support these use cases. We then designed the platform architecture described here to meet these requirements based on web technologies.

For all platform services open source reference implementations are available, that are used by developers of AI apps as an integration target. Developers of compatible clinical systems can either use and integrate components of the reference implementation or directly implement the interfaces as per specification, allowing apps to run in their clinical environment. Pathology laboratories can use both on-premises and cloud deployments of the platform. Apps can be obtained via a central marketplace so that pathologists can use them in their daily workflow.

An adoption of this platform will enable interoperability among different existing digital pathology software systems. This reduces integration efforts for software vendors, while users will benefit from a wider variety of tools and a quicker availability of new and innovative methods. Ultimately, the platform will reduce barriers to market entry for AI vendors and provide pathologists with access to advanced AI tools.

1. Introduction

With the advance of digitization in pathology, new opportunities for automated analysis and applications using artificial intelligence (AI) are emerging. Image analysis approaches, such as Convolutional Neural Networks, have greatly improved in recent years, many studies report successfully using AI in computational pathology [1–7], some approaches have been implemented as products for routine use [8,9]. These methods could supplement the pathologist's toolbox, e.g., to quickly quantify tumor cells and to reduce the amount of tedious and repetitive tasks for humans. This becomes increasingly important due to the increasing shortage of pathologists worldwide [10]. For small and medium-sized companies specializing in AI and image processing, it can be challenging to integrate their applications into a multitude of different and mostly non-standardized hardware and software systems currently present in the heterogeneous digital pathology landscape [11]. These systems include WSI scanners producing vendor-specific image formats, Image Management Systems (IMS), Vendor Neutral Archives (VNA), DICOM Picture Archiving and Communication Systems (PACS), Anatomic Pathology Laboratory Information Systems (APLIS), and digital pathology workstations providing local data processing capabilities, as well as a user interface (UI) for pathologists.

* Corresponding author.
E-mail address: christoph.jansen@charite.de (C. Jansen).

https://doi.org/10.1016/j.future.2022.10.025

The EMPAIA Consortium (EcosysteM for Pathology Diagnostics with AI Assistance) aims to reduce the barriers to market entry for AI vendors into the clinical routine of histopathological diagnostics by providing vendor-neutral API specifications. The specifications allow for a clear separation of concerns in digital pathology infrastructures and therefore enable the integration of apps in existing pathology systems via API adapters. EMPAIA platform services are integrated into existing local clinical infrastructures in a decentralized way, with the option for cloud-based service hosting per organization. A global app marketplace is used for distribution and provides marketing and billing capabilities for application vendors, with the long-term goal to foster a self-sustaining ecosystem.

1.1. State of the art

The EMPAIA Consortium is partnering with pathology laboratories, clinics, and university hospitals, to evaluate the current state of digital pathology. In 2021 the centers have answered questionnaires about the hardware and software components deployed in their routine workflows. The results shown in Table 1 have been provided by one Austrian and nine German centers, therefore representing a small portion of the global market. Although the number of participating centers is limited, the data already shows a strong fragmentation of the system landscape. In total, eight different APLIS, six different IMS/VNA/PACS, and 17 different WSI scanner models have been used at that time. In total, these components are provided by 19 different vendors. Each center is using an APLIS and at least one scanner WSI scanner model, but one center does not use a dedicated IMS/VNA/PACS software and instead uses a network attached storage to store WSI files. While D-Icom IS-P (Imassense Deutschland GmbH) is a combined APLIS+PACS solution, in all other cases the APLIS and IMS/VNA/PACS components are developed by separate vendors. In order to provide an improved user experience and workflow for pathologists, some vendors cooperate by creating one-to-one proprietary integrations between their systems. The IntelliSite Pathologist Solution (Koninklijke Philips N.V.) [12], pathoZoom digitalLab (Smart In Media AG) [13] and Digital Slide Suite (VMscope GmbH) [14] integrate AI/advanced image analysis models in the IMS that are being applied in the centers for routine diagnostics. These IMS either integrate their own AI solutions or cooperate with an external AI vendor to realize a proprietary integration. In total only four centers use AI/advanced image analysis applications in the clinical routine, developed by at most two different AI vendors per center.

1.2. Related work

In the pathology landscape, various vendor-neutral and vendor-specific options for integrating apps exist. Zeiss offers two solutions: Zeiss Zen [15] integrates with Zeiss microscopes and allows image processing via scripting and macros. The results can be examined directly in a viewer software. Zeiss Apeer [16] is a cloud-based digital image processing platform with a focus on research use cases. Compatible processing modules are packaged as container images. Module specifications describe inputs and outputs, allowing the modules to be combined as workflows. With HALO [17] and HALO AI [18], Indica Labs offers a research-focused platform that enables data access via GraphQL queries or a Python SDK. The Sectra Digital Pathology Module [19] provides a vendor-neutral API to integrate apps, again aiming for routine clinical use. Cytomine [20] is a server-based system that allows for remote image inspection, analysis, and collaboration. Apps are packaged in container images and run as jobs, communicating via the Cytomine HTTP API. The open DICOM standard, originally developed for radiology imaging, has been extended to support WSIs [21,22]. In addition, WSI access via DICOMweb is currently being developed [23,24], but is not yet sufficiently stable for general use. Basic support for DICOM WSI files is already available in the EMPAIA platform (Section 2.2.2), but offering a DICOMweb interface is an ongoing research topic.

Moreover, new commercial cloud-based platforms for digital pathology have been announced, namely the Sectra Amplifier Marketplace [25] and the Roche Digital Pathology Open Environment [26]. Similar platforms for AI integration also exist in the domain of radiology (e.g., [27]).

1.3. Contribution

The hypothesis for the present article is that it is possible to design an open and modular architecture of a digital pathology platform for routine use. Openness here means enabling interoperability among existing digital pathology infrastructure systems, as well as between such systems and image analysis apps, in particular interoperability between systems by different vendors. The EMPAIA Consortium defines a reference architecture and specifies APIs in close collaboration with hardware and software vendors, that participate in the ecosystem. The open-source reference implementation by itself is not a certified medical device software and does not replace the existing software solutions available in the market. While the AI platforms mentioned in Section 1.2 allow different AI vendors to integrate their applications, these integrations are always platform specific. In contrast, EMPAIA aims to standardize an integration path for AI vendors, to be compatible with many existing AI platforms and pathology software systems at once. This many-to-many relation has the potential to act as a multiplier to accelerate the adoption of AI in pathology. An up-to-date list of industry partners is available on the project website [28].

The project is currently focusing on routine diagnostics, helping industry partners to overcome barriers to market entry in terms of technical integration, validation, and certification. Once the adoption in clinics has increased, research use-cases will benefit as well. For example, AI developers will be able to provide their research-grade applications via the marketplace, allowing pathologists to easily evaluate the latest academic achievements. Such an integration could also lower the barriers to conduct multi-centric studies, where AI solutions are applied in a clinical environment on local data sets, without the need to transfer data to the researchers.

This manuscript is based on a previous publication presented at the Workshop on Clusters, Clouds and Grids for Life Sciences – CCGrid Life 2022 [29], but extending on how requirements were derived and elaborating on clinical integration approaches. We identify key use cases and requirements for AI-enabled software platforms in pathology, describe the design decisions and the resulting architecture of the platform, and how the platform meets the requirements. Extensive API descriptions are not in the scope of this manuscript. However, a previous publication [30] covers the App API specification for image processing and AI applications, comprising an app description format, an HTTP API, container technologies for the deployment, and an open source app test suite (see Section 2.5). Further APIs for the decoupling of web-based UIs in the frontend and the connection to storage, compute, and information systems in the backend are in active development. Up-to-date API specifications in the form of OpenAPI [31] documents and text descriptions, as well as development tutorials, are available in a public documentation [32].

Table 1
Software and hardware components deployed in one Austrian and nine German pathology centers (university hospitals, clinics, and pathology laboratories of different sizes). Vendors are sorted alphabetically and components are grouped by APLIS, IMS/VNA/PACS and WSI Scanner models. Data has been reported by the centers in a 2021 questionnaire initiated by the EMPAIA Consortium, illustrating the heterogeneous landscape of systems.

Vendor	APLIS	IMS/VNA/PACS	Scanner
Basys Data GmbH	PathoWin+		
dc-systeme Informatik GmbH	dc-Pathos		
F. Hoffmann-La Roche Ltd.			VENTANA DP200, VENTANA iScan HT Brightfield, VENTANA iScan Coreo Au
Grundium Ltd.			OCUS
Hamamatsu Photonics K.K.			NanoZoomer 2.0-HT
ifms GmbH	PathoPro		
Imassense Deutschland GmbH	D-Icom IS-P	D-Icom IS-P	
Koninklijke Philips N.V.		IntelliSite Pathologist Solution	Ultra Fast Scanner
Leica Biosystems Nussloch GmbH		Aperio eSlide Manager	Aperio AT2, Aperio Versa 200, Aperio GT450
NEXUS/AG	NEXUS PATHOLOGIE		
NEXUS/Paschmann GmbH	PAS.NET		
NEXUS/SWISSLAB GmbH	SWISSLAB		
Objective Imaging Ltd.			Glissando Desktop Scanner
PerkinElmer Inc., Akoya Biosciences Inc.			Vectra 3.0
PROGRAMMIERFABRIK GmbH	PAS Xanthos		
Sakura Finetek Europe B.V.			VisionTek Live
Smart In Media AG		PathoZoom digitalLab	
VMscope GmbH		Digital Slide Suite	
3DHISTECH Ltd.		SlideCenter (former CaseCenter)	P1000, P250, Pannoramic MIDI, Pannoramic SCAN II, DESK II DW

1.4. Use cases and requirements

We conducted a use case and requirements analysis for AI-based pathology software platforms. The EMPAIA Consortium initially identified a total of 27 use cases in discussion rounds and workshops. From these original use cases, the 8 most important key use cases have been selected as a basis for the development work and were specified in detail using Business Process Model and Notation (BPMN) and component diagrams. The full list of original use cases is available in Appendix A. The key use cases have been further aggregated and prioritized, resulting in the 3 use cases presented in this manuscript. While the use cases were focused on the initial release of the EMPAIA platform, we have already considered a broader vision in the requirements analysis, including regulatory and economic aspects as well as existing software and hardware vendors. Hence, we prioritized requirements in what should already be available in a first version of the platform and what should be implemented later. Specific requirements for image processing and AI apps have been addressed before [30] and are not covered in this manuscript.

The following critical use cases were identified:

- Use case 1: Pathologists using apps via the Workbench Client web UI (Section 2.2.3)
- Use case 2: Registering users and organizations via the Portal web UI (Section 2.2.1)
- Use case 3: Displaying apps available in the marketplace via the Portal web UI

Furthermore, the following general requirements were identified:

R1: Central marketplace and user management.

- authentication and authorization (auth)
- visibility of apps
- quality control of apps, e.g., based on clinical validation
- allow for later implementation of accounting services (app usage, billing, telemetry) to commercialize the platform and build a self-sustaining ecosystem

R2: Public API specifications.

- allow software vendors to take part in the platform ecosystem by following these specifications
- allow software vendors to be part of the API specification process
- cross-vendor software compatibility

R3: Open core platform.

- provide open source reference implementations of the core infrastructure services
- allow software vendors to test their own API implementations against existing reference implementations
- allow software vendors to use and build upon existing software components

R4: Distributed core platform.

- one service stack per pathology laboratory
- data separation between laboratories
- deployment on-premises, in a cloud, or across both (control over data storage and processing locations)
- integration into the local system via API implementation or API adapters (e.g., APLIS, PACS, VNA, Digital Pathology Workstations)

R5: Extensibility.

- first, focus on common features used by most apps
- extend with more specialized features later

R6: Web-based systems.

- accessible via modern browsers (no installation or update required on workstations)
- compatible with IT security restrictions in clinic networks
- well-known software engineering approach accepted by developers

R7: Testability.

- provide app test suite for app developers
- provide extensive platform tests for service integration testing

R8: Traceability.

- all data entities and app runs have unique IDs
- precise documentation of algorithm version and data inputs that produced certain outputs

R9: Visual diagnostic workflow.

- users can mark regions of interest for processing
- algorithms can return various geometric annotations (e.g., polygons, points) on histological images
- provide flexible class namespaces to classify annotations (e.g., tumor cell, non-tumor cell)

R10: Automation.

- later support automated pre-processing of long-running computations, i.e., trigger apps as soon as a new WSI has been produced
- later also support automated quality assurance (e.g., image sharpness analysis)

R11: Legal and regulatory concerns.

- provide data separation and access restrictions
- provide anonymization/pseudonymization capabilities for research use cases
- later support regulatory approval of apps and platform integrations

2. Methods

This section describes the core technologies used in the platform, as well as the platform architecture with respect to the requirements defined in Section 1.4. Furthermore, the platform's authentication mechanisms and the EMPAIA App Test Suite are explained.

2.1. Core technologies

The following subsections summarize the core technologies used for the platform's reference implementation. Alternative platform implementations by third-party software vendors can opt for a different technology stack. Only the use of OAuth2 [33] for authentication and authorization is a fixed requirement enforced by a global auth service. In this manuscript, we use the term auth to refer to authentication and authorization at the same time when a clear distinction is not required in the given context. The open source reference implementation can also be used independently of the global infrastructure for development and testing purposes.

2.1.1. Whole slide images

WSIs are high-resolution scans of histopathological tissue samples. Slide scanners of various vendors use their own proprietary file formats [34] that store not only the full-resolution image but also a pyramidal structure of pre-computed downsampled versions of the image. These image layers are usually stored in a tile-based fashion allowing WSI viewing software to access a certain field of view at a particular resolution in a fast and memory-efficient way. For example, the desktop viewing software QuPath [35] uses software libraries such as OpenSlide [36] as an abstraction over different WSI file formats, retrieving image data by coordinates and resolution level. Web-based viewers do not directly access the file data but instead call a web API, where the server implements the file access via external image libraries. Such an API provides endpoints to obtain image metadata and individual regions or tiles required to cover a desired resolution and viewport.

The Digital Imaging and Communications in Medicine (DICOM) supplement 145 has been published in 2010 [22], specifying the usage of the DICOM standard for WSI data. Since then, the adoption of DICOM in digital pathology has slowly been increasing while proprietary formats are still most prevalent in routine usage [37]. The Big Picture project of the innovative medicines initiative (IMI) is improving the tooling around DICOM WSI [38,39], including open source Python packages for conversion and accessibility. The open source wsidicomizer [39] is able to convert many proprietary WSI formats to DICOM, while wsidicom [38] allows programmers to obtain WSI image tiles from a DICOM file. Therefore the library provides an API that is comparable to OpenSlide Python and abstracts away the complexity of the DICOM standard. Furthermore, WSI support is being added to the HTTP-based DICOMweb protocol [37], with an early functional implementation in the Google Cloud Healthcare API [40].

2.1.2. Software and tools

The decentralized service architecture introduced in the present manuscript is mainly developed using Python 3.8 with the modern HTTP API framework FastAPI [41]. FastAPI enables the asynchronous processing of incoming HTTP requests via Python's asyncio implementation. Multiple FastAPI worker processes can be started via the Asynchronous Server Gateway Interface (ASGI) [42] implementation Uvicorn [43] to fully leverage the available compute resources. Service-to-service communication is implemented with the Python library aiohttp [44], enabling asynchronous web requests and can be used in an asynchronous FastAPI request handler. It also allows the implementation of proxy routes, where the HTTP request content of one service is streamed through another service. This is especially useful when building multiple API layers to provide differing levels of abstraction and authorization.

Services are bundled using Docker container images. Docker supports the Open Container Initiative (OCI) [45] image specification, such that the container runtime is exchangeable. Test and production environments are deployed on Linux servers using docker-compose [46], although docker-compose could be replaced with a more powerful orchestration framework (e.g., Kubernetes [47]) as soon as the need arises. Docker also allows for developer deployments on a local Linux, Windows, or Mac machine.

2.1.3. OpenAPI

The OpenAPI specification [31], formerly known as Swagger, has become the industry standard for documenting web APIs [48]. OpenAPI allows for static definitions of API endpoints, their URL and query parameters, expected HTTP response codes, error messages, as well as models of JSON documents sent in the HTTP message body. The documentation itself is stored in a JSON file that can be parsed to generate a documentation page. In addition, the OpenAPI definition can be interpreted by software libraries to auto-generate client code [49]. FastAPI has built-in support for OpenAPI, enabling access to the API documentation at runtime.

2.1.4. OAuth2

OAuth2 [33] is a standardized protocol for authentication (confirming identity of a client) and authorization (confirming client access permission for specific resources) in software systems. Implementing an OAuth2 mechanism in a web-based service platform must cover two major steps. First, a client must retrieve an access token from an auth provider (service) using one of many OAuth2 flows. Second, a client must include the token in HTTP requests to a service endpoint such that the service endpoint can validate the token to grant or deny access to the resource.

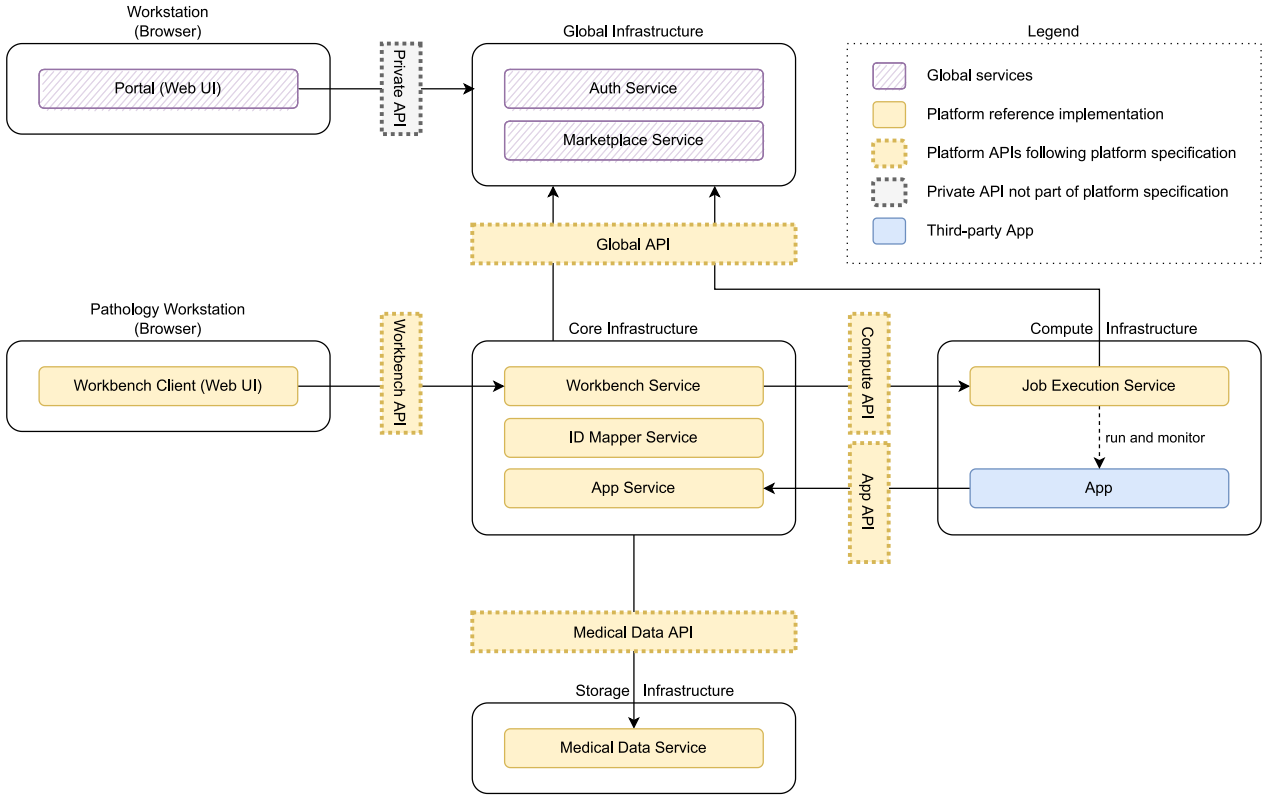

Fig. 1. Platform Architecture: Global Services for central user-/organization-management, authentication, and app distribution are rendered in purple. Decentralized Platform Services for pathology laboratories are rendered in yellow. Solid arrows denote the direction of HTTP API requests.

2.2. Service architecture

The software service architecture of the envisioned platform was designed to satisfy the requirements defined in Section 1.4. It also satisfies the three key use cases, with extensibility for future use cases already in mind. An overview of the architecture is shown in Fig. 1. The global infrastructure (purple) is hosted in the EMPAIA cloud and exists exactly once. The distributed infrastructure (yellow) is hosted once for each clinic/laboratory, either on-premises, in the cloud, or partly on-premises and partly in the cloud (Section 3.4). The distributed platform services are categorized as storage, compute and core services. All of these logical units communicate via HTTP APIs that serve as an abstraction layer and enable compatibility between units. The following sections describe the individual units of the platform architecture in the context of their APIs.

2.2.1. Global API

The Global API is comprised of Auth and Marketplace services. The Auth Service provides a global organization, user and role management. Each organization has admins and unprivileged members that gain access to decentralized services through a centralized OAuth2 mechanism. The marketplace hosts apps and app metadata. The EMPAIA Portal is a central Web UI that gives access to the user, organization and app management. In addition, it provides a public storefront where available apps are presented. These components implement the complete behavior of use cases 2 and 3 and fulfill the requirements R1 and R6. The global services are self-contained and do not depend on the distributed services of the platform.

2.2.2. Medical data API

The Medical Data API functions as an abstraction layer for storage systems and is provided by the Medical Data Service as a reference implementation. The API gives access to clinical metadata (e.g., cases and WSI metadata), WSIs in the form of image tiles and regions cropped from the requested WSI image level, geometric annotations with coordinates on the image data (e.g., polygons; R9), primitive data (e.g., floating-point score of an app result), collections for grouping data elements and jobs containing references to input and output data of an app execution. A job is a first-class data structure stored alongside the medical data, such that it is always known which app version has produced output data based on specific input data. Data entities that serve as job inputs or outputs become immutable to ensure traceability (R8). All data entities, including jobs, are referenced by unique IDs. Therefore it is possible to reuse the data across multiple jobs and also to query input and output data by their associated job ID. The Medical Data Service itself is implemented as an HTTP API layer that relays requests to an underlying microservice architecture (not shown in Fig. 1) and satisfies the requirements R2, R3, R4, and R6. The Medical Data API specification only covers API endpoints required by core components, namely the Workbench Service and the App Service. The API explicitly does not cover how clinical data is transferred into the platform because this can be implemented in many different ways, depending on the clinical infrastructure. Integration approaches with clinical software systems are described in Section 3.5.

2.2.3. Workbench API

The Workbench Service is part of the core infrastructure, and there exists one service instance per organization. The Workbench Service has two main purposes, first to provide the Workbench API for the Workbench Client, a web UI reference implementation for pathologists shown in Fig. 2; and second to aggregate and transmit data between services that do not have direct connections due to system architecture considerations. The

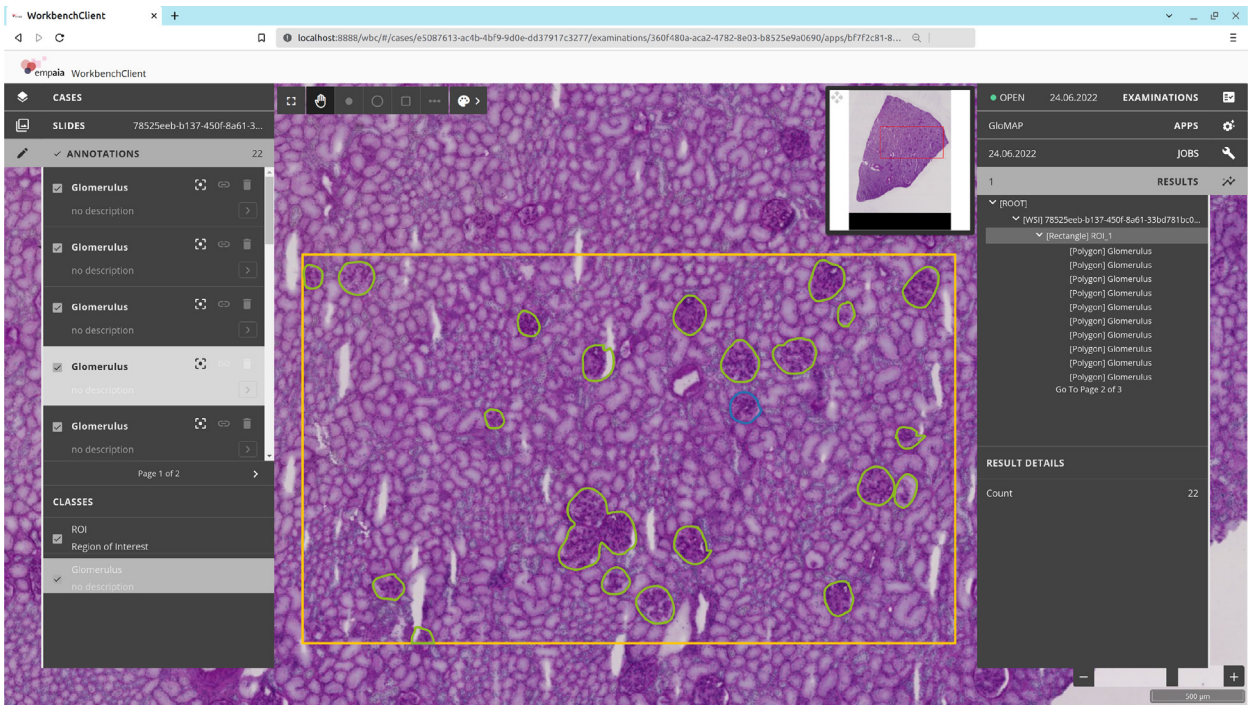

Fig. 2. Workbench Client: A browser-based web UI for pathologists, that enables the usage of EMPAIA compliant image processing apps. The screenshot shows annotations on a WSI that are the result of a successful job.

Workbench Client triggers the execution of a new job via the Workbench Service, which connects to the Medical Data API and the Job Execution API to synchronize the current states of running jobs. The process flow of use case 1 that is controlled by the Workbench Service is described in Section 3.1. In addition, the Workbench Client contains a viewer for pathological image data and geometric annotations that can be utilized for classification (e.g., as tumor or non-tumor). Classified geometric annotations produced by an app are intended to explain more abstract results (R9), such as a calculated tumor/non-tumor ratio, and provide pathologists with the necessary insights to evaluate and trust the output.

The automated pre- and post-processing of apps (R10) is an additional use case that will be covered in a future version of the platform.

The ID Mapper Service maps platform internal IDs of cases and WSIs to local IDs of a pathology laboratory. This mapping allows the platform to rely on pseudonyms that are only resolved via the Workbench Client/Workbench Service when the user views the data (R11). As long as the Workbench Service and ID Mapper Service are deployed on premises, it is possible to set up the data storage and processing in a cloud environment without the need to upload identifying patient data. Hence, WSI files may need to be anonymized/pseudonymized before uploading data to a cloud environment. Various deployment scenarios are evaluated in Section 3.4.

2.2.4. Compute API

The compute environment is managed by a Job Execution Service. This service executes apps from the platform marketplace based on given job data. The app runs as a headless process in a container and is able to communicate with the App API. The Job Execution Service itself never contacts the App API or any other core service, but only the global marketplace and authentication services. In case a job failure is detected by the Job Execution Service, the Workbench Service fetches the status change using a polling strategy (R6). The Workbench Service then transfers the status update to the Medical Data API. Depending on the Job Execution Service implementation, the service can distribute jobs in a cluster of compute nodes, including nodes with GPU resources to accelerate image processing operations and AI apps based on Artificial Neural Networks.

2.2.5. App API

The App Service is part of the core infrastructure and provides an App API [30]. It is used by apps to query job input data, send job output data, and to finalize a job. For successful API access, the app requires job ID, job token, and App API URL environment variables that the Job Execution Service initializes in the app container on startup. The App API serves as an abstraction over the Medical Data API with a limited (job-based) access scope. There exists exactly one App Service instance per pathology laboratory, and each instance always connects to exactly one Medical Data API of the corresponding organization (R2, R4). Fig. B.5 in Appendix B shows an exemplary HTTP request sequence of a very simple AI app, using the App Service in the platform reference implementation with the underlying Medical Data Service and its microservices.

2.3. Authentication and authorization

The auth mechanisms are coordinated by a global auth service (R1) that follows the OAuth2 standard (Section 2.1.4). A user login is performed via a web client using the Authorization Code Flow [33, Sec. 1.3.1]. Since services can be deployed in a distributed environment, potentially using untrustworthy networks, a service-to-service auth is mandatory. Services that connect to other services use a Client Credentials Flow [33, Sec. 1.3.4].

In an organization (e.g., pathology laboratory), it should not be possible for every user to connect to every platform service and for every service to connect to every other service. Therefore, the platform uses an audience system, where users and services explicitly get assigned audiences they may access (R11). These audiences are encoded in JSON Web Tokens (JWT) [50] by the

global auth service, such that a service receiving a request including the token is able to check whether or not itself is mentioned as audience. If not, the authorization to access the API is rejected. Using the audience system, every service only needs to know its own audience identifier that is provided to the services as a configuration setting. The alternative approach, where each service has a list of all clients allowed to send requests, would be an error-prone configuration overhead that is not feasible for a large, distributed service architecture. In addition to the described intra-organization authorization, this audience system also covers inter-organization authorization, where user/service access from one organization to another organization is blocked by default but could be enabled for research or study collaboration purposes. Another possible scenario could be an organization hosting only a Job Execution Service providing compute capabilities for smaller pathology laboratories or even clinics. A namespace structure is used to assign a unique audience identifier to each service of each organization.

The Medical Data Service and App Service provide a separate auth system for the App API. When a job is started, the Workbench Service retrieves a JWT created and signed by the Medical Data Service. This token is handed to the Job Execution Service alongside the remaining job data. The Job Execution Service then hands the token to the actual app that processes the job in the compute cluster. The app uses the token to connect to the App Service that in turn can validate the token using the public key of the Medical Data Service. This token contains the jobs ID, such that authorization is only granted to retrieve the job input data and to write appropriate output data that is expected from the app based on its app description [30]. This job-based token mechanism cannot be provided by the global auth service because it does not have any connection to the Medical Data Service deployed for the various organizations. In the future, this approach could be extended to further restrict resource access for users and services, for example, only allowing pathologists to access a subset of cases stored in the Medical Data Service of their organization.

2.4. Service development

The EMPAIA platform provides web API specifications (R2, R6) that can be implemented and/or consumed by third-party pathology software vendors. In order to properly design and test these APIs, the consortium also provides a reference implementation of web services and clients (R3). This is necessary because API specification tools like OpenAPI (Section 2.1.3) can only document static definitions of routes, parameters, and models but do not reflect dynamic API behavior implicitly defined by the backend code. Furthermore, third-party implementers might want to program a client and test it against an existing server or might want to implement an API server and check whether or not it is compatible with a reference client (R7). APIs are versioned to allow for extensions (R5) and even breaking changes in the future. Old API versions can co-exist with new versions while being deprecated for new apps, clients and other services.

2.5. App development

App developers need a way to develop and test their app before uploading it into the marketplace (R7). For this purpose, the EMPAIA App Test Suite (EATS) [51] can be used on a local computer. The EATS is open source and contains all relevant platform services required to run an app (R4). It provides a commandline interface to start/stop services, register docker images as apps, register/run jobs, and export data from the services to JSON files. It also contains a web client to examine WSIs and input/output annotations in a viewer. The EATS is entirely self-contained, without a connection to global services. Therefore, user and service-to-service auth is turned off, but the job token auth provided by the Medical Data Service and App Service (both contained in the EATS) is turned on, as it is relevant to an app and must be testable.

2.6. App catalog

Five software vendors have already adapted a total of ten apps to the EMPAIA App Interface [30]. All apps use image processing and AI technologies to process user-selected regions of interest in a WSI. Analyses are available for H&E stains, immunohistochemistry (IHC) and fluorescence in situ hybridization (FISH). The current app catalog contains two apps based on H&E stains from non-small cell lung cancer (NSCLC) tissue. For IHC, there are two apps for the analysis of Ki-67-IHC from breast cancer tissue, two apps for HER2-IHC also from breast cancer, and one app each analyzing IHC for ER, PR, and P53 in breast cancer. One app that is based on FISH is available for the detection of HER2/neu gene amplification in breast and stomach carcinomas. The integrations of additional apps are under active development.

3. Results

The distributed platform services, the Workbench Client (Section 2.2.3), and the EATS (Section 2.5) are open source and can be accessed on gitlab.com [52]. The following sections describe how the initial use cases (Section 1.4) have been implemented. In addition, data throughput performance tests are reported (Appendix C.2).

3.1. Use case 1: pathologists using apps

Use case 1 (Section 1.4) describes the examination of a pathology case aided by image processing apps. Such an examination is conducted by pathologists via a graphical UI. The Workbench Client is a reference implementation provided by the EMPAIA Consortium to demonstrate how this use case can be implemented. Software vendors can integrate similar functionality into existing pathology software based on the Workbench API as it is defined by the platform.

When pathologists open the Workbench Client in the browser, they are forwarded to a login page. The users enter their login credentials and are redirected back to the Workbench Client. The authentication process is the only client interaction with an API that is not proxied by the Workbench Service (Section 2.2.3). After a successful login, the user is prompted with a list of cases for selection. For this purpose, the Workbench Service fetches the case data from the Medical Data Service (Section 2.2.2) and aggregates additional data about slides, examinations and jobs of the case. As soon as a case is selected, the pathologists can start browsing the case-related data. This mainly includes WSI (Section 2.1.1) and user-created geometric annotations located on these slides. The slide data includes metadata, image tiles, as well as overview, macro, and label images.

To use image processing apps, the pathologists must select an existing examination or create a new one if no open examination exists for this case. Multiple examinations per case can exist, but there should only be a single open examination at a time. In the future, clinical reports could be generated as soon as an examination is closed, summarizing the results obtained from apps.

Apps obtained from the marketplace must be explicitly assigned to an open examination before they can be used. For this purpose, the Workbench Service fetches app metadata from the

global marketplace API and stores the assignment of an app to an examination in the Medical Data Service. The user can then start a new job for this app. In the current version of the Workbench Client, the job parameterization is limited to the selection of a single WSI from the case and drawing/selecting one or more regions of interest on the selected WSI to be analyzed. In a future version of the Workbench Client, apps will ship custom UI elements that allow for a more sophisticated job parameterization. The job is then stored in the Medical Data Service, and the Workbench Service sends the job data to the Job Execution Service (Section 2.2.4) for processing. The Workbench Service regularly updates the job status in the Medical Data Service as reported by the Job Execution Service. As soon as a job is finished, the user can view the results that were written to the Medical Data Service through the App Service. Again, the result view in the current Workbench Client version is very generic, although custom visualization will be possible in the future via app UI integrations.

Pathologists can add multiple apps to an examination and run multiple jobs per app. An examination can be closed as soon as all jobs in the examination are finished.

3.2. Use case 2: registering organizations and users

Users can register on the platform on their own behalf via the EMPAIA Portal. By default, users do not have access to any organizational resources. A user can join an existing organization or create a new organization.

Users can create their own organization using the EMPAIA Portal. A new organization must be approved by an EMPAIA administrator to finalize the creation. Only approved organizations show up on the public organization overview page of the portal. The user who created the organization by default is an organization administrator who can approve requests from other users to join the organization. Organizations can be assigned different organization types, including *AI Consumer* and *AI Vendor*. An *AI Consumer* organization can request the deployment of the EMPAIA platform services via the EMPAIA support. The deployment of an organization-specific software stack is not an automated process because many different integration scenarios exist (Section 3.4).

3.3. Use case 3: displaying available apps

Apps are displayed and described on a public marketplace page in the EMPAIA Portal. Each app is associated with an *AI Vendor* organization. In future versions of the platform, organizations will be able to upload and manage their own apps. For the initial release of the platform, apps are integrated into the marketplace by platform administrators. In the Portal, apps can be filtered by metadata tags that refer to the type of analysis performed with each app (e.g., tissue type, stain, and indication). These apps will also show up in the Workbench Client for pathologists to use in an examination. In the future, the apps listed in the Workbench Client will be limited to the ones previously licensed by the *AI Consumer* organization via the Portal.

3.4. Deployment

The platform architecture described in Section 2.2 supports different deployment scenarios, where services can be deployed on-premises, in a cloud environment, or in a mixture of both. Fig. 3 depicts four different scenarios where the Pathology Laboratories 1 to 3 can be pathology institutes of larger clinics/university hospitals or individual pathology laboratories. Depending on the size and resources of an institute, it might be possible to host everything on-premises, while others might want to use cloud resources.

Pathology Laboratory 1 (yellow) deploys all decentralized platform services on-premises. This even includes a compute cluster controlled by a Job Execution Service instance. Only user/service-to-service auth and the app marketplace are provided by global services. In this scenario, medical data is kept locally, even for the compute intensive processing of AI apps. It requires appropriate server resources to be available in an institution, but also provides the best protection of medical data, in comparison with the following deployments.

Pathology Laboratory 2 (blue) uses an App Service and Medical Data Services deployed on cloud servers outside the protected network of the institution. The medical data is transmitted to the cloud beforehand, such that the cloud compute cluster has fast access to the data at the time of processing. For data privacy reasons, the data is anonymized before uploading it to the cloud (R11). Since the Workbench Service and ID Mapper Service are deployed on premises, the mappings to local case and slide IDs that could be classified as identifying information never leave the originating institution. This scenario allows an institution to leverage remote storage and compute resources, while largely maintaining data privacy due to data anonymization. The applicability of this deployment strategy depends on regulatory and legal requirements.

Pathology Laboratory 3 (green) has a full cloud deployment, such that no local resources are used. This scenario is not feasible for clinical use because storing non-anonymized case mappings and slide mappings in a cloud is a privacy issue. On the other hand, this scenario is very useful for research purposes using already anonymized data. Also, the initial release of the platform, used for evaluation purposes only, follows this scenario. It allows the platform engineers to deploy the services in a controlled environment to simplify debugging and avoid the variety of local infrastructures.

Independent of the deployment scenario, platform services should not share compute resources with apps for several reasons: (1) Web services and apps differ greatly in their hardware requirements and load profile. Apps produce high bursts in CPU or GPU load but do not need storage capacities because they send their results to the Medical Data Service. (2) A high computational load produced by apps might have a negative effect on the availability or responsiveness of services. (3) The Job Execution Service can be extended to utilize multiple scheduling and orchestration frameworks in the future. For example, the Job Execution Service might schedule apps in a high-performance SLURM or Kubernetes cluster, while platform services (including the Job Execution Service itself) are orchestrated using docker-compose.

3.5. Integration

The EMPAIA platform architecture is designed in a way that allows parts of the platform to be interchangeable and implemented by different third-party software vendors. For example, as shown in Fig. 1, a third-party compute cluster provider can implement the EMPAIA Compute API in a compatible way to accept job execution requests from an existing Workbench Service implementation. Furthermore, the modular design of the EMPAIA platform reference implementation allows a gradual integration into clinical software and hardware infrastructures. While the reference implementation itself is not meant to be certified on its own as medical device software, it might be beneficial for a software vendor to reuse certain open source components: either to be used temporarily until a proprietary implementation is available, or to be permanently integrated into their software product and certified for clinical use.

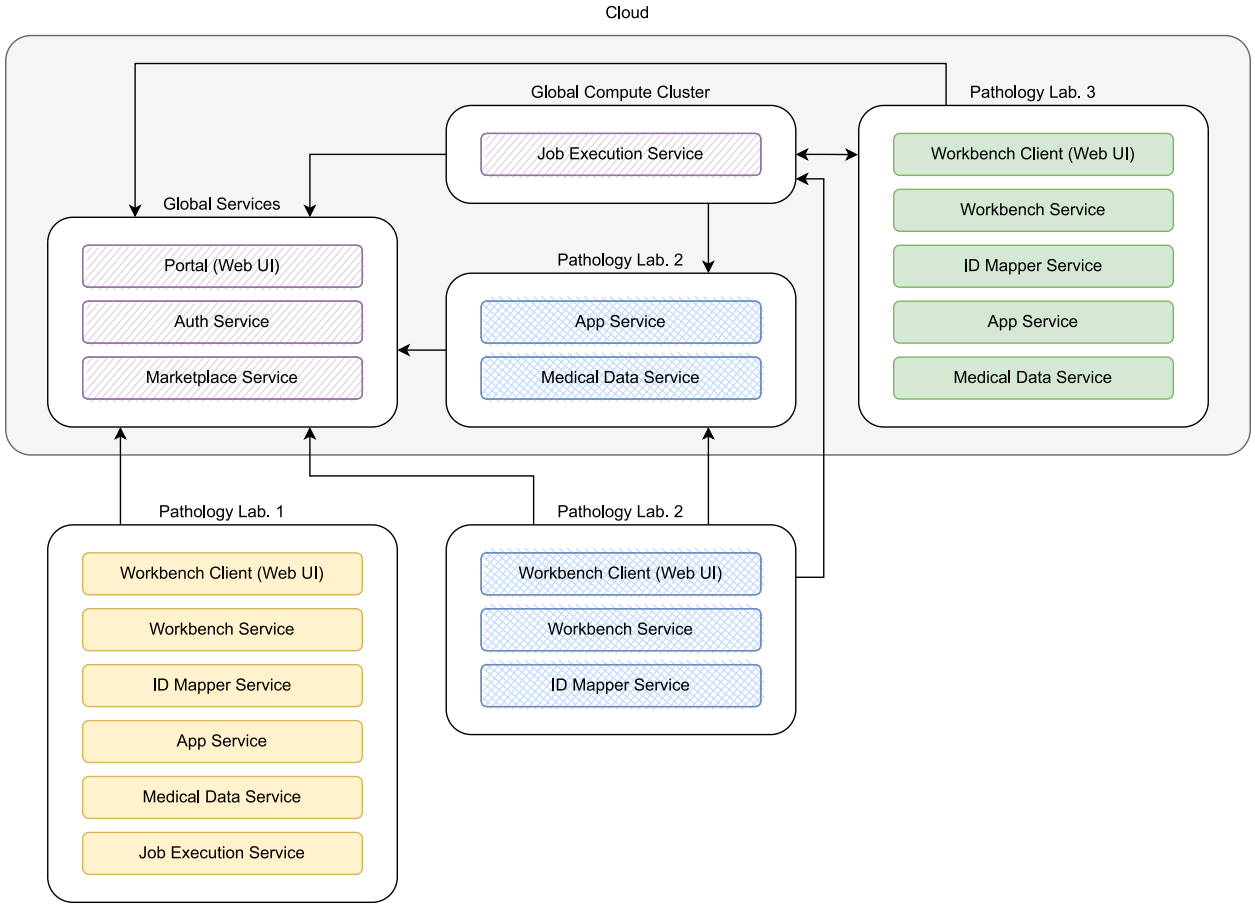

Fig. 3. Deployment: Decentralized platform services are deployed per pathology laboratory. Different scenarios are depicted for each laboratory, ranging from lab. 1 (yellow) with a complete local deployment, to lab. 3 (green) with a complete cloud deployment. Lab. 2 (blue) shows a mixed deployment. Arrows denote the direction of HTTP API requests.

In order to make the adoption of the platform reference implementation as easy as possible, the Medical Data Service has been implemented as an HTTP API layer (Section 2.2.2) in front of microservices that store and serve different data types. This separation is considered a technical detail of the reference implementation and is not part of the platform design. Namely these microservices are the Clinical Data Service, the WSI Service, the Annotation Service, the Job Service and the Examination Service. The Clinical Data Service stores case data and clinical metadata about the WSIs contained in a case. The WSI Service serves the actual image tiles of a WSI and provides technical metadata, such as the resolution and extent of each image layer. The Annotation Service stores geometric annotations and their pixel coordinates on a WSI, as well as primitive data types like numerical scores. All data types can be contained in data collections. Data elements and collections can have references to other elements and collections in order to create a semantic structure. Every element and collection used as a job input or created by a job as an output is being locked and becomes immutable. A flexible query system allows data to be fetched by data types, references, jobs, annotation coordinates (viewport) and more. The Job Service stores job objects, that link the ID of a certain app version and the IDs of input data to the IDs of the generated output data, which is crucial for traceability and reproducibility of processing results. The Examination Service stores examination objects, which link the IDs of apps used in a certain examination to the case ID and the corresponding job IDs.

As described in Section 3.4, the platform architecture allows services to be deployed on premises in clinical infrastructures (see pathology lab. 1 in Fig. 3). Based on this deployment scenario, Fig. 4 demonstrates a shallow integration approach with local systems, that can be used as a stepping stone to integrate the platform more deeply at a later point in time. A software vendor can, e.g., decide to use the Medical Data Service implementation, replace the Clinical Data Service and WSI Service with adapter implementations and use the Examination Service, Annotation Service, and Job Service as provided by the EMPAIA platform reference implementation. The Clinical Data Adapter can translate a request for fetching metadata of a clinical case into calls to a local APLIS, either using proprietary APIs or standardized HL7/FHIR APIs. The WSI Adapter can translate requests for fetching WSI image tiles to a proprietary IMS, to a VNA, or to a PACS. Since all adapter requests are translated just in time, there is no need to copy data from existing clinical software systems into the EMPAIA platform services. In the described scenario, the Examination Service, Annotation Service, and Job Service are not replaced by an adapter, because exact equivalents are most likely not yet present in the existing clinical systems. Fig. 4 also shows a third-party Pathology Workstation software that connects to the Workbench API and therefore replaces the Workbench Client reference implementation. The workstation provides a graphical user interface for pathologists to browse patient data, cases, and slides. Connecting to the Workbench API allows the software to run AI apps and display the processing results, leveraging the full potential of the platform.

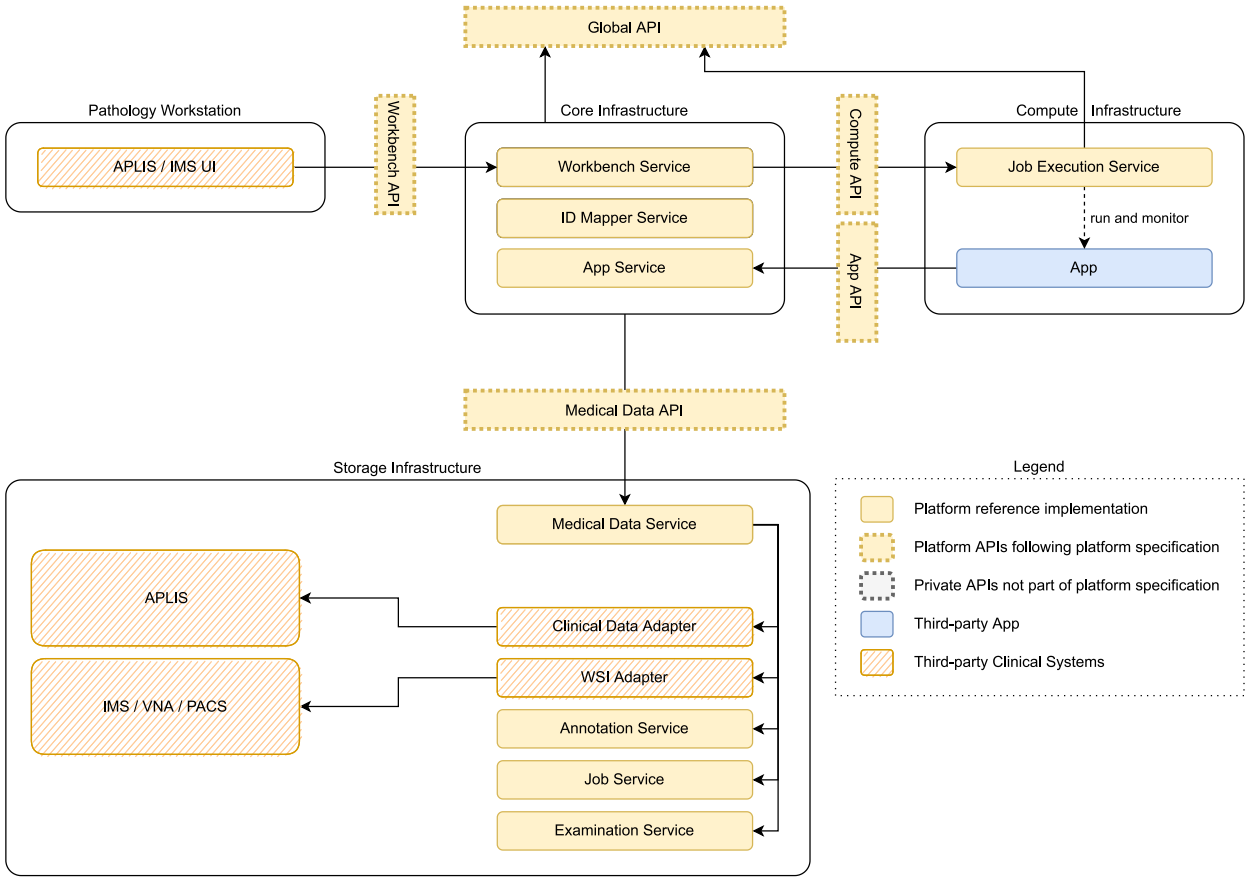

Fig. 4. Platform Integration: Third-party clinical storage systems are integrated via API adapters replacing Medical Data microservices. A third-party user interface (UI) for pathologists integrates with the Workbench API to access AI app functionality. Solid arrows denote the direction of HTTP API requests.

A deeper integration between the clinical systems and the EMPAIA platform implies that the storage systems directly implement the Medical Data API including support for examinations, annotations, and jobs, thereby eliminating the need for a separate API layer and adapters. Additionally, third-party vendors can replace the EMPAIA core services by implementing the Workbench and App APIs and can replace the Job Execution Service by implementing the Compute API. Due to the modular design, interoperability between the different parts of the architecture is ensured, even if these components are provided by different parties.

As described in Section 2.1.1, the slide scanners currently used in the routine of digital pathology produce proprietary image formats. Even though the DICOM standard for WSI data has been published more then a decade ago [22], the adoption by clinical hardware and software vendors has not made significant progress yet. For this reason the HTTP endpoints of the WSI Service have been designed as an abstraction over all existing image formats, including DICOM WSI. The WSI Service implements a plugin interface, that allows the usage of various WSI software libraries, like OpenSlide [36], to open and read image tiles from the corresponding formats. While the WSI Service is published under the MIT license, the plugin system also allows proprietary plugins to be implemented. For example, the isyntax image format can only be opened using the Philips Pathology SDK [53]. The SDK can be downloaded for free, but is not licensed under open source conditions and therefore cannot be redistributed. The WSI Services' isyntax plugin allows developers to download the SDK under the terms of Philips from the official website and build the plugin locally. The DICOM plugin of the WSI Service is based on wsidicom (Section 2.1.1), a library that provides a high abstraction over the DICOM standard. For example, DICOM image regions are usually addressed via coordinates in metric units, e.g., (x, y) position on the physical slide in millimeters. The wsidicom library on the other hand allows access to the image regions via pixel coordinates and therefore behaves very similar to OpenSlide and other WSI libraries. Implementing a WSI Service compatible plugin based on this library was trivial and serves as a proof of concept that other DICOM sources like DICOMweb can be integrated via a plugin or an adapter as described in Section 3.5.

3.6. Evaluation of data throughput performance when accessing image data

One important characteristic of both interactive and batch use of a digital pathology platform is data throughput performance. A modular architecture, as presented here, might imply technical overheads compared to a monolithic application when accessing image data or storing analysis results. In order to evaluate the data throughput performance of accessing image data, tests comparing different service setups and concurrency settings in our reference implementation have been conducted and presented in [29]. Key results are summarized here, more detailed information is provided in [29] and Appendix C.

In the described architecture, the Workbench Client requests image tiles for viewing purposes in the browser, it connects to the Workbench Service. The Workbench Service then forwards the request to the Medical Data Service and streams the response back to the client. In the same way, the App Service streams tile requests from the Medical Data Service when an App requests

them. For all connections from the client to the Workbench Service and from the Workbench Service to the Medical Data Service, OAuth2 authentication is enabled.

In the reference implementation, the Workbench Service use the HTTP client aiohttp (Section 2.1.2) for its requests. The Medical Data Service itself is only an API layer that connects to multiple microservices (Section 3.5). It also uses aiohttp to stream the requested image tiles from the WSI Service, therefore generating additional overhead. The latter is not inherently part of the platform architecture, but only part of the reference implementation, third-party implementations could choose a different approach. The experiment for evaluating data throughput performance when accessing image data involved requesting 500 image tiles. As a baseline, the WSI Service was timed directly when requesting the 500 image tiles concurrently, resulting in a total mean round trip time of 2.14 s. For the actual modular setting, requesting the tiles via the Workbench, Medical Data, and WSI Services and streaming the tiles back, the total mean round trip time was 2.50 s, an increase of 17%.

4. Discussion

The following sections summarize the presented achievement, show current limitations of the platform and discuss the future directions of the project.

4.1. Achievements

The implementation status of the EMPAIA platform covers the three main use cases presented in this publication (Section 1.4). The platform is designed to support flexible deployment scenarios (Section 3.4) that meet various regulatory and technical requirements. The EMPAIA Consortium collaborates with ten German and three international reference centers (hospitals and pathology laboratories). As of September 2022, reference centers have access to cloud-based deployments. These deployments allow pathologist to upload research data to gain a first impression of the platform. Five different third-party vendors have already integrated ten AI apps into the platform, that are now available to pathologists via the Workbench Client web UI. App developers are actively using the EATS (Section 2.5) to test their apps and provide valuable feedback concerning API design, features, and bugs via a dedicated mailing list.

For maximum compatibility, the EMPAIA platform is deliberately kept similar to existing free and commercial pathology software systems (Section 1.2). Unlike many existing solutions, the EMPAIA platform is not only intended for research applications, but also for clinical use. The EMPAIA platform is unique in that it offers open interfaces to both different AI app vendors and different pathology software system vendors. Another unique feature is that open source reference implementations of all components required for a functional, standalone deployment are provided.

Performance is an important aspect of the user experience. Delays in image and annotation rendering have a negative impact on the acceptance of new tools. The results from Section 3.6 show that it is possible to implement the services of the multi-layered API architecture, using the chosen technology stack, with merely 17% loss of data throughput performance under realistic load conditions when fetching image tiles. While there is potential for further improvements, the performance characteristics are acceptable for the current state of the platform reference implementation and the early adoption phase. Although a usability test has not yet been conducted, usability of the WSI viewer built into the Workbench Client does not seem to be negatively affected in practice. Furthermore, Section 3.4 demonstrates the flexibility of the architecture, such that the advantages outweigh the reduction in performance.

4.2. Limitations

On the one hand, the EMPAIA App Interface clearly specifies the integration of AI apps. On the other hand, the integration of the platform into clinical systems is much more diverse and a large variability of different systems has to be considered. Section 3.5 describes how a combination of open source components from the platform reference implementation and custom software adapters can enable a quick shallow integration with existing infrastructures. Although the integration approaches are currently being discussed with third-party clinical system vendors, a first practical implementation is still pending. Furthermore, the shallow integration approach is only a first step and a full integration using only certified medical device software must be achieved for unconditional routine usage of AI apps.

When defining APIs that are implemented and used by multiple parties, it is crucial to maintain compatibility even if new features are added to the platform and errors are being fixed. While each App uses a certain API version, systems might have to provide different API versions for different Apps to connect. A platform versioning strategy that allows APIs to evolve and features to be added, while not imposing a huge burden on system providers in terms of maintainability, is yet to be specified and discussed with all stakeholders. In addition, API versioning impacts the App validation and certification process that requires certain stability guarantees.

Section 3.5 describes the implementation of DICOM WSI file support. The WSI Service API endpoints are an abstraction layer to retrieve image data and associated metadata from DICOM and other proprietary WSI file formats, that are addressed as different data sources using plugins in the service backend. The decision to not directly serve DICOMweb endpoints is clearly justified by the heterogeneous landscape, but begs the question how the EMPAIA API can evolve, when DICOM finally becomes the norm. The project closely tracks the standardization efforts and must find suitable solutions to properly integrate DICOM into the EMPAIA APIs in the future.

The evaluation of data throughput performance presented here is limited to a reference implementation. Components of third-party implementations might have smaller or larger overhead due to streaming data through multiple API layers and might implement the Medical Data Service differently, necessitating dedicated performance measurements. Moreover, the evaluation is limited to accessing image data, further benchmarking will be needed: can image analysis apps store their output (e.g., large number of annotations) sufficiently fast and is the overall usage including running image analysis sufficiently fast, also under high load conditions? Such performance analyses depend on the image analysis apps and on the hardware setup (in particular network speed between the involved servers) at the respective site using the platform, and are beyond the scope of the present study.

4.3. Outlook

The development of the EMPAIA platform is still ongoing. Additional use cases, that will be covered in later versions of the platform, are the integration of a billing system, as well as an app validation and certification process (R1 and R11 in Section 1.4). Furthermore, the requirement to allow the automatic pre-processing of WSIs from a scan pipeline is not fulfilled yet (R10 in Section 1.4). Substantial gains in diagnostic efficiency can be expected if analyses are already finished or image quality issues have already been resolved when a pathologist first opens a case for diagnosis. A corresponding concept that allows apps to be used in different processing modes has been specified and improved based on feedback provided by third-party vendors.

The development process is currently ongoing and the feature will be released as part of the EATS (see Section 2.5) soon.

The Workbench API is currently being revised to support external UI modules (App UI) that ship with apps to enable custom interactions for app parameterization and result visualization. Six app vendors are currently adapting their apps and web UIs to integrate with this new API. All vendors will provide feedback and first prototypes until the end of the year. The results will be part of a future publication dedicated to the Workbench API and App UI concepts. Furthermore, on-premises deployments and integration approaches are being discussed with reference centers and clinical software vendors.

Satisfactory performance is only one aspect of an assessment whether the presented platform architecture is suitable for routine use and enables interoperability of digital pathology infrastructure systems and image analysis apps. Like other desirable properties such as flexibility, modularity, openness, and scalability, these aspects cannot be quantified in a meaningful way. Instead, usability studies are planned once infrastructure systems and image analysis apps have been integrated. These studies will involve different stakeholder groups (pathologists/end users, IT technicians at pathology labs, infrastructure system developers, image analysis app developers) in order to evaluate to what extent the EMPAIA platform lives up to required and desirable properties. Moreover, a comparison of user and developer experience to other platforms would be interesting, and should ideally be performed by an independent group. An important aspect for the acceptance of AI solutions in clinical use is the possibility for pathologists to get some insight into why an app has obtained a certain score for a given image [54]. Providing such explainability of algorithmic results is mostly a task of the individual app, but also requires support by the platform: it is already possible to display intermediate results, e.g., detected cells or tumor areas, by storing and retrieving geometric annotations (requirement R9, Section 1.4) via the Medical Data and Workbench APIs. Handling of suitable data representations for, e.g., saliency maps or more advanced explainability approaches will be included in the future.

Obtaining regulatory approval is an important step for medical software before it can be used in clinical routine [8]. App validation can only be successful if the underlying infrastructure is targeting the same quality standard. However, obtaining regulatory approval is more than a technical challenge. Therefore, the EMPAIA Consortium also aims to support vendors to follow best practices in medical device software development and documentation, with the initial validation and certification of new solutions, as well as the mandatory post-market surveillance for gathering clinical performance data and problem reports [55,56]. For this purpose, strict processes and supporting platform components are yet to be defined.

5. Conclusion

The EMPAIA Consortium aims to launch a sustainable ecosystem based on open and well-defined APIs connecting all relevant stakeholders in the digital pathology landscape. We here described the open and modular architecture of the EMPAIA platform and how its components can be used for previously identified use cases and how an integration with clinical infrastructures can be achieved. The first implementation already meets most of the identified requirements and will be evaluated at different reference centers, while development is ongoing. Once enough stakeholders use the platform, it can help accelerate the adoption of image processing and AI solutions in routine laboratory diagnostics.

CRediT authorship contribution statement

Christoph Jansen: Conceptualization, Methodology, Software, Visualization, Writing – original draft. **Björn Lindequist:** Conceptualization, Software, Writing – review & editing. **Klaus Strohmenger:** Conceptualization, Software, Writing – review & editing. **Daniel Romberg:** Conceptualization, Software, Writing – review & editing. **Tobias Küster:** Conceptualization, Software, Writing – review & editing. **Nick Weiss:** Conceptualization, Software, Writing – review & editing. **Michael Franz:** Conceptualization, Software, Writing – review & editing. **Lars Ole Schwen:** Writing – review & editing. **Theodore Evans:** Software, Writing – review & editing. **André Homeyer:** Conceptualization, Funding acquisition, Supervision, Writing – review & editing. **Norman Zerbe:** Conceptualization, Funding acquisition, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgment

This work was supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) via the EMPAIA project (grant numbers 01MK20002A, 01MK20002B, 01MK20002C, 01MK20002E, 01MK20002F).

Appendix A. Full list of empaia platform use cases

The original use cases contained in Table A.2 refer to the use cases presented in Section 1.4, where two original use cases have been joined to use case 2 (Section 1.4).

Appendix B. HTTP request sequence of a simple AI app

Fig. B.5 shows an exemplary HTTP request sequence of a very simple AI app, using the App Service in the platform reference implementation with the underlying Medical Data Service and its microservices. As soon as the app is initialized, it connects to the App API. In this case it fetches a Region Of Interest (ROI), which is a geometric annotation of type Rectangle, Polygon, or Circle. It is specified by a pathologist using drawing tools in a client-side WSI viewer. The ROI refers to the ID of the corresponding WSI, that is also part of the job's data input. The corresponding meta data (e.g., pixel resolution) are fetched by the app. This simple app detects the location of cells in the tissue and marks them in the form of Point annotations with pixel coordinates in the WSI's coordinate system. All annotations are stored in a collection that is created before the actual processing starts and new annotations are added to the collection with every processing iteration. The app only processes the image data located inside the ROI. In a loop, the app fetches all image tiles located in the ROI one by one. Each image tile is fetched via the API and processed by an algorithm to calculate cell locations. The resulting annotation objects are added to the collection using API requests. Note that the loop for fetching WSI tiles, processing the image data, and writing output annotations could be implemented using concurrent data IO while processing to reduce idle times. In the end, the job is finalized, such that all created data elements are implicitly locked to be immutable and the job status is updated. The app process then terminates itself. All mentioned data types are also briefly described in Section 2.2.2.

Table A.2
Original use cases defined by the EMPAIA Consortium and selected key use cases. Key use cases, that are not marked as work in progress (WIP), have been aggregated and are being presented in this publication.

Original use cases	User groups	Key use cases
Use App for diagnostics	Pathologist	Use case 1
Use App in research/education	Researcher, tutor, student	
Buy App	Pathologist, lab. manager/director	
Create/extend data set	Pathologist	
Conduct study (collect and evaluate data)	Researcher, pharmacist	
Train AI model	Researcher, software engineer	
Upload App to marketplace	Researcher, software engineer	WIP
Upload AppUI to marketplace	Researcher, software engineer	WIP
Update App in marketplace	Researcher, software engineer	
Update AppUI in marketplace	Researcher, software engineer	
Request clinical validation of an App	Researcher, software engineer	
Perform clinical validation of an App	EMPAIA validator	
Upload WSIs	Medical technical assistant, automation pipeline	
Register user account	All users	Use case 2
Register organization	Lab./company manager	Use case 2
Administration of users, data etc.	EMPAIA administrator	
Automatic preprocessing for diagnostics	Automatic pipeline	WIP
Creating an ad-hoc App with a generic AppUI	Researcher, tutor, student	
Publishing a research-grade App (for reference in publications)	Researcher	
Providing quality management services for clinical validation	EMPAIA validator, external certifier	
Beta testing and feedback from pathologist	Company manager, software engineer	
Billing and reimbursement	Company manager	
Display aggregated usage data	Company manager	WIP
Presentation of all Apps in the marketplace	All users	Use case 3
Testing App before buying	Researcher, pathologist	
Removing organization access/roles	Lab. manager	
Delete user account	All users	

Appendix C. Evaluation of data throughput performance when accessing image data

C.1. Test setup

In order to evaluate the overhead added by multiple API layers in the platform architecture, simple data throughput performance tests were executed. For this purpose, a Python script was implemented that requests 500 different image tiles from a single WSI (Mirax format) using the aiohttp library in a single process. Each tile is 256 × 256 pixels in size, using the JPEG format with a mean tile size of 20.91 KiB (min: 1.61 KiB, max: 43.11 KiB, total: 10.21 MiB). The time it takes to request all 500 tiles is measured 20 times (trials). The script is executed on a desktop computer and connects to the platform services deployed on a remote Virtual Machine (VM) with 16 CPU cores (Intel® Xeon® Gold 6154 CPU @ 3.00 GHz) and 64 GB RAM. The network throughput from the desktop to the VM was measured as 940 Mb/s (wget download of a 1GiB random binary file via HTTPS served by nginx). The number of processes assigned to the FastAPI services in total is always lower than the number of VM CPU cores during all experiments, such that the number of cores is not a limitation. All experiments use an nginx HTTPS reverse proxy in front of the API services. All services, except for the nginx reverse proxy, are deployed with docker-compose.

In **Experiment 1a**, the 500 requests are executed concurrently, only limited by aiohttp's default pool size of 100 connections. First, the requests are sent directly to the WSI Service for 20 trials, where the number of uvicorn workers (Section 2.1.2) is limited to 1. Then the trials are repeated with 2, 3 and 4 worker processes. In **Experiment 1b**, the requests are sent to the Medical Data Service with 1, 2, 3, and 4 workers. The Medical Data Service streams the data from the WSI Service running in the background using 4 workers. In **Experiment 1c**, the requests are sent to the Workbench Service with 1, 2, 3, and 4 workers. The Workbench Service streams the data from the Medical Data Service using 4 workers, which itself connects to the WSI Service with 4 workers. These experiments demonstrate how the platform architecture scales for parallel requests and how much overhead is added by the API layers. **Experiment 2** uses a setup of 4 Workbench Service, 4 Medical Data Service and 4 WSI Service workers, but enables the OAuth2 authentication for all connections from the client to the Workbench Service and from the Workbench Service to the Medical Data Service. **Experiment 3** uses a setup of 1 Workbench Service, 1 Medical Data Service and 1 WSI Service worker. In this case, a semaphore is used in aiohttp to limit the number of requests sent by the client's test script to one at a time. By omitting the concurrency, it is possible to gain a better understanding of the latency that is added by each API layer for individual requests. Results are reported in Appendix C.2.

C.2. Test results

Fig. C.6 shows the results of experiments 1a, 1b, and 1c, where the WSI Service, the Medical Data Service and the Workbench Service were targeted by the test script, respectively. As described in Appendix C.1, each service was run with 1, 2, 3, or 4 workers to demonstrate the scaling. As demonstrated, the WSI Service benefits heavily from a larger number of workers (w) to improve the trial duration for loading 500 tiles (WSI Service, $w = 1$: 6.71 ± 0.16 s (trial mean $\pm$ standard deviation); WSI Service, $w = 4$: 2.14 ± 0.10 s). This is explained by the fact that even though the FastAPI service uses asynchronous service handlers, the low-level image access functions of OpenSlide are blocking (synchronous) while they are waiting for storage IO operations. This could be improved in the future by developing a proper async wrapper for the OpenSlide plug-in. The Medical Data Service and Workbench Service experiments (1b and 1c) only show a slight performance increase with multiple workers because they already use the asynchronous http client library aiohttp that does not block during network IO operations, to stream the image tiles. The mean trial duration, 2.14 s, of targeting the WSI Service (experiment 1a) with 4 workers is now used as a baseline for the experiments 1b and 1c. Targeting the Medical Data Service with 4 workers adds an overhead of 6% (MDS, $w = 4$: 2.26 $\pm$ 0.08 s) over the WSI Service. Targeting the Workbench Service, with the

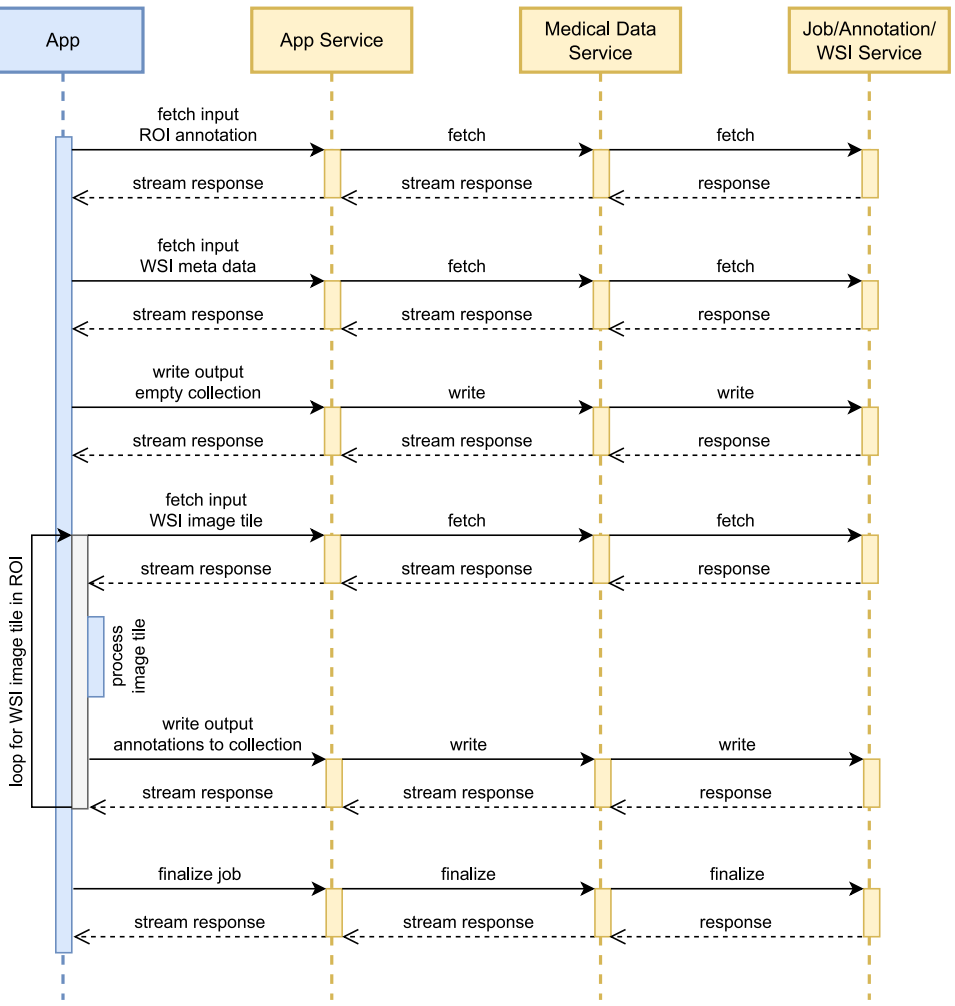

Fig. B.5. As soon as an app container is initialized, the app process is able to connect to the App Service using a token-based authentication. Job ID, job token and App API URL are provided to the app container in the form of environment variables.

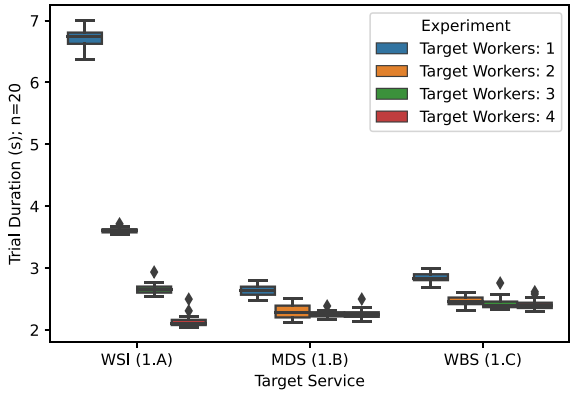

Fig. C.6. Performance characteristics of web request round-trip times when requesting image tiles, while targeting different API layers. Experiment 1a targets the WSI Service (WSI) directly with a differing number of worker processes enabled. Experiment 1b targets the Medical Data Service (MDS) as an abstraction layer over the WSI Service and experiment 1c targets the Workbench Service (WBS) as an additional abstraction layer over the Medical Data Service.

Medical Data Service as a proxy between Workbench Service and WSI Service, adds a total overhead of 12% (WBS, $w = 4$: 2.41 ± 0.08 s) over the WSI Service.

Experiment 2 (not shown in Fig. C.6) again uses a setup with 4 workers for each of the services. In contrast to experiment 1c, OAuth2 is enabled for client-to-service and service-to-service auth. The total overhead added by this setup is 17% (Workbench Service with OAuth2, $w = 4$: 2.50 ± 0.07 s) over the WSI Service with 4 workers.

Experiment 3 (not shown in Fig. C.6) is similar to the previous setups but does not allow concurrent client requests in the test script. When targeting the WSI Service with one request at a time, each request round trip on average takes 21.49 ms (WSI Service, $w = 1$: 21.49 ± 0.42 ms). Targeting the Medical Data Service in the same way adds an overhead of 12% (Medical Data Service, $w = 1$: 24.08 ± 0.53 ms). Targeting the Workbench Service in total adds an overhead of 24% (Workbench Service, $w = 1$: 26.74 ± 0.61 ms). Enabling OAuth2 in the platform adds a total overhead of 31% (Workbench Service with OAuth2, $w = 1$: 28.07 ± 1.05 ms) over the WSI Service.

References

[1] E. Abels, L. Pantanowitz, F. Aeffner, M.D. Zarella, J. van der Laak, M.M. Bui, V.N.P. Vemuri, A.V. Parwani, J. Gibbs, E. Agosto-Arroyo, A.H. Beck, C. Kozlowski, Computational pathology definitions, best practices, and recommendations for regulatory guidance: a white paper from the digital pathology association, J. Pathol. 249 (3) (2019) 286–294, http://dx.doi.org/10.1002/path.5331.

[2] B. Acs, M. Rantalainen, J. Hartman, Artificial intelligence as the next step towards precision pathology, J. Internal Med. (2020) http://dx.doi.org/10.1111/joim.13030.
[3] F. Aeffner, M.D. Zarella, N. Buchbinder, M.M. Bui, M.R. Goodman, D.J. Hartman, G.M. Lujan, M.A. Molani, A.V. Parwani, K. Lillard, O.C. Turner, V.N.P. Vemuri, A.G. Yuil-Valdes, D. Bowman, Introduction to digital image analysis in whole-slide imaging: a white paper from the digital pathology association, J. Pathol. Inform. 10 (2019) http://dx.doi.org/10.4103/jpi.jpi_82_18.
[4] A. Echle, N.T. Rindtorff, T.J. Brinker, T. Luedde, A.T. Pearson, J.N. Kather, Deep learning in cancer pathology: a new generation of clinical biomarkers, Br. J. Cancer 124 (2020) 686–691, http://dx.doi.org/10.1038/s41416-020-01122-x.
[5] A. Serag, A. Ion-Margineanu, H. Qureshi, R. McMillan, M.-J. Saint Martin, J. Diamond, P. O'Reilly, P. Hamilton, Translational AI and deep learning in diagnostic pathology, Front. Med. 6 (2019) http://dx.doi.org/10.3389/fmed.2019.00185.
[6] C.L. Srinidhi, O. Ciga, A.L. Martel, Deep neural network models for computational histopathology: A survey, Med. Image Anal. (2020) 101813, http://dx.doi.org/10.1016/j.media.2020.101813.
[7] H.R. Tizhoosh, L. Pantanowitz, Artificial intelligence and digital pathology: Challenges and opportunities, J. Pathol. Inform. 9 (2018) http://dx.doi.org/10.4103/jpi.jpi_53_18.
[8] A. Homeyer, J. Lotz, L.O. Schwen, N. Weiss, D. Romberg, H. Höfener, N. Zerbe, P. Hufnagl, Artificial intelligence in pathology: From prototype to product, J. Pathol. Inform. 12 (13) (2021) 1–13, http://dx.doi.org/10.4103/jpi.jpi_84_20.
[9] K. Bera, K.A. Schalper, D.L. Rimm, V. Velcheti, A. Madabhushi, Artificial intelligence in digital pathology—new tools for diagnosis and precision oncology, Nature Rev. Clin. Cncol. 16 (11) (2019) 703–715, http://dx.doi.org/10.1038/s41571-019-0252-y.
[10] D.J. Gross, W.S. Black-Schaffer, R.D. Hoffman, D.S. Karcher, E. Lopez Estrada, S.J. Robboy, M.B. Cohen, The state of the job market for pathologists: Evidence from the college of American pathologists practice leader survey, Arch. Pathol. Lab. Med. 144 (4) (2020) 420–426, http://dx.doi.org/10.5858/arpa.2019-0356-cp.
[11] S.T.C. Wong, Is pathology prepared for the adoption of artificial intelligence? Cancer Cytopathol. 126 (6) (2018) http://dx.doi.org/10.1002/cncy.21994.
[12] Philips, Philips IntelliSense Pathology Solutions, 2022, https://www.philips.de/healthcare/resources/landing/philips-intellisite-pathology-solution, accessed: 2022-06-21.
[13] Smart in Media AG, pathoZoom digitalLab, 2022, https://www.smartinmedia.com/pathozoom-digital-lab/, accessed: 2022-09-25.
[14] VMscope, Digital Slide Suite, 2022, https://virtuelle-mikroskopie.de/dss/mobilesuite.aspx, accessed: 2022-09-25.
[15] Carl Zeiss AG, Zeiss Zen, 2022, https://www.zeiss.com/, accessed: 2022-06-21.
[16] Carl Zeiss AG, Apeer, 2022, https://www.apeer.com/, accessed: 2022-06-21.
[17] Indica Labs, HALO, 2022, https://indicalab.com/halo/, accessed: 2022-06-21.
[18] Indica Labs, HALO AI, 2022, https://indicalab.com/halo-ai/, accessed: 2022-06-21.
[19] Sectra, Sectra Digital Pathology Solutions, 2022, https://medical.sectra.com/, accessed: 2022-06-21.
[20] G. Vincke, et al., Cytomine, 2022, https://cytomine.com/, accessed: 2022-06-21.
[21] M. Herrmann, D. Clunie, A. Fedorov, S. Doyle, S. Pieper, V. Klepeis, L. Le, G. Mutter, D. Milstone, T. Schultz, R. Kikinis, G. Kotecha, D. Hwang, K. Andriole, A. Iafrate, J. Brink, G. Boland, K. Dreyer, M. Michalski, J. Golden, D. Louis, J. Lennerz, Implementing the DICOM standard for digital pathology, J. Pathol. Inform. 9 (37) (2018) http://dx.doi.org/10.4103/jpi.jpi_42_18.
[22] R. Singh, L. Chubb, L. Pantanowitz, A. Parwani, Standardization in digital pathology: Supplement 145 of the DICOM standards, J. Pathol. Inform. 2 (23) (2011) http://dx.doi.org/10.4103/2153-3539.80719.
[23] R. Lebre, R. Jesus, P. Nunes, C. Costa, Collaborative framework for a whole-slide image viewer, in: 2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS), 2019, pp. 221–224, http://dx.doi.org/10.1109/CBMS.2019.00053.
[24] B.W. Genereaux, D.K. Dennison, K. Ho, R. Horn, E.L. Silver, K. O'Donnell, C.E. Kahn Jr., DICOMweb™: Background and application of the web standard for medical imaging, J. Digital Imaging 31 (2018) 321–326, http://dx.doi.org/10.1007/s10278-018-0073-z.
[25] Sectra, Sectra Amplifier Marketplace, 2022, https://medical.sectra.com/product/sectra-amplifier-marketplace/, accessed: 2022-06-21.
[26] Roche, Roche Digital Pathology Open Environment, 2022, https://diagnostics.roche.com/global/en/article-listing/roche-digital-pathology-open-environment.html, accessed: 2022-06-21.
[27] deepc, deepcOS, 2022, https://www.deepc.ai/, accessed: 2022-06-21.
[28] EMPAIA Consortium, EMPAIA, 2022, https://www.empaia.org/, accessed: 2022-09-28.
[29] C. Jansen, K. Strohmenger, D. Romberg, T. Küster, N. Weiss, B. Lindequist, M. Franz, A. Homeyer, N. Zerbe, The EMPAIA Platform: Vendor-neutral integration of AI applications into digital pathology infrastructures, in: 2022 22nd International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2022, http://dx.doi.org/10.1109/CCGrid54584.2022.00124.
[30] D. Romberg, K. Strohmenger, C. Jansen, T. Küster, N. Weiss, C. Geißler, T. Sołtysiński, M. Takla, P. Hufnagl, N. Zerbe, A. Homeyer, EMPAIA app interface: An open and vendor-neutral interface for AI applications in pathology, Comput. Methods Programs Biomed. 215 (2022) 106596, http://dx.doi.org/10.1016/j.cmpb.2021.106596.
[31] OpenAPI Initiative, OpenAPI Specification, 2022, https://spec.openapis.org/oas/latest, accessed: 2022-09-26.
[32] EMPAIA Consortium, EMPAIA Developer Portal, 2022, https://developer.empaia.org/, accessed: 2022-09-28.
[33] D. Hardt, The OAuth 2.0 Authorization Framework, 2012, http://dx.doi.org/10.17487/RFC6749, RFC 6749.
[34] L. Pantanowitz, P.N. Valenstein, A.J. Evans, K.J. Kaplan, J.D. Pfeifer, D.C. Wilbur, L.C. Collins, T.J. Colgan, Review of the current state of whole slide imaging in pathology, J. Pathol. Inform. 2 (36) (2011) http://dx.doi.org/10.4103/2153-3539.83746.
[35] P. Bankhead, M.B. Loughrey, J.A. Fernández, Y. Dombrowski, D.G. McArt, P.D. Dunne, S. McQuaid, R.T. Gray, L.J. Murray, H.G. Coleman, J.A. James, M. Salto-Tellez, P.W. Hamilton, QuPath: Open source software for digital pathology image analysis, Sci. Rep. 7 (1) (2017) 1–7, http://dx.doi.org/10.1038/s41598-017-17204-5.
[36] A. Goode, B. Gilbert, J. Harkes, D. Jukic, M. Satyanarayanan, OpenSlide: A vendor-neutral software foundation for digital pathology, J. Pathol. Inform. 4 (2013) http://dx.doi.org/10.4103/2153-3539.119005.
[37] D.A. Clunie, DICOM format and protocol standardization—A core requirement for digital pathology success, Toxicol. Pathol. 4 (49) (2020) http://dx.doi.org/10.1177/0192623320965893.
[38] IMI-BigPicture, wsidicom, 2022, https://github.com/imi-bigpicture/wsidicom, accessed: 2022-06-21.
[39] IMI-BigPicture, wsidicomizer, 2022, https://github.com/imi-bigpicture/wsidicomizer, accessed: 2022-06-21.
[40] Google Cloud Healthcare API, Using the Cloud Healthcare API for digital pathology, 2022, https://cloud.google.com/healthcare-api/docs/how-tos/dicom-digital-pathology, accessed: 2022-06-21.
[41] S. Ramírez, et al., FastAPI, 2022, https://fastapi.tiangolo.com/, accessed: 2022-06-21.
[42] A. Godwin, et al., Asynchronous Server Gateway Interface, 2022, https://asgi.readthedocs.io/, accessed: 2022-06-21.
[43] T. Christie, et al., Uvicorn, 2022, https://www.uvicorn.org/, accessed: 2022-06-21.
[44] A. Svetlov, et al., AioHttp, 2022, https://docs.aiohttp.org/, accessed: 2022-06-21.
[45] Open Container Initiative, OCI Image Format Specification, 2022, https://github.com/opencontainers/image-spec, accessed: 2022-06-21.
[46] Docker, Docker, 2022, https://docs.docker.com/, accessed: 2022-06-21.
[47] The Kubernetes Authors, Kubernetes, 2022, https://kubernetes.io/, accessed: 2022-06-28.
[48] H. Ed-Douibi, J.L. Cánovas Izquierdo, J. Cabot, OpenAPItoUML: a tool to generate UML models from OpenAPI definitions, in: International Conference on Web Engineering ICWE 2018, in: Lecture Notes in Computer Science, vol. 10845, Springer, 2018, pp. 487–491, http://dx.doi.org/10.1007/978-3-319-91662-0_41.
[49] L.F. Planella Gonzalez, et al., Angular OpenAPI 3 code generator, 2022, https://www.npmjs.com/package/ng-openapi-gen/, accessed: 2022-06-21.
[50] M. Jones, J. Bradley, N. Sakimura, JSON Web Token (JWT), 2015, http://dx.doi.org/10.17487/RFC7519, RFC 7519.
[51] EMPAIA Consortium, EMPAIA App Test Suite, 2022, https://gitlab.com/empaia/integration/empaia-app-test-suite, accessed: 2022-09-28.
[52] EMPAIA Consortium, EMPAIA Open Source Software Repositories, 2022, https://gitlab.com/empaia/, accessed: 2022-06-21.
[53] Philips, Philips Pathology SDK, 2022, https://www.usa.philips.com/healthcare/sites/pathology/about/sdk/, accessed: 2022-06-21.
[54] T. Evans, C.O. Retzlaff, C. Geißler, M. Kargl, M. Plass, H. Müller, T.-R. Kiehl, N. Zerbe, A. Holzinger, The explainability paradox: Challenges for xAI in digital pathology, Future Gener. Comput. Syst. 133 (2022) 281–296, http://dx.doi.org/10.1016/j.future.2022.03.009.
[55] Regulation (EU) 2017/746 of the European parliament and of the council of 5 april 2017 on in vitro diagnostic medical devices and repealing directive 98/79/EC and commission decision 2010/227/EU, 2017, pp. 176–332, OJ L 117. URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32017R0746.
[56] 21 C.F.R. § 822, 2022, https://www.ecfr.gov/current/title-21/chapter-I/subchapter-H/part-822.

Christoph Jansen acquired his master's degree in computer science with a specialization in visual computing at the HTW Berlin in 2015 and acquired his doctoral degree in medical sciences at Charité Berlin in 2020. He has worked in several research projects focusing on compute infrastructures and explainable machine learning methods in sleep medicine. Since 2020 he is the lead software architect in the EMPAIA project at the Institute of Pathology at Charité Berlin.

Björn Lindequist acquired his master's degree in Applied Computer Science at the HTW Berlin in 2014. Afterwards, he was a research assistant at the Institute of Pathology at Charité Berlin and HTW in the field of biobanking, digital image processing and classification. Since 2020, he is a research assistant at the Charité Berlin in the context of the EMPAIA project, with a focus on data management, platform and web service development.

Klaus Strohmenger acquired his master's degree in Applied Computer Science at the HTW Berlin in 2018. Afterwards, he was a research assistant at the HTW in the field of machine and deep learning. Since 2021, he is a research assistant at the Charité Berlin in the context of the EMPAIA project, with a focus on artificial intelligence for medical image processing and platform development.

Daniel Romberg received his diploma in computer science from the University of Lübeck, in 2010. He then joined Fraunhofer MEVIS as a research software engineer with a focus on real-time image processing and asynchronous operations for medical products. His current research interests include cloud computing, API usability and framework design. In the EMPAIA project, he focuses on the App interface, client-side anonymization of whole-slide images and conceptual development.

Tobias Küster has obtained his diploma in computer science in 2007 at TU Berlin, and his doctoral degree in 2017. He is currently heading the competence center "Agent Core Technologies" (ACT) at DAI-Labor, TU Berlin, and has worked in different research projects on multi-agent systems, process modeling, and the optimization of industrial processes and schedules. In the EMPAIA project, he is primarily working on services for managing and executing individual app executions.

Nick Weiss received his master's degree in Medical Engineering Science from the University of Lübeck, in 2014. Afterwards, he joined Fraunhofer MEVIS in Lübeck, Germany, as software engineer with focus on image registration and computational pathology. Within EMPAIA, he is primarily working on data management, platform and web service development.

Michael Franz has obtained his master's degree in Applied Computer Science at the HTW Berlin in 2020 and subsequently started working as a software developer at the Charité Berlin. The main focus of his research are medical image formats, image registration and 3D visualization. Within the EMPAIA project he maintains various medical data services and works on the integration of the EMPAIA marketplace.

Lars Ole Schwen received his diploma in Technomathematics from University of Duisburg-Essen in 2005, and his doctoral degree in Mathematics from University of Bonn in 2010. Since then, he has been a scientist at Fraunhofer MEVIS in Bremen, Germany, working in the fields of modeling and simulation in systems biology as well as data science and artificial intelligence in computational pathology. Within EMPAIA, he has mostly worked on data science topics.

Theodore Evans obtained his M.Phys. degree in Physics from the University of Manchester, with a focus on theoretical physics and information theory. He has a PGCE in Secondary Education from the University of Sussex and is a qualified science teacher. Since 2020 he has conducted research as part of EMPAIA into computer vision and explainable AI for the medical imaging domain at the Distributed AI Laboratory, TU Berlin.

André Homeyer is a principal scientist at Fraunhofer MEVIS, Bremen. He received a Ph.D. in computer science from the University of Bremen. Since 2008, he has been conducting research in the field of computational pathology. His research interests include applications of machine learning and data analytics in biomedicine, with a current focus on technology transfer.

Norman Zerbe holds a Computer Science degree from the University of Applied Sciences Berlin. He is Head of Digital Pathology Research at the Institute of Pathology at Charité. He was co-organizer of several International Scanner Contests that assessed and validated the performance and quality of WSI scanners. Norman is President of the European Society of Digital and Integrative Pathology (ESDIP) and chair of the Informatics, Digital Pathology and Biobanking WG of the German Society of Pathology (DGP). Moreover, he actively contributes to standardization and interoperability efforts within the DICOM WG-26 as well as in the IHE PaLM WG.