ДОДАТОК А

Апробація результатів роботи

Investigation of Architecture and Technology Stack for e-Archive System

Hennadii Falatiuk, Mariya Shirokopetleva, Zoia Dudar
Department of Software Engineering
Faculty of Computer Science
Kharkov National University of Radio Electronics
Kharkov, Ukraine
hennadii.falatiuk@nure.ua, marija.shirokopetleva@nure.ua, zoia.dudar@nure.ua

Abstract **– This work describes main concepts of architecture style and technologies selection for building distributed e-Archive system. The paper presents overall concepts, vocabulary, data models, responsibilities that an electronic archive system must fulfill, and a set of recommended functions to cover these responsibilities. The article describes the main information flows and functions. It outlines the theory of the OAIS model, illustrates an example of complex microservices architecture design and approaches to solve data consistency and application deployment challenges. The comparative analysis of software architecture styles was performed and, as result, the combination of Microservice and Event-Driven architectures was chosen as the most suitable architecture for building electronic archive system. It implies having microservices that communicate with each other via some message bus instead of direct calls. The technology stack chosen for that architecture implementation and application delivery is proven to be production-ready, has detailed documentation and large community support. The technology stack is Asp .Net Core 2.1 – framework, Rabbit MQ, Apache Tika, Elasticsearch, Mongo DB, MS SQL Server, Azure Blob Storage, Event Store, Signal R, Docker, Kubernetes.**

Keywords – microservices, e-archive, OAIS, software architecture, containers, orchestration, eventual consistency, software design patterns

I. Introduction

For the last decades, dozens of thousands of bits of digital information are being created every day. People film movies, take photos, record songs, write texts and create documents and all these data must be stored in some way. When it comes for big organizations with hundreds of files created for millions of users it is not an easy task to organize the storage of that digital information in an efficient form that allows quick access to required pieces. This problem is most critical for government organizations where according to law all the cases of the citizens must be stored for a long period of time. But storing such amount of information in the system that produces it in most cases makes the system much slowly and eventually kills its productivity at all. And all this is for nothing since most of the information is not access very often and is stored mostly for history or in case of something.

To deal with these challenges the e-archive systems come for the rescue. Such systems are intended to shift the weight of storing huge seldom accessible data from the shoulders of the systems- producers. The e-archive system must be well constructed to handle the expected amount of data by design. And this is not just about code, the first thing that must be properly selected is the conceptual model of the system because the only way to build something good is to understand what you are building.

II. Open Archival Information System Model

The process of e-archiving is not a new challenge. A long time ago the NASA's CCSDS (Consultative Committee for Space Data Systems) has faced the problem of proper organization and long-term preservation of the information generated by decades of space missions and observations. As a result, the OAIS model was created which then become an ISO standard (ISO 14721:2012).

The major information streams and functions are shown on the Fig. 1.

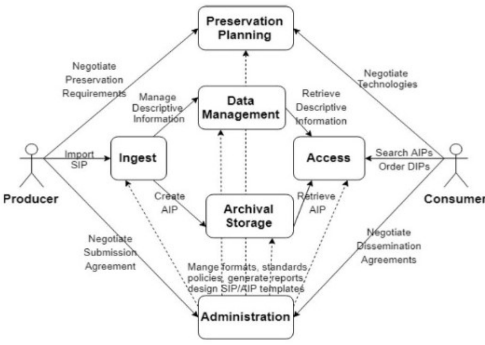

Preservation Planning
Negotiate Preservation Requirements
Manage Descriptive Information
Data Management
Retrieve Descriptive Information
Negotiate Technologies
Import SIP
Ingest
Access
Search AIPs
Order DIPs
Producer
Consumer
Create AIP
Archival Storage
Retrieve AIP
Negotiate Submission Agreement
Negotiate Dissemination Agreements
Mange formats, standards, policies, generate reports, design SIP/AIP templates
Administration

Fig. 1. OAIS Functional Model

OAIS stands for Open Archival Information System, it is a framework that provides a broad understanding of actions necessary for the long-term preservation and accessibility of data. Unlike many standards, OAIS does not specify any implementation, programming API, data format, or protocol. Instead, it provides a conceptual model that consists of four basic parts [1]:

- Vocabulary that defines common terms, operations, services, and information structures of the e-archive system.
- Data model of the information that is sent to, managed inside and exported from the system. This model implies the following package types: Submission Information Package (SIP) for import, Archival Information Package (AIP) for

preservation and Dissemination Information Packages (DIP) for export.

- Mandatory responsibilities of the e-archive. They imply negotiation with producers and consumers for the completeness and intelligibility of information being archived and following well-defined and well-documented procedures for obtaining, preserving, authenticating, and providing this information.
- Recommended set of functions to fulfill the responsibilities of the e-archive system. These functions are grouped into six functional modules: ingest, data management, archival storage, access, administration, and preservation planning.

So, the OAIS model gave us the overall concepts, vocabulary, data models, responsibilities the e-archive system must fulfill and set of recommended functions to cover these responsibilities. Now, the actual challenge is to build the software, design its architecture, select technologies, data formats, communication protocols, design internal and external API and create the deployment and maintenance strategy, so that all functional and non-functional requirements are met.

III. Architecture Considerations

Architecture is the foundation stone of the application, that is the first thing the system construction must start with. Given functional and non-functional requirements the main goal of software architecture is to define which components the system should consist of, how those components are going to communicate with each other and how they must be deployed in order to fulfill these requirements. That is the hardest and the most important part of the application construction because all the mistakes made on this stage are the most expensive to fix in future, so you always need to put the right amount of effort on this task.

A. Choosing the architecture style

The architecture of software application is usually complex and comprehensive, especially when it comes to web-oriented systems with the rich business domain. That`s why it is impossible to make a good solution in the rush from scratch. So, the first thing to choose when designing the application is the architecture style, which defines a high-level paradigm and principles. Experts from Microsoft distinguish six different architecture styles [3]:

- N-tier – a traditional approach that usually implies dividing an application into logical layers (i.e. presentation, business logic, and data access) and placing these layers on N tiers (nodes). Layers have vertical orientation and each layer can call only the layer below.
- Web-Queue-Worker – this architecture implies thin web layer which main purpose is to validate user requests and issue commands for long-running CPU-intensive tasks to queue and a worker that listens to this queue for incoming tasks and processes them asynchronously in the background. Such architecture is suitable for domains with resource intensive long-running tasks.
- Microservices – in that case, application is composed of many small, independent services, each of them implements the single business capability. Microservices are loosely coupled and communicate with each other via strictly defined API. This architecture is suitable for complex domains.
- CQRS (Command Query Responsibility Segregation) – approach that implies different read and write data models. This architecture allows independently scale read and write workloads.
- Event-driven architecture – implies the publish-subscribe communication model, where producers publish events and consumers subscribe to them. Both producers and consumers are independent of each other.
- Big Data and Big Compute – architecture style that deals with large data volumes or high-performance computing by means of splitting data into chunks and performing parallel processing of them.

Considering the e-archive domain, we already have vertically decomposed functions (i.e. import, data management, storage, administration, search, export and preservation planning) and models (i.e. SIP, AIP, DIP). Moreover, the workloads of each functional capability may differ dramatically, which leads to necessity of independent scaling of each function. The e-archive system must be able to handle large data volumes by design, which also implies need to horizontal scaling. As any software, the e-archive system cannot be built whole at once, so it is going to be constructed incrementally, which implies frequent releases. Also due to loosely coupled functions the updates in one function might not affect others, so to minimize impact and downtime of the update process it should be possible to independently ship updates of different parts of the system. The fulfillment of the e-archive system requirements by different architecture styles is shown on.

TABLE I. E-ARCHIVE SYSTEM REQUIREMENTS FULFILLMENT BY DIFFERENT ARCHITECTURE STYLES

E-Archive Requirement	**Architecture styles**					
	N-tier	***Web-Queue-Worker***	***Microservices***	***CQRS***	***Event-Driven***	***Big Data***
Independent functional scaling	-	+/-	+	-	+	+/-
Scaling out	+	+	+	+	+	+
Fault isolation	-	+/-	+	+/-	+	+/-
Gracefull degradation	-	+/-	-	+	-	-
Modular updates	-	-	+	-	+	+
Ease of modification	-	-	+	-	+	-
Loose coupling	-	+/-	+	-	+	+/-

Considering the results of the architecture styles comparison, the most suitable style is the combination of the Microservices and Event Driven architectures, which implies having microservices that communicate with each other via some message bus instead of direct calls.

Besides, this architecture style naturally fits the SaaS (Software as a service) delivery model which is the primary way of how the e-archive system is going to be shipped to end-users. This delivery model shifts imposes all the deployment and maintenance activities on the company that develops software, while the end-user just buys a subscription which provides access directly to application.

B. Defining the architecture components and communication scheme

Having chosen the high-level system architecture, the next step is to define components, set of functions they implement, and the data flow between them.

As mentioned earlier there are three types of information packages: SIP, AIP, DIP. Each type represents data format on certain stage of the archiving process as well as translations between them. Basically, any information package consists of the following parts [2]:

1) Content Information (CI) – the set of information that is the original target of preservation by the e-archive.

2) Preservation Descriptive Information (PDI) – the set of information that maintains confidence, authenticity, and context of the CI over an indefinite period (i.e. reference, context, provenance, fixity and access rights information).

3) Descriptive Information (DI) – the set of information that describes the CI and is used to locate, analyze, retrieve, or order information from the e-archive system.

The difference between those package types is in the completeness and format of the information. Thus, SIP typically contains CI and part of PDI, the AIP contains the CI and must contain full PDI and DI, the DIP may contain parts of CI, DI and PDI. Since CI, DI and PDI are abstract terms, it is needed to strictly define their formats and representation in the current system. So, in the AIP package the CI is going to be represented as collection of files of any supported format, DI is an XML file that corresponds to the target archive`s schema and PDI is an XML file with XSD schema defined by the system. Considering these package types and functions that the e-archive must provide, the set of components the system will consist of are present on the Fig. 2:

1) Import Svc – service that is responsible for preprocessing of data supplied for import to the archive (SIPs). That is the first stage of AIP creation process which performs data transformation to the format acceptable by the target archive (e.g. split one SIP into several AIPs, split / transform DI, convert CI to another file format, etc.)

2) Archive Svc – service that is responsible for actual storing of archiving data. The top-level entities of this service are "Archives", they are analogs of an actual archive from the real world. Archive contains information packages in form of AIPs. Each archive has its schema in XSD format which defines the structure of AIP`s DI. All the packages that are being imported to the system are stored in certain archive. The process of placing or updating AIP into archive includes the following stages that are going to be carried out by that service:

a) Data Validation. Validation of supplied AIP`s DI according to the schema of the archive this AIP is being imported. Validation of CI (i.e. file formats, file sizes, content, etc.)

b) Checksum calculation. Calculate fixity information for CI and DI.

c) PDI maintenance. Update of related sections in PDI when CI and DI are changed.

d) DI maintenance. According to configured AIP template, if there is a relation between the CI and DI update DI when CI changes.

Itself the Archiving service provides set of API for adding AIP, retrieving AIP info, retrieving and updating individual parts of AIP.

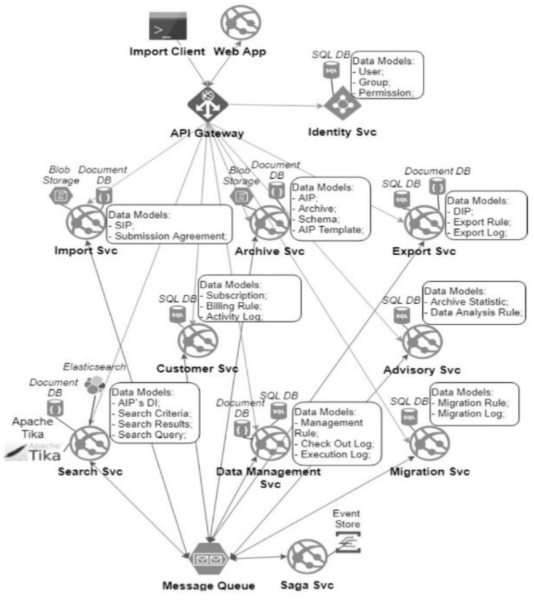


Fig. 2. Components diagram of the e-archive system architecture

3) Search Svc – service that is responsible for search functionality in the e-archive system. It indexes the DI of each AIP, also, if configured, performs text extraction from CI and indexes it as well. Besides this service contains all the configuration for search query and results appearance.

4) Data Management Svc – service that is responsible for manual and automatic management of AIPs. This services provides such functionality as configuration and launch of rules for AIPs` DI, PDI and CI modification, checking out AIPs for editing, visualization of AIP`s parts for manual editing.

5) Export Svc – service that is responsible for export of packages from the e-archive system. This services transforms AIPs to DIPs of the format that is acceptable by the export target.

6) Migration Svc – service that is responsible for data migration between archives. This service performs preservation planning function which implies safe migration of data from archive with stale storage to archive with the modern storage.

7) Identity Svc – service that is responsible for user management. This service provides authentication and authorization, users and groups configuration and permissions management. The best way to do authentication / authorization in the microservice architecture is token based authentication mechanism [2]. This approach implies separate microservice that stores all the account info, so can authenticate user. When user has proven his identity (via password or by other means) this service issues the authorization token with all the information about user rights in the system. Then this token is encrypted and sent to the client. When user performs any restricted operation in the system, it sends this token along with request to the respective service. Each microservice in the system is able to decrypt token and use information in it to verify user`s rights to perform requested action. This authentication scheme has following advantages:

a) Centralized identity management. Users, groups and permissions are controlled by single microservice.

b) No centralized failure for authorization. When user is authenticated and has received the token it can access any of the microservices in the system until the token expires even if the Identity Service is down.

c) No network latency for authorization. Token contains all the information about user permissions, so the requested microservice does not have to call the Identity Service for any additional details.

d) Strong security. Token is encrypted with strong cryptographic algorithm which makes it impossible to forge it.

The format of the token used for the Identity Svc is JSON Web Token (JWT) with HMACSHA256 signing algorithm.

8) API Gateway – this is a deployment component that provides a single, unified API entry point across all the internal microservices` APIs.

9) Web App – this is the first user interactive part of the system which provides access to all the functionality shipped by microservices APIs.

10) Import Client – this is the second user interactive part of the system which is a desktop application that performs import of packages into the system.

11) Saga Svc – this service is responsible for maintaining consistency of long running operation in the system.

12) Message Queue – this is the message bus that establishes asynchronous communication model between microservices.

C. Choosing the consistency model

Consistency is an important part of the e-archive system, since it is intended to preserve information for an indefinite period. Maintaining consistency in the distributed system is a tricky task.

According to the CAP theorem it is impossible to simultaneously maintain more than two out of three guarantees: Consistency, Availability and Partition tolerance. Considering this "2 of 3" limitation the e-archive system will stick to Availability and Partition tolerance combination (AP). Since system has transactional operations which span multiple durable resources, we need to implement Eventual Consistency across these resources. The overall concept of the Eventual Consistency is shown on Fig. 3 by example of import process:

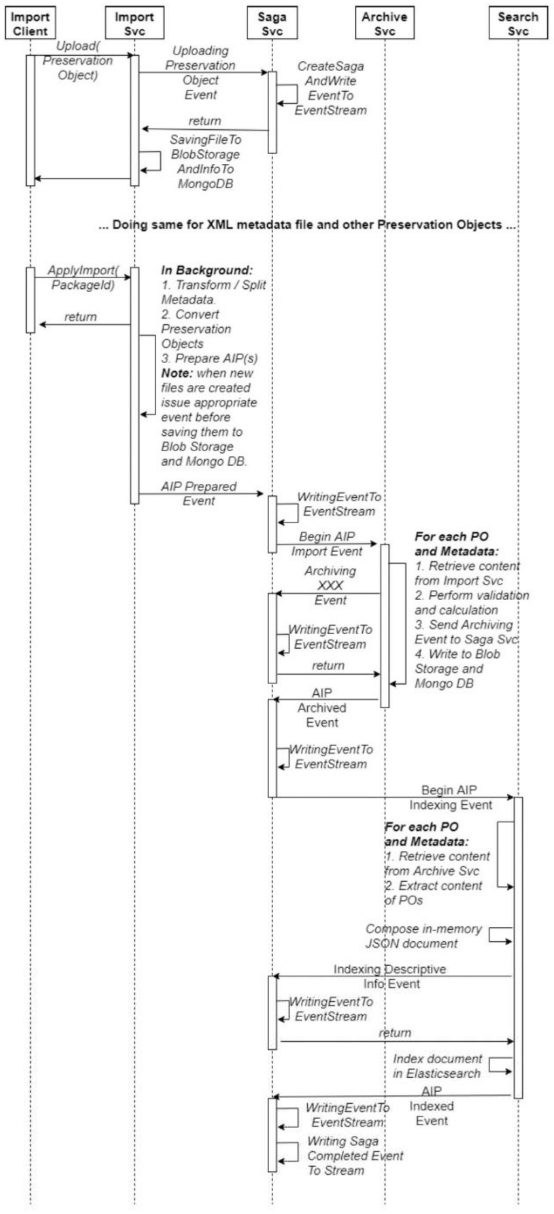

Import Client
Import Svc
Saga Svc
Archive Svc
Search Svc
Upload(Preservation Object)
Uploading Preservation Object Event
CreateSaga AndWrite EventTo EventStream
return
SavingFileTo BlobStorage AndInfoTo MongoDB
... Doing same for XML metadata file and other Preservation Objects ...
ApplyImport(PackageId)
return
In Background: 1. Transform / Split Metadata. 2. Convert Preservation Objects 3. Prepare AIP(s) Note: when new files are created issue appropriate event before saving them to Blob Storage and Mongo DB.
AIP Prepared Event
WritingEventTo EventStream
Begin AIP Import Event
For each PO and Metadata: 1. Retrieve content from Import Svc 2. Perform validation and calculation 3. Send Archiving Event to Saga Svc 4. Write to Blob Storage and Mongo DB
Archiving XXX Event
WritingEventTo EventStream
return
AIP Archived Event
WritingEventTo EventStream
Begin AIP Indexing Event
For each PO and Metadata: 1. Retrieve content from Archive Svc 2. Extract content of POs
Compose in-memory JSON document
Indexing Descriptive Info Event
WritingEventTo EventStream
return
Index document in Elasticsearch
AIP Indexed Event
WritingEventTo EventStream
Writing Saga Completed Event To Stream

Fig. 3. Sequence diagram of eventual consistency implementation during the AIP import process

To achieve this goal the combination of the following design patterns is used [3]:

- *Saga* – concept that represents a sequence of local transactions in different microservices. To coordinate sagas the orchestration way was chosen with separate microservice responsible for it.
- *Transaction Log Tailing* – approach of updating data store that implies writing an intent of update action to the separate persistent log file and after this log file is saved to disc perform the update of data source.
- *Compensating Transaction* – design pattern that implies that command / handler that updates data source has compensating action that can undo changes to data source that were made by the intended action.
- *Idempotent Event Handlers* – approach that allows to execute same handler with same arguments multiple times and each time result will be the same. Thus, calling event handler to added item with same id to database must neither add multiple records nor throw an exception, but create record only once.
- *Event Sourcing* – design pattern from Domain Driven Design that implies concept of storing not the state of certain entity, but rather sequence of events that lead to that state. This provides such advantage as optimistic concurrency (there is no need to lock records, since each event contains expected version of the object, so if the last persisted event`s version differs from the incoming event, then concurrent access is detected and error is raised and data is not changed).

Before each operation that is going to update data store that is part of overall consistency model of the e-archive an event with all the information that describes action intent is sent to Saga Svc. And after the event is written to disc (Event Store) the actual action is performed. Events are delivered via Rabbit MQ. Each event handler that processes received event is idempotent and implements compensative transaction pattern. The event stream representation of the import saga is shown on the Fig. 4.

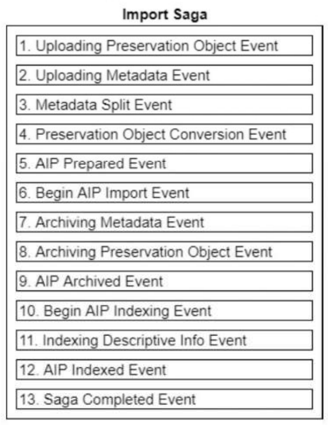

Import Saga
1. Uploading Preservation Object Event
2. Uploading Metadata Event
3. Metadata Split Event
4. Preservation Object Conversion Event
5. AIP Prepared Event
6. Begin AIP Import Event
7. Archiving Metadata Event
8. Archiving Preservation Object Event
9. AIP Archived Event
10. Begin AIP Indexing Event
11. Indexing Descriptive Info Event
12. AIP Indexed Event
13. Saga Completed Event

Fig. 4. Event stream representation of import saga

Since all events are sequentially written to event stream they can be easily traversed forward and backward. So, in case of failure at any step of the distributed transaction the compensation sequence is launched which implies reading of saved events in reverse order and redelivering of these messages to respective handlers with compensate flag set to true.

IV. Technology Stack Selection

Considering the microservice architecture the following technology stack is considered the most suitable for its implementation:

- Asp .Net Core 2.1 – framework used for implementing functional microservices (Import, Archive, Search, Data Management, Export, Migration, Identity, Saga and Web App). This framework allows to write web applications that are possible to host on Linux and Windows platforms.
- Rabbit MQ – message broker that enables asynchronous communication between functional microservices. This message queue supports multiple messaging protocols, flexible message routing, messages persistence, has convenient API and official client library for the .Net programming language. Moreover, it supports horizontal scaling, and can be deployed as a cluster.
- Apache Tika – text extraction toolkit that is used for extraction of text from AIP`s CI for further indexing in the search engine. This is an open source solution with convenient HTTP API. Since this is a stateless service it can be easily scaled horizontally.
- Elasticsearch – search engine that is used for indexing of AIP`s DI and text extracted from CI. It supports flexible queries, aggregations, full-text queries, suggestions and other features valuable for e-archive system. Elasticsearch supports horizontal scaling and can be easily deployed as a cluster.
- Mongo DB – is an open source document-oriented database. It is used for storing complex document-like entries in different microservices (e.g. AIP, SIP, DIP, Submission Agreement, Search Query, Management Rule, etc.). This database provides ACID data integrity guarantees and horizontal scaling, which is beneficial for e-archive system.
- MS SQL Server – traditional SQL database in .Net world. It is used for storing user management data and records regarding long-running processing in several microservices (i.e. Check Out Log, Execution Log, Migration Log, etc.).
- Azure Blob Storage – cloud storage for SIP`s files and AIP`s CI. This storage supports replication which is necessary for e-archive system.
- Event Store – stream database for event sourcing that is used for storing sagas of transactional actions in the system. This database can be scaled horizontally and easily deployed in cluster.

- Signal R is a .Net library that simplifies adding ability to push content from server to connected clients as it becomes available rather than waiting for clients to request data from server. This technology supports four types of transport (Web Sockets, Server-Sent Events, Forever Frame, Ajax Long Polling) and fully encapsulates the fallback mechanism for transport selection, which automatically enables support of any browser.

The communication between Web App and microservices is performed via API Gateway. The API Gateway allows to monitor and limit the API calls to microservices. Since communication between Web App and other microservices is synchronous, the simple HTTP REST approach was chosen. For communication between microservices the asynchronous messaging approach was chosen using AMQP 0-9-1. This decision was made to decouple microservices from each other and due to the long-running time-consuming nature of the operations that these microservices take part in.

The combination of those technologies allows to build e-archive system that fully implements all the required functions, can handle large data volumes, meet the performance, consistency and recoverability requirements.

V. Application deployment

At that point the overall system consists of 11 functional microservices and about 13 third-party components instances used by those microservices (since the database per microservice pattern was chosen during system design). And this is the minimal deployment configuration with single instance of each service. So, when it comes to scaling out the number of services will increase to several tens. Manual deploying of such system and its configuration will take enormous amount of time, that is just unproductive. Moreover, the ideal option is to deploy each component on a separate node for the sake of fault isolation. But it would be an extreme waste of resources if single node runs single application.

A. Simplifying product deployment with Docker

To tackle the complexity of microservice app deployment the containerization comes for the rescue. Application containerization is an OS-level virtualization method used to deploy and run distributed applications without launching an entire virtual machine (VM) for each app [4]. Containers are more lightweight than VMs, because the container engine virtualizes the OS for each container, so all the containers share the same OS files, while the hypervisor virtualizes the hardware which leads to having a full copy of OS for each VM. Application and all the dependencies (i.e. runtime components, environment variables, dependent libraries, system tools) are packed into an image, which represents a layered set of files with copy on write approach. Containers provide a lot of benefits for application deployment:

1) Isolation – application running in container is fully isolated, which means other applications on the host machine do not affect it. Also the container contains all the files and dependencies, so when it is removed it does not leave any artifacts related to this application on the host OS.

2) Standardization and portability – containers provide stadardization of the execution environment and abstract away the specifics of the underlying operating system and hardware. This allows images run uniformly on any host no matter where they were created.

3) Fine-grained control – each container gets its own set of system resources with defined quotas, which cannot be exceeded.

4) Ease of deployment and configuration – containers are not needed to be installed or specifically configured, they are just activated by single command line. All the configuration is already coded into an image.

5) Security – strong isolation and complete control over traffic flow and management of the containerized application minimizes its vulnerable surface.

The most common app containerization technology is Docker which became an industry standard for containers. To containerize an application, it is needed to create an image using the Dockerfile. In microservice application each service must has its own image. After the image is created and pushed to the container registry, the container can be spun on any host running the Docker Engine by means of single command. Of course, manual launch of dozens of images is much easier than manual installation of services, but still far from production. To simplify deployment of all the microservices at once the Docker Compose file comes for the rescue. This is a file in YAML format that allows declaratively specify the deployment configuration of multiple images, so the respective containers are spun up altogether as a single application. Docker Compose file can be used with Compose CLI which deploys containers on a single node or with Stack Deploy command of Docker CLI that can work with cluster of nodes.

So, now we have convenient and easy to use tool for deploying the whole microservice application by literally executing single command line. The next challenge is to maintain and monitor the running app.

B. Orchestration of containerized application

Orchestration is an important part of containerized application deployment and maintenance. Having a lot of containers that execute their own independent tasks and communicates with each other is great up until the point when they arbitrarily go down. With a traditional monolithic app, it is not a big deal, when application stops working it is enough to just connect to that single node it is deployed on and fix the issue. But when it comes to containers, this is far from easy to manually handle such faults. For that bare reason the container orchestration tools were constructed. Such tools aim to automate container management and provide a framework not only for defining initial containers deployment but also for managing multiple containers as a single application. Container orchestration tools provide such important functionality for microservice application as:

- *Self-healing* – automatic spinning of container instances up to required count when existing ones go down.
- *Auto-scaling* – automatic creation / removal of the containers considering the load of the system.

- *Resource control* – control over allocation of resources for running containers.
- *Load balancing* – forwarding incoming traffic to containers considering their load.
- *Service discovery* – resolving the internal IP address of the container that provides the requested service.
- *Monitoring* – statistics and logs collection for running containers.

The power of container orchestration tools is in the fact that you do not need to write any code to make it work, just specify desired properties via configuration file and see how your app is up and running. Among all the available orchestration tools Kubernetes is the most widely used one.

Kubernetes is from the ground designed to work on cluster of machines and distribute workloads across them. The top-level concepts in the Kubernetes cluster are Master and Worker Nodes. Master Node is a single machine that coordinates work of the whole cluster: monitors applications, maintains their desired state, performs updates, manages networking and communication between machines. The Worker Node is a machine that runs containers. Each Worker Node has a Kubelet process which performs communication with the Master Node and manages running containers and a container runtime that pulls images and runs applications. The great thing about Kubernetes is that it abstracts away all the infrastructure details about physical nodes in the cluster, so that it is not needed to configure which nodes the application should be deployed on, all that matters is declaring what components the application consists of and dependencies between them. To achieve this goal Kubernetes brings the following abstractions:

- *Deployment* – API object that represents application configuration (which components it consists of, how many replicas each container must have, etc.). Once this object is created in Kubernetes, the Master Node uses it to create Pods with specified containers on Worker Nodes of the cluster. Each Deployment has corresponding Deployment Controller that maintains the desired state of the application.
- *Pod* – is the smallest unit of deployment. A Pod contains a group of one or more related containers and resources shared between them. Such resources are storage, networking and containers specifications. Each Pod has a unique cluster IP address that is shared between all the containers in the Pod. Such IP addresses are internal in cluster and are not exposed to the outside world.
- *Service* – is an API objects that defines a logical set of Pods and rules by which they can be accessed. Services handle routing and load balancing between dependent Pods, which allows dependent Pods to die on one node and replicate on the other without breaking the communication.
- *Volume, Persistent Volume and Persistent Volume Claim* – those are API objects that abstract storage. Since local files in containers are transient (they die along with the container) it is needed to have a mechanism to store data outside the container. Volumes lifetime is tied to Pod`s one. But Pods are also ephemeral objects in Kubernetes, so there should be a mechanism to persist data beyond the Pod`s lifetime. Persistent Volume and Persistent Volume Claim serve this functionality.

Those are just concepts needed for basic application deployment, while there are much more things to configure in Kubernetes for the production-ready deployment. Kubernetes brings a lot of complexity and is a bit tough to get started, but once it is understood it provides all the functionality for automatic production-ready deployment and maintenance of applications that consist of hundreds of microservices. Since e-archive system is designed to scale horizontally and even the simplest deployment requires dozen microservices to be up and running, the Kubernetes is a perfect tool to choose.

VI. Conclusion

In this paper we have described and compared different architectures styles that can be used for e-archive system construction. Designing application architecture is not an easy task especially when it comes to distributed systems with large data volumes. Every software development process must start from analysis of application domain and requirements, especially non-functional ones, because they cause biggest impact on choosing architecture style and resulting system complexity. We have designed architecture that is based on Microservices and Event Driven architecture style, which accounts all the characteristics of the e-archive system domain and provides such important possibilities as horizontal scaling, modular updates, loose coupling of components, ease of modification and large volume handling. The technology stack chosen for that architecture implementation and application delivery is proven to be production-ready, has detailed documentation and large community support. One important point that can be seen from this analysis is that construction of microservices requires strong knowledge of cloud architecture patterns and mature DevOps culture, because it is not enough to just write code it also must be correctly deployed, orchestrated and monitored to bring the real value. The designed architecture has big cost of initial construction but offers chipper updates, modification and maintenance in future.

References

[1] Korb and S. Strodl, "Digital preservation for enterprise content: a gap-analysis between ECM and OAIS", in *7th International Conference on Preservation of Digital Objects*, Vienna, Austria, 2010, pp. 221-228.

[2] Consultative Committee for Space Data Systems, *Reference model for an open archival information system (OAIS) recommendation for space data system practices : recommended practice CCSDS 650.0-M-2.* Washington, DC Magenta Book, June 2012, p. 4-1.

[3] C. de la Torre, B. Wagner and M. Rousos, *.NET Microservices: Architecture for Containerized .NET Applications,* 2nd ed. Redmond: , Washington 98052-6399, 2019, pp. 311–324.

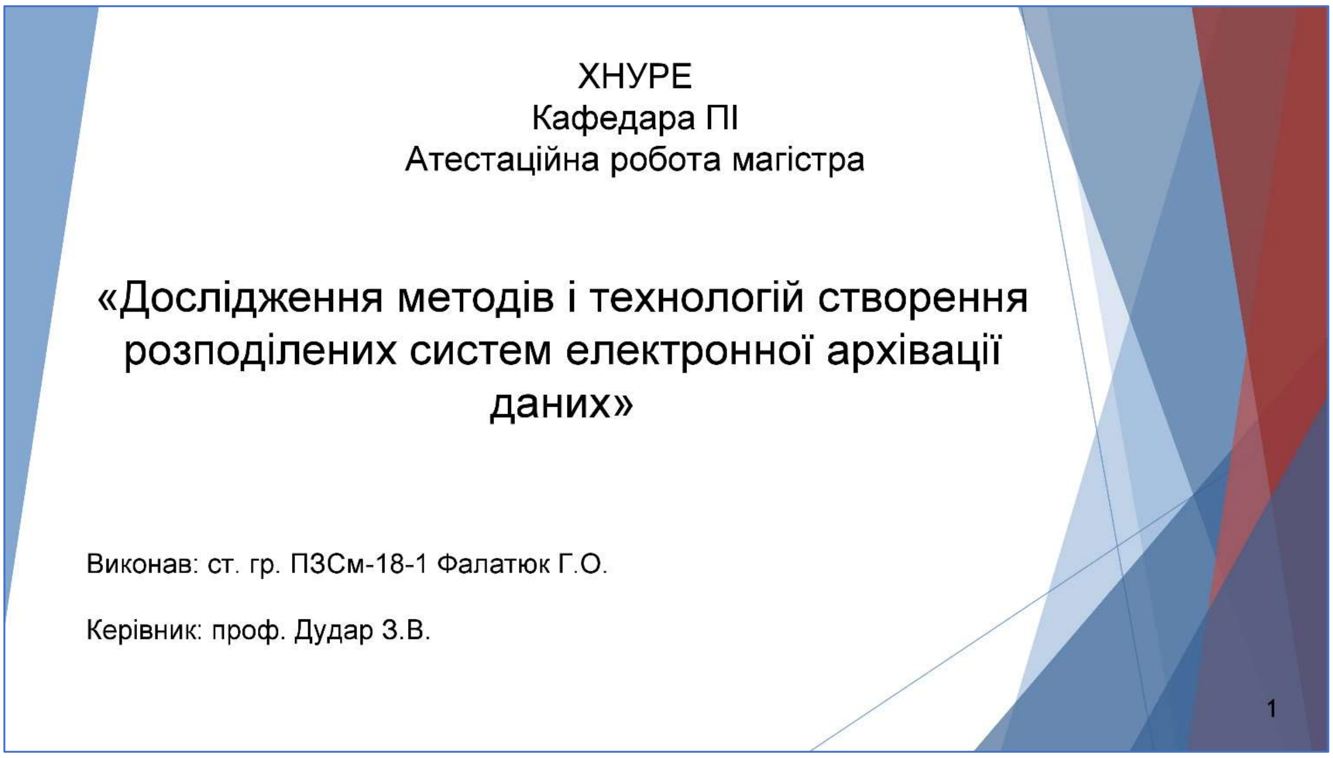
[4] A. Homer, J. Sharp, L. Brader, M. Narumoto and T. Swanson, *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications.* Microsoft patterns & practices, 2014, pp. 190-197.

[5] C. de la Torre, *Containerized Docker Application Lifecycle with Microsoft Platform Tools.* Redmond: Washington 98052-6399, 2017, pp. 1-12.

[6] J. Baier and J. White, *Getting Started with Kubernetes,* 2nd ed. Packt Publishing Ltd, 2015, pp. 87-130.

ДОДАТОК Б

Слайди презентації



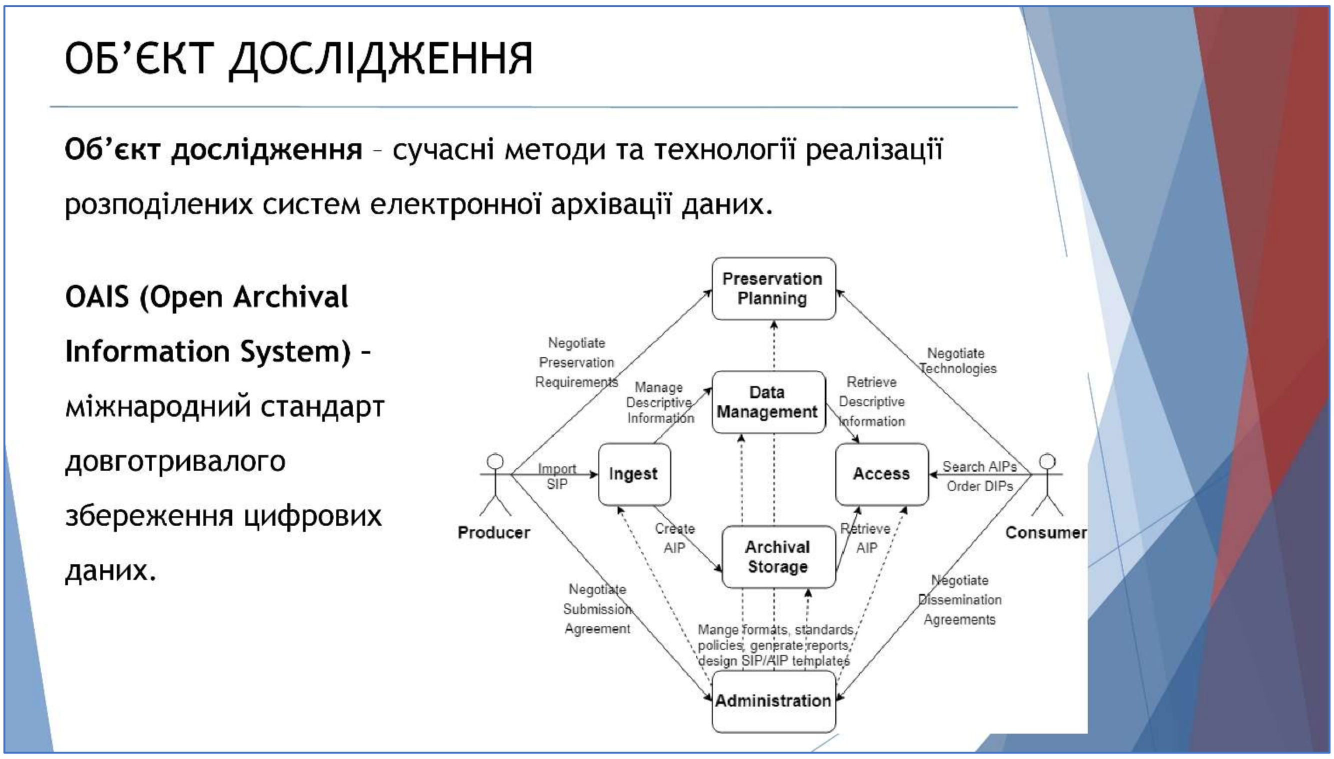
ХНУРЕ
Кафедара ПІ
Атестаційна робота магістра

«Дослідження методів і технологій створення розподілених систем електронної архівації даних»

Виконав: ст. гр. ПЗСм-18-1 Фалатюк Г.О.

Керівник: проф. Дудар З.В.

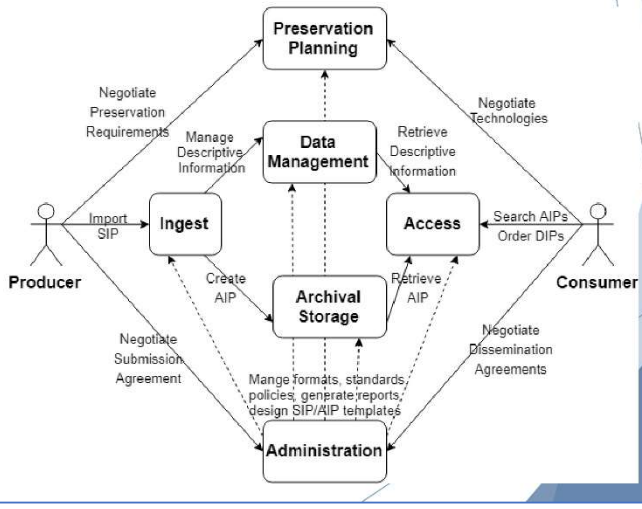
1

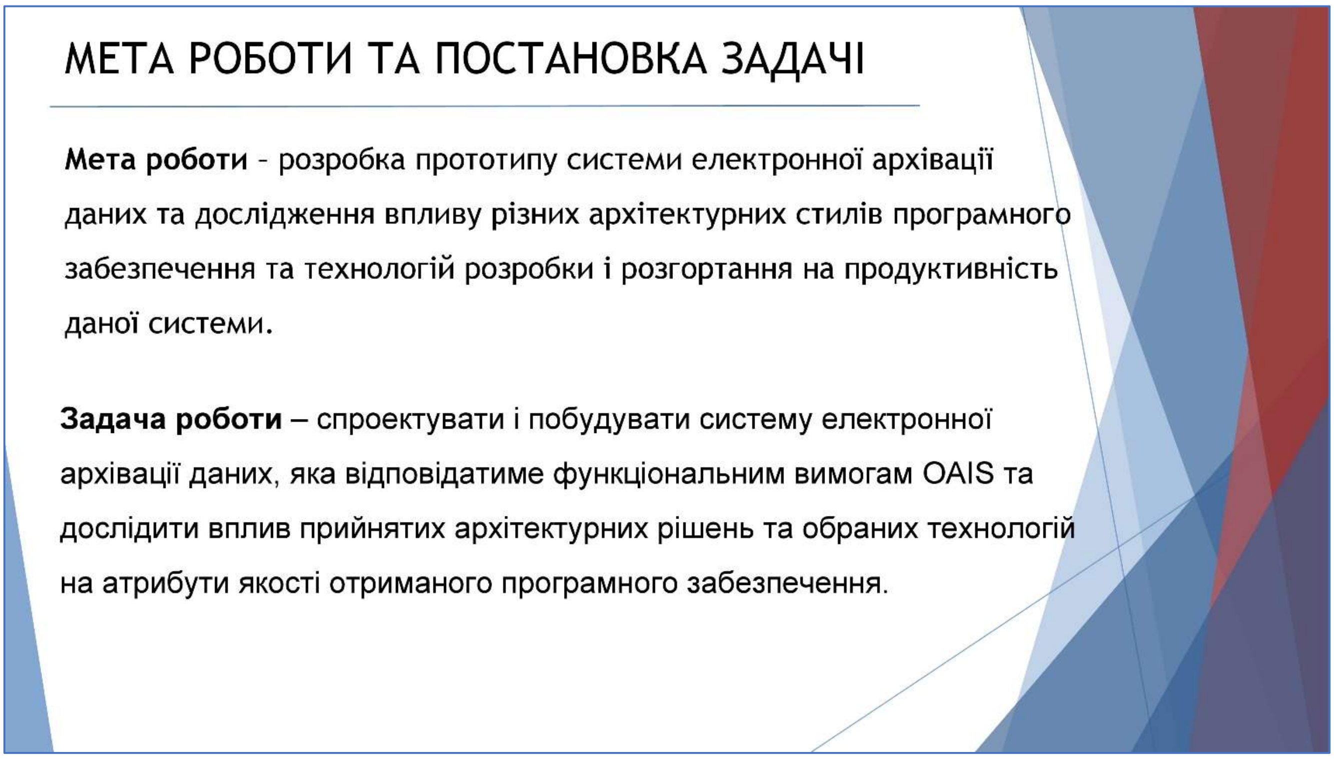


ОБ’ЄКТ ДОСЛІДЖЕННЯ

Об’єкт дослідження – сучасні методи та технології реалізації розподілених систем електронної архівації даних.

OAIS (Open Archival Information System) – міжнародний стандарт довготривалого збереження цифрових даних.


Preservation Planning
Negotiate Preservation Requirements
Negotiate Technologies
Manage Descriptive Information
Data Management
Retrieve Descriptive Information
Import SIP
Ingest
Access
Search AIPs
Order DIPs
Producer
Consumer
Create AIP
Archival Storage
Retrieve AIP
Negotiate Submission Agreement
Negotiate Dissemination Agreements
Mange formats, standards, policies, generate reports, design SIP/AIP templates
Administration

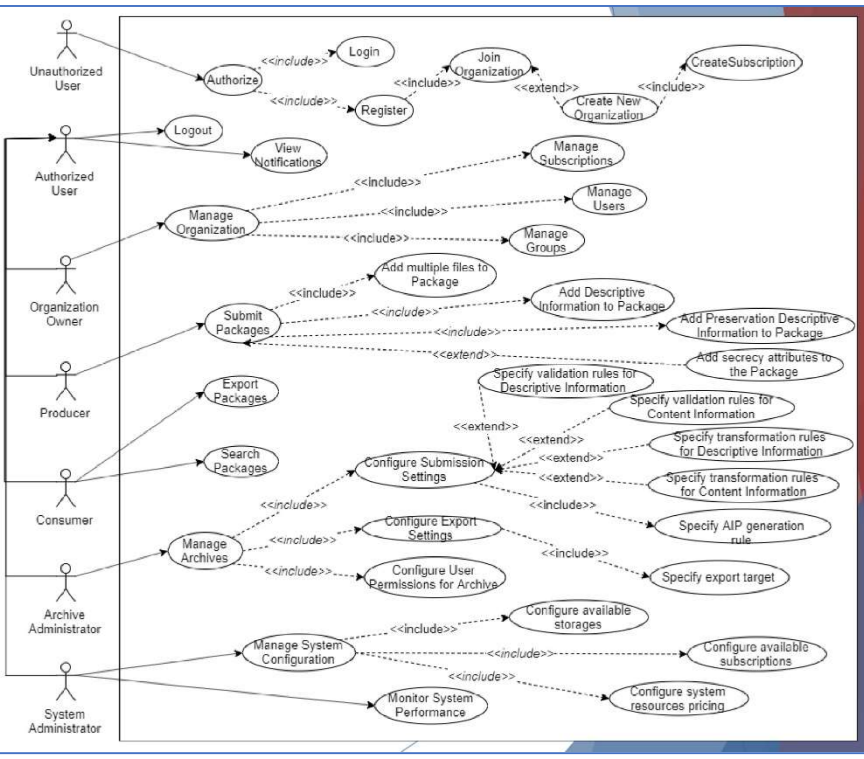


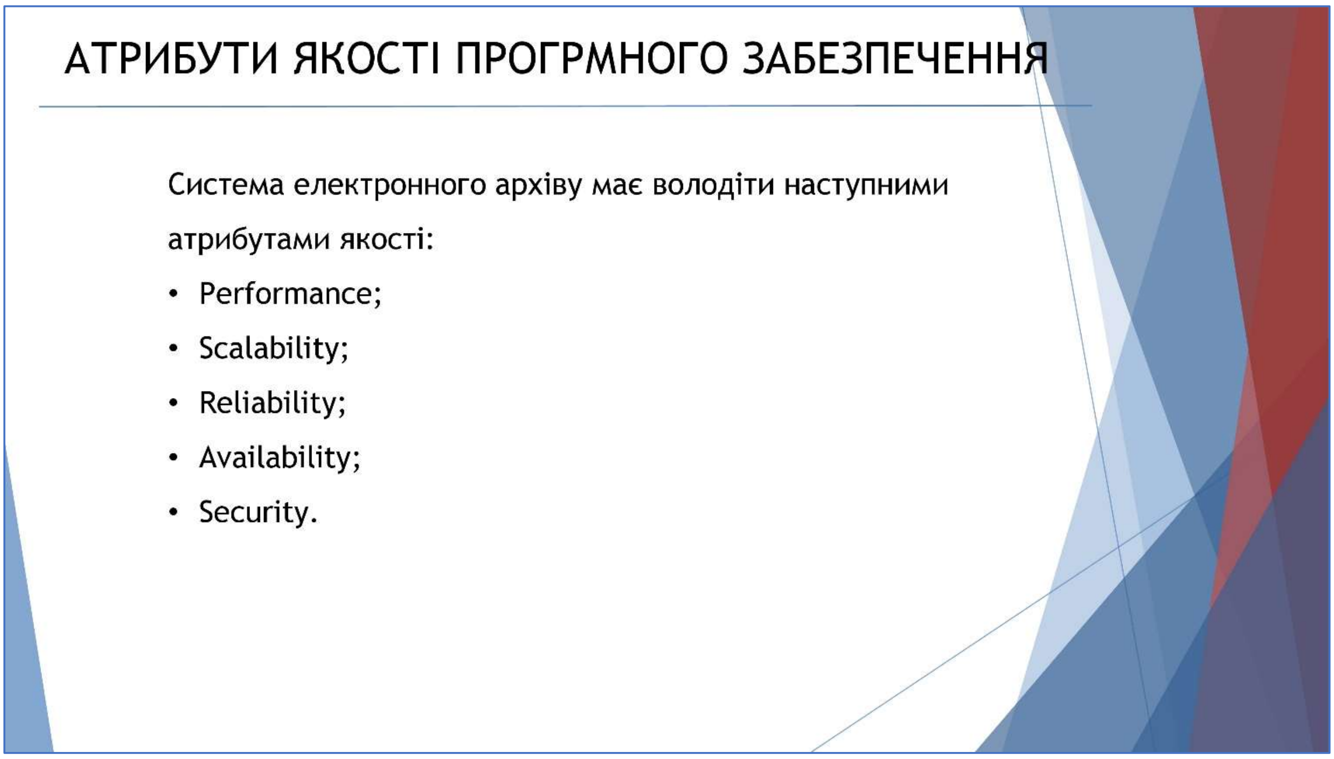
МЕТА РОБОТИ ТА ПОСТАНОВКА ЗАДАЧІ

Мета роботи – розробка прототипу системи електронної архівації даних та дослідження впливу різних архітектурних стилів програмного забезпечення та технологій розробки і розгортання на продуктивність даної системи.

Задача роботи – спроектувати і побудувати систему електронної архівації даних, яка відповідатиме функціональним вимогам OAIS та дослідити вплив прийнятих архітектурних рішень та обраних технологій на атрибути якості отриманого програмного забезпечення.

ДІАГРАМА ПРЕЦЕДЕНТІВ

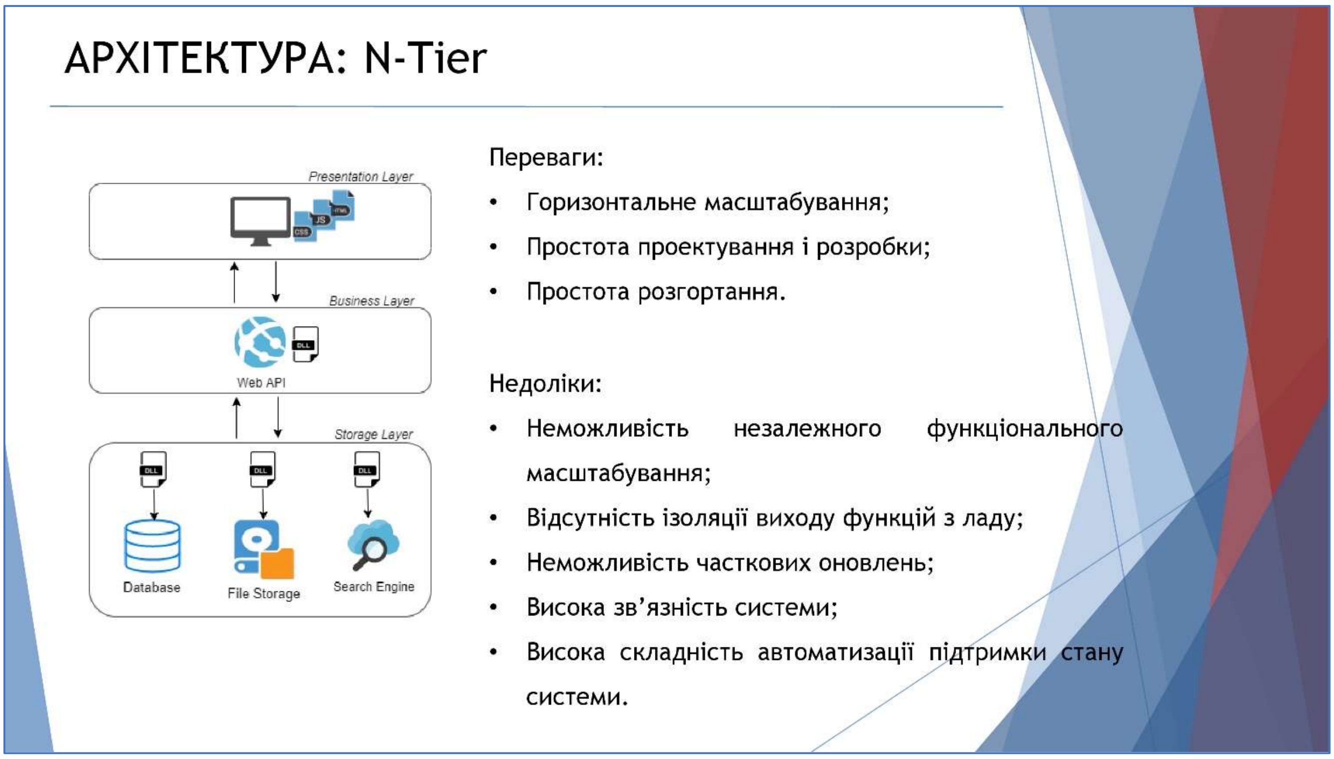

Unauthorized User
Authorized User
Organization Owner
Producer
Consumer
Archive Administrator
System Administrator
Authorize
<<include>>
Login
<<include>>
Register
<<include>>
Join Organization
<<extend>>
Create New Organization
<<include>>
CreateSubscription
Logout
View Notifications
Manage Organization
<<include>>
Manage Subscriptions
<<include>>
Manage Users
<<include>>
Manage Groups
Submit Packages
<<include>>
Add multiple files to Package
<<include>>
Add Descriptive Information to Package
<<include>>
Add Preservation Descriptive Information to Package
<<extend>>
Add secrecy attributes to the Package
Export Packages
Search Packages
Manage Archives
<<include>>
Configure Submission Settings
<<extend>>
Specify validation rules for Descriptive Information
<<extend>>
Specify validation rules for Content Information
<<extend>>
Specify transformation rules for Descriptive Information
<<extend>>
Specify transformation rules for Content Information
<<include>>
Specify AIP generation rule
<<include>>
Configure Export Settings
<<include>>
Specify export target
<<include>>
Configure User Permissions for Archive
Manage System Configuration
<<include>>
Configure available storages
<<include>>
Configure available subscriptions
<<include>>
Configure system resources pricing
Monitor System Performance



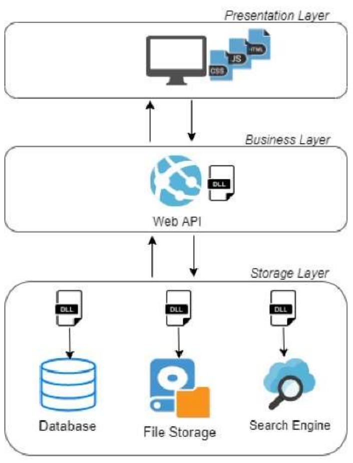
АТРИБУТИ ЯКОСТІ ПРОГРМНОГО ЗАБЕЗПЕЧЕННЯ

Система електронного архіву має володіти наступними атрибутами якості:

- Performance;
- Scalability;
- Reliability;
- Availability;
- Security.



АРХІТЕКТУРА: N-Tier



Переваги:

- Горизонтальне масштабування;
- Простота проектування і розробки;
- Простота розгортання.

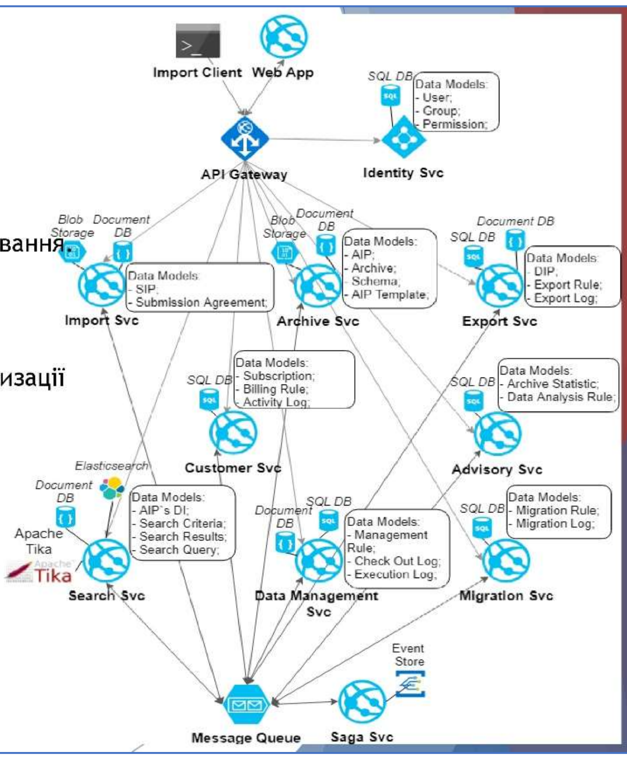
Недоліки:

- Неможливість незалежного функціонального масштабування;
- Відсутність ізоляції виходу функцій з ладу;
- Неможливість часткових оновлень;
- Висока зв’язність системи;
- Висока складність автоматизації підтримки стану системи.

АРХІТЕКТУРА: Microservices

Переваги:

- Горизонтальне масштабування;
- Підтримка незалежного функціонального масштабування;
- Ізоляція виходу функцій з ладу;
- Можливість часткових оновлень;
- Наявність протестованих відкритих засобів автоматизації
- розгортання і підтримки стану системи.



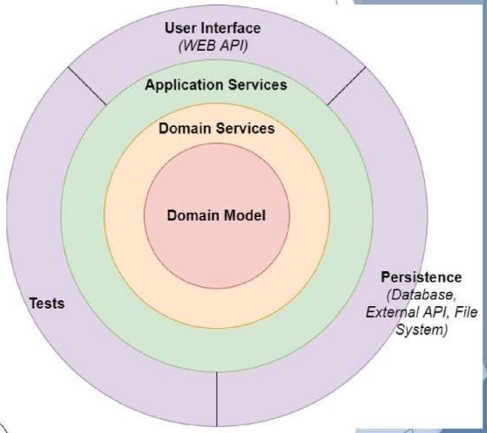
Недоліки:

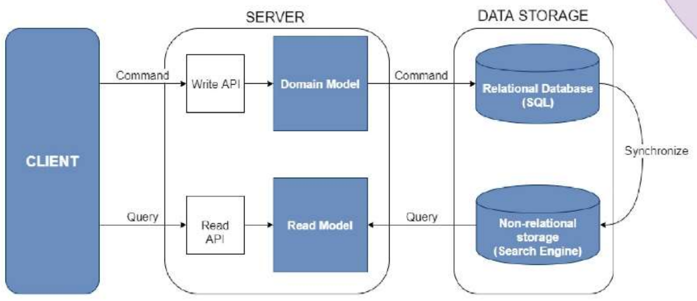
- Висока складність проектування і розробки системи;
- Потребує високо-кваліфікованих інженерів для розробки

ШАБЛОНИ ПРОЕКТУВАННЯ

Шаблони проектування:

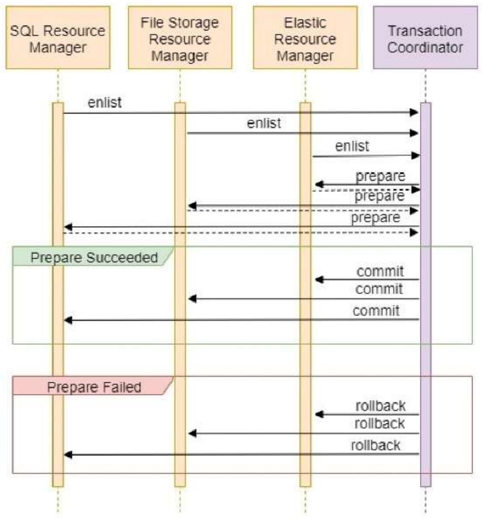
- Onion Architecture
- CQRS + Mediator
- Repository + Unit of Work
- Compensating Transaction
- Transaction Log Tailing



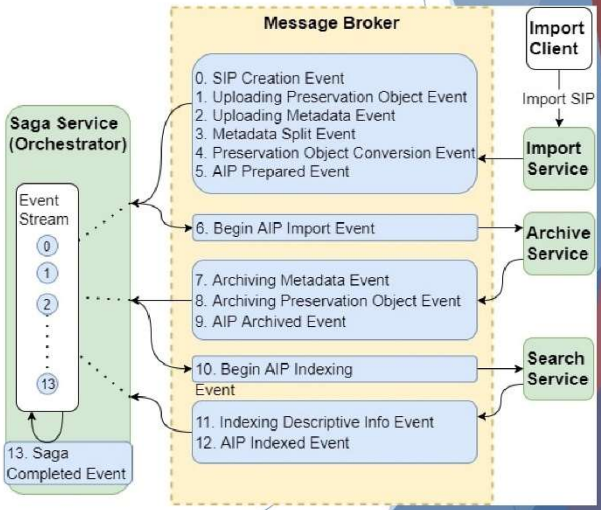


ЦІЛІСНІСТЬ ДАНИХ

Transactional Consistency



Eventual Consistency


Message Broker
Import Client
Import SIP
0. SIP Creation Event
1. Uploading Preservation Object Event
2. Uploading Metadata Event
3. Metadata Split Event
4. Preservation Object Conversion Event
5. AIP Prepared Event
Import Service
Saga Service (Orchestrator)
Event Stream
0
1
2
13
6. Begin AIP Import Event
Archive Service
7. Archiving Metadata Event
8. Archiving Preservation Object Event
9. AIP Archived Event
10. Begin AIP Indexing Event
Search Service
11. Indexing Descriptive Info Event
12. AIP Indexed Event
13. Saga Completed Event

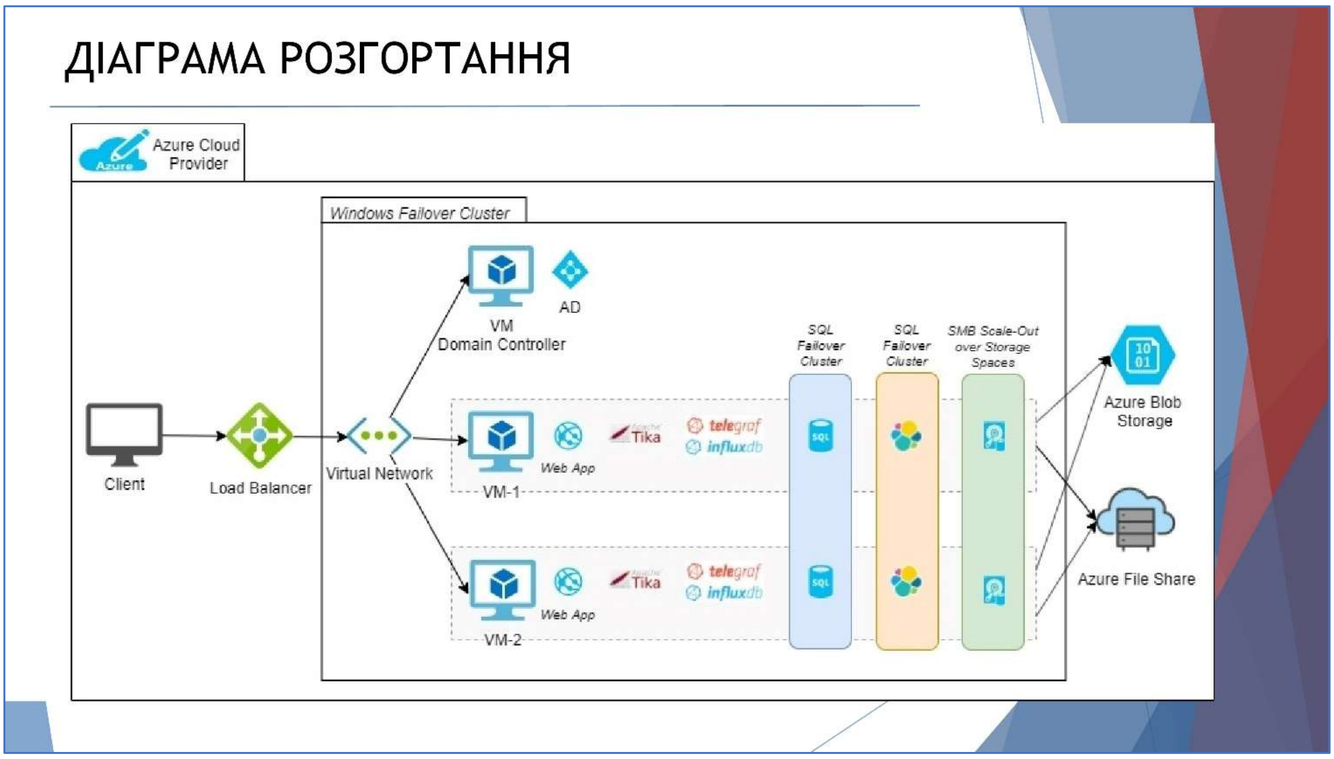
ТЕХНОЛОГІЇ

Технології розробки:

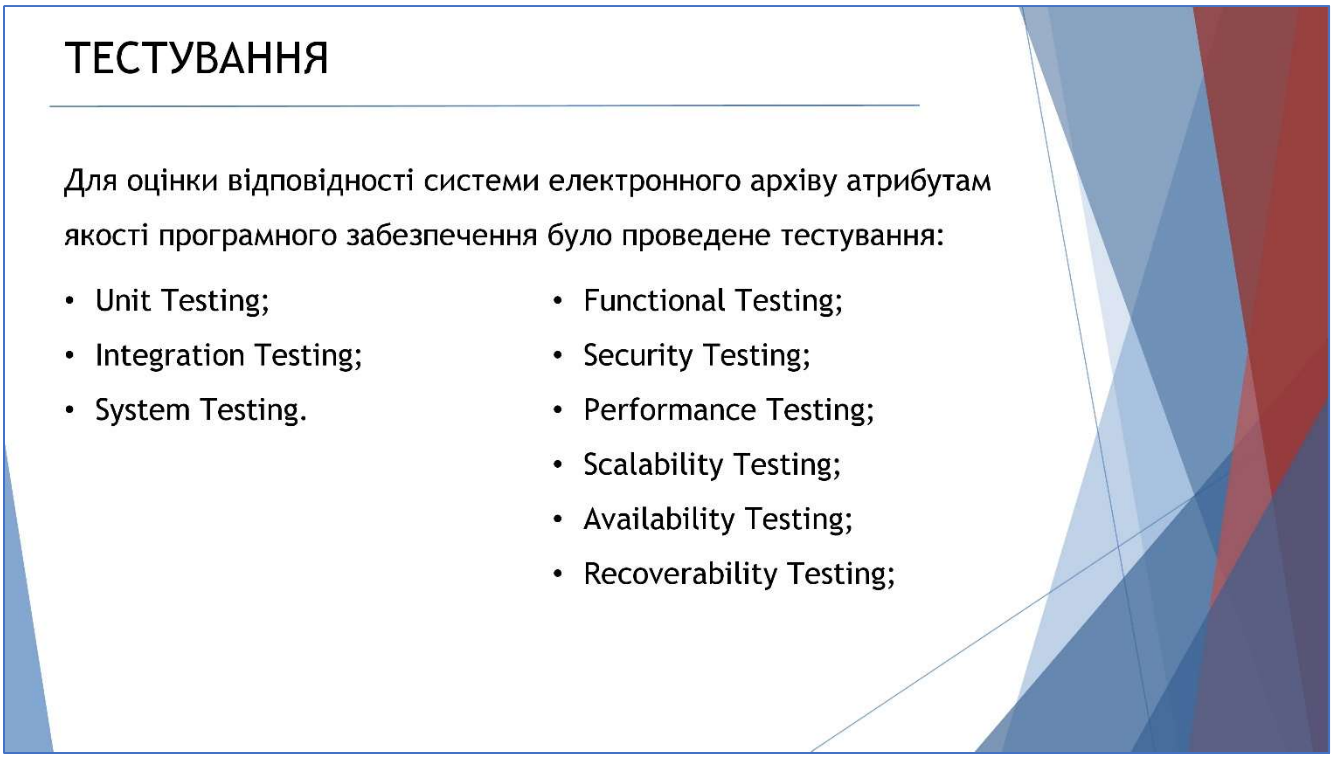
- Asp .Net Framework MVC та Web API 2.0
- MS SQL Server
- Elasticsearch
- Apache Tika
- Azure Blob Storage
- SMB File Share
- Microsoft Distributed Transaction Coordinator
- Angular JS
- Bootstrap 3.0

Технології розгортання і впровадження:

- Azure
- Virtual Machines
- Windows Failover Cluster
- SQL Server Failover Cluster Instance
- SMB Scale Out
- Azure File Share



ДІАГРАМА РОЗГОРТАННЯ



ТЕСТУВАННЯ

Для оцінки відповідності системи електронного архіву атрибутам якості програмного забезпечення було проведене тестування:

- Unit Testing;
- Integration Testing;
- System Testing.
- Functional Testing;
- Security Testing;
- Performance Testing;
- Scalability Testing;
- Availability Testing;
- Recoverability Testing;

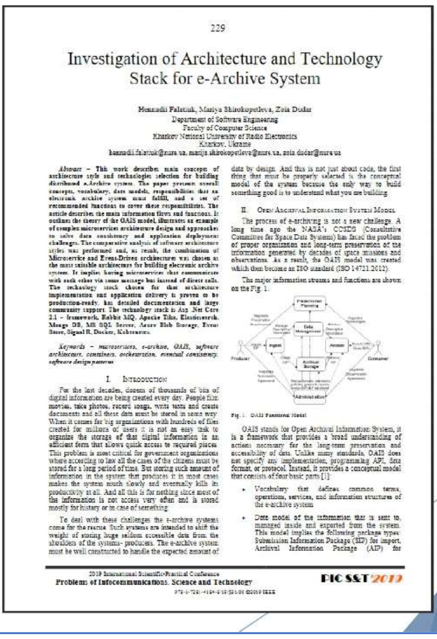
ПЕРСПЕКТИВИ РОЗВИТКУ

Варто відзначити, що створена система має відкриті питання для подальшого дослідження:

- Розробка стратегії мінімізації витрат для розгортання системи у використання;
- Дослідження меж масштабування запропонованої архітектури;
- Реалізація мікросервісної архітектури та дослідження зміни атрибутів якості системи;
- Використання Docker контейнерів для розгортання в Kubernetes кластері та дослідження впливу цих технологій на атрибути якості системи у порівнянні зі звичайними віртуальними машинами

АПРОБАЦІЯ РЕЗУЛЬТАТІВ

Результати роботи опубліковані у статті «Investigation of Architecture and Technology Stack for e-Archive System» на міжнародній науковій IEEE конференції «Problems of Infocommunications. Science and Technology (PIC S&T'2019)»



229

Investigation of Architecture and Technology Stack for e-Archive System

Hennadii Falatiuk, Mariya Shirokopetleva, Zoia Dudar
Department of Software Engineering
Faculty of Computer Science
Kharkov National University of Radio Electronics
Kharkov, Ukraine
hennadii.falatiuk@nure.ua, mariya.shirokopetleva@nure.ua, zoia.dudar@nure.ua

Abstract – This work describes main concepts of architecture style and technologies selection for building distributed e-Archive system. The paper presents overall concepts, vocabulary, data models, responsibilities that an electronic archive system must fulfill, and a set of recommended functions to cover these responsibilities. The article describes the main information flows and functions. It outlines the theory of the OAIS model, illustrates an example of complex microservices architecture design and approaches to solve data consistency and application deployment challenges. The comparative analysis of software architecture styles was performed and, as result, the combination of Microservice and Event-Driven architectures was chosen as the most suitable architecture for building electronic archive system. It implies having microservices that communicate with each other via some message bus instead of direct calls. The technology stack chosen for that architecture implementation and application delivery is proven to be production-ready, has detailed documentation and large community support. The technology stack is Asp .Net Core 2.1 – framework, Rabbit MQ, Apache Tika, Elasticsearch, Mongo DB, MS SQL Server, Azure Blob Storage, Event Store, SignalR, Docker, Kubernetes.

Keywords – microservices, e-archive, OAIS, software architecture, containers, orchestration, eventual consistency, software design patterns

I. Introduction

For the last decades, dozens of thousands of bits of digital information are being created every day. People film movies, take photos, record songs, write texts and create documents and all these data must be stored in some way. When it comes for big organizations with hundreds of files created for millions of users it is not an easy task to organize the storage of that digital information in an efficient form that allows quick access to required pieces. This problem is most critical for government organizations where according to law all the cases of the citizens must be stored for a long period of time. But storing such amount of information in the system that produces it in most cases makes the system much slowly and eventually kills its productivity at all. And all this is for nothing since most of the information is not access very often and is stored mostly for history or in case of something.

To deal with these challenges the e-archive systems come for the rescue. Such systems are intended to shift the weight of storing huge seldom accessible data from the shoulders of the systems- producers. The e-archive system must be well constructed to handle the expected amount of data by design. And this is not just about code, the first thing that must be properly selected is the conceptual model of the system because the only way to build something good is to understand what you are building.

II. Open Archival Information System Model

The process of e-archiving is not a new challenge. A long time ago the NASA's CCSDS (Consultative Committee for Space Data Systems) has faced the problem of proper organization and long-term preservation of the information generated by decades of space missions and observations. As a result, the OAIS model was created which then become an ISO standard (ISO 14721:2012).

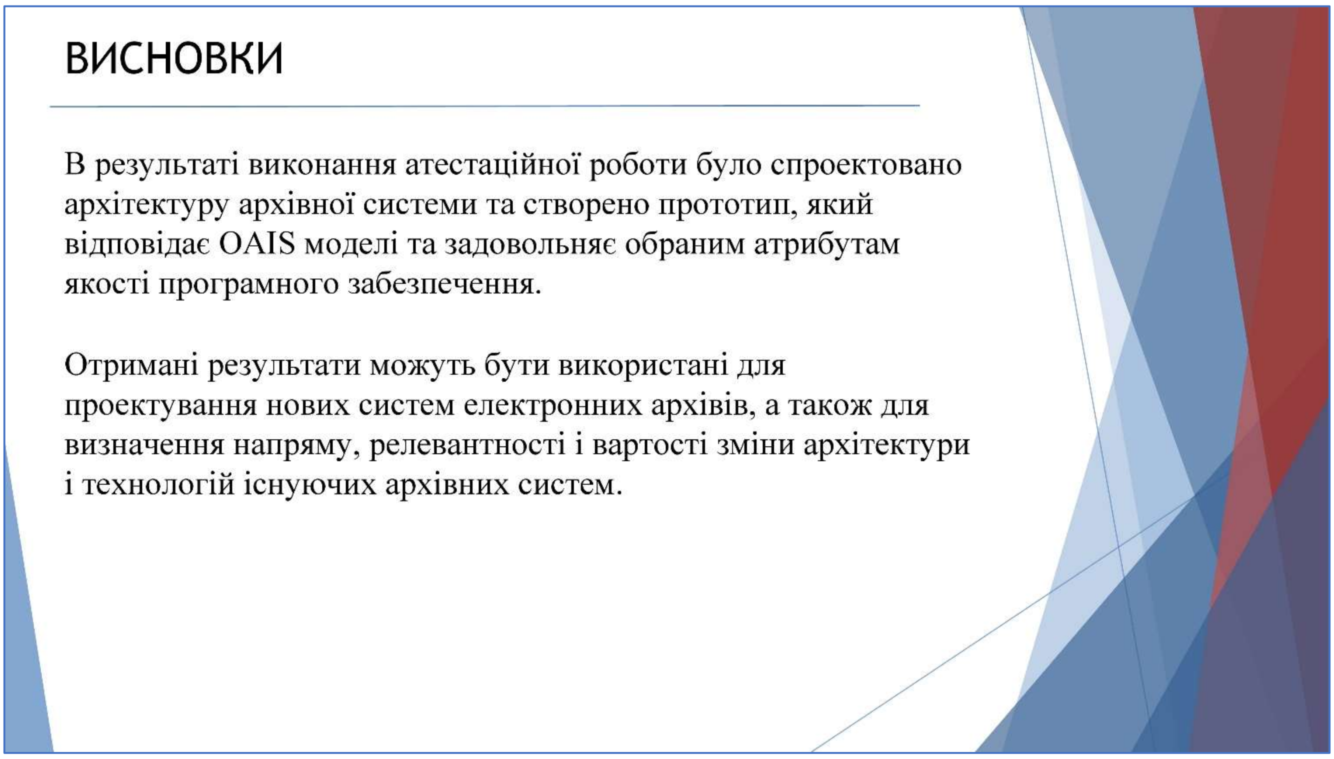
The major information streams and functions are shown on the Fig. 1.

Fig. 1. OAIS Functional Model

OAIS stands for Open Archival Information System, it is a framework that provides a broad understanding of actions necessary for the long-term preservation and accessibility of data. Unlike many standards, OAIS does not specify any implementation, programming API, data format, or protocol. Instead, it provides a conceptual model that consists of four basic parts [1]:

- Vocabulary that defines common terms, operations, services, and information structures of the e-archive system.
- Data model of the information that is sent to, managed inside and exported from the system. This model implies the following package types: Submission Information Package (SIP) for import, Archival Information Package (AIP) for

2019 International Scientific-Practical Conference
Problems of Infocommunications. Science and Technology
PIC S&T'2019
978-1-7281-4184-2/19/$31.00 ©2019 IEEE



ВИСНОВКИ

В результаті виконання атестаційної роботи було спроектовано архітектуру архівної системи та створено прототип, який відповідає OAIS моделі та задовольняє обраним атрибутам якості програмного забезпечення.

Отримані результати можуть бути використані для проектування нових систем електронних архівів, а також для визначення напряму, релевантності і вартості зміни архітектури і технологій існуючих архівних систем.

ДОДАТОК В

Відгук та рецензії

Рецензія
на атестаційну роботу магістра
студента групи ПЗСм-18-1 Фалатюка Геннадія Олександровича
спеціальність – 121- Інженерія програмного забезпечення
освітньо-професійна програма «Програмне забезпечення систем»
«Дослідження методів і технологій створення розподілених систем електронної архівації даних».

(Тема атестаційної роботи)

Структура атестаційної роботи: пояснювальна записка 86 сторінок; графічна частина 15аркушів; програмне застосування (прикладна програма) 127 файлів загальним обсягом 32.8Мбайт.

В атестаційній роботі представлено прототип системи електронного архіву, який реалізує основні функції згідно з міжнародним стандартом збереження цифрових даних OAIS (Open Archival Information System). Розроблена програмна система відповідає таким основним атрибутам якості програмного забезпечення, як продуктивність, масштабованість, доступність, надійність, безпека. Насьогодні, в умовах інтенсивного зростання об'ємів цифрових даних, питання довготривалого їх збереження є надзвичайно важливим і тому тема атестаційної роботи є достатньо актуальною.

Пояснювальна записка до розробленого проекту відповідає усім вимогам, що висуваються до атестаційних робіт: доцільне розміщення тексту та графічних матеріалів виходячи з тем розділів та підрозділів, вміле застосування літератури та посилань на неї та виконане на достатньо високому рівні. Програмне забезпечення, надане на рецензування, відповідає завданню та є працездатним.

Аналіз предметної галузі, атрибутів якості програмного забезпечення, архітектурних стилів, шаблонів проектування і технології було проведено на високому рівні. Варто відзначити доцільність обраних студентів методів і засобів розробки, тестування і впровадження програмного забезпечення. Усі прийняті технічні рішення є обґрунтованими і зваженими.

Розроблений прототип системи електронного архіву, а також результати дослідження впливу шаблонів проектування і технологій на атрибути якості системи єдостовірним і можуть бути використані при створенні систем зі схожими функціональними і нефункціональними вимогами.

Серед недоліків атестаційної роботи магістра варто відзначити неінтуїтивний інтерфейс користувача, відсутність підтримки SIP пакетів у форматі FGS Arendehantering та відсутність підтримки PDI пакету у форматі PREMIS. Проте зазначені недоліки значно не впливають на якість представленої роботи.

В цілому атестаційна робота магістранта групи ПЗСм-18-1 Фалатюка Г.О. відповідає вимогам до атестаційних робіт і заслуговує оцінки «відмінно» (96 балів, А). Атестаційну роботу можна представити для захисту в ЕК за спеціальністю 121- Інженерія програмного забезпечення, освітньо-професійною програмою «Програмне забезпечення систем».

Рецензент [illegible] к.т.н, доц. каф ПУ Галян В.В

Рецензія
на атестаційну роботу магістра
студента групи ПЗСм-18-1 Фалатюка Геннадія Олександровича
спеціальність – 121- Інженерія програмного забезпечення
освітньо-професійна програма «Програмне забезпечення систем»
«Дослідження методів і технологій створення розподілених систем електронної архівації даних».

(Тема атестаційної роботи)

Структура атестаційної роботи: пояснювальна записка 86 сторінок; графічна частина 15аркушів; програмне застосування (прикладна програма)127файлів загальним обсягом 32.8Мбайт.

Представлений в атестаційній роботі прототип системи електронного архіву відповідає функціональним вимогам моделі OAIS та обраним атрибутам якості програмного забезпечення. Актуальність теми роботи зумовлена необхідністю створення надійного сховища цифрових даних в умовах стрімкого зростання їх об'ємів.

Пояснювальна записка відповідає стандартам та вимогам, які висуваються щодо атестаційних робіт магістрів. Розділи пояснювальної записки пропорційні, написані грамотно. У роботі автор продемонстрував вміння працювати з науковою літературою, в достатній мірі процитував її в пояснювальній записці. Програмне забезпечення, надане на рецензування, відповідає завданню та є працездатним.

Технічні рішення, прийняті під час аналізу і розробки програмного забезпечення є оптимальними, використання шаблонів проектування і технологій розробки й впровадження системи є обґрунтованим з зваженим.

Дослідження впливу архітектури, шаблонів проектування і технологій на атрибути якості системи проведено з використанням провідних засобів і методів розробки програмного забезпечення. Отримані результати є достовірними і можуть бути використані при розробці розподілених систем, що відповідають необхідних атрибутів якості програмного забезпечення.

З недоліків атестаційної роботи магістра слід відзначити відсутність порівняння використання альтернативних технологій реляційних і нереляційних баз даних, окрім загальновідомого MSSQLServer, і відсутність дослідження використання Docker контейнерів для розгортання в Kubernetes кластері та впливу цих технологій на атрибути якості системи у порівнянні зі звичайними віртуальними машинами. Проте результати поточної роботи можуть бути використані при подальшому дослідженні вищезазначених питань.

В цілому атестаційна робота магістранта групи ПЗСм-18-1 Фалатюка Г.О. відповідає вимогам до атестаційних робіт і заслуговує оцінки «відмінно» (95 бал., В). Атестаційну роботу можна представити для захисту в ЕК за спеціальністю 121- Інженерія програмного забезпечення, освітньо-професійною програмою «Програмне забезпечення систем».

Рецензент к.т.н., доц., доцент кафедри ЦІ [illegible] /Захарій В.В./

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет комп'ютерних наук

ВІДГУК

на атестаційну роботу магістра
Фалатюка Геннадія Олександровича,
спеціальність 121- Інженерія програмного забезпечення
освітньо-професійна програма «Програмне забезпечення систем»
Тема атестаційної роботи: Дослідження методів і технологій створення розподілених систем електронної архівації даних

Атестаційна робота магістра Фалатюка Г.О. присвячена дослідженню методів і технологій створення розподілених систем електронної архівації даних. Забезпечення надійного сховища для великих обсягів цифрових даних є надзвичайно важливим завданням в сучасному світі. Вирішення проблеми довгострокового зберігання даних описано в моделі OAIS (Open Archival Information Model), яка насьогодні є міжнародним стандартом цифрового збереження. Оскільки OAIS описує загальні поняття, моделі даних та рекомендованих функції, які має виконувати електронний архів, питання будування програмного забезпечення, яке відповідатиме усім вимогам, залишається відкритим. У час стрімкого розвитку відкритого програмного забезпечення та хмарних технологій, постає актуальним питання вибору архітектури, шаблонів проектування і технологій розробки і впровадження, з урахуванням досягнення необхідних атрибутів якості програмного забезпечення.

Робота виконана якісно, самостійно, під час розробки студент показав знання та вміння використовувати в практичній діяльності засоби розробки Visual Studio, Azure Dev Ops та Azure. У процесі роботи студент продемонстрував серйозне відношення до роботи, високий рівень підготовленості до самостійних наукових досліджень та самостійної роботи, показав уміння користуватися науково-технічною літературою, ресурсами мережі Інтернет, виявив глибокі знання в області алгоритмізації та мов програмування.

Під час виконання атестаційної роботи була проаналізована предметна галузь, досліджені вплив архітектурних стилів і шаблонів проектування на атрибути якості програмного забезпечення, правильно виконана постановка завдання, обрана архітектура та технології, виконано проектування та реалізація прототипу системи електронного архіву. Результати роботи відповідають завданню на атестаційну роботу і є автентичними, про що свідчить низький відсоток схожості з іншими роботами в Інтернеті.

Магістрант гр. ПЗСм-18-1 Фалатюк Г.О. готовий до самостійної інженерної діяльності. Атестаційну роботу можна подати до захисту в ЕК за спеціальністю 121 – «Інженерія програмного забезпечення», освітньо-професійною програмою «Програмне забезпечення систем».

«______» _____________ 20__р.

____
підпис

Керівник атестаційної роботи магістра
проф. Дудар З.В.