Usage of OpenAPI specification in distributed microservices-oriented architecture of the information system for astronomical data processing

Stanislav Orlov[1,*,†], Tetiana Trunova[2,†], Emil Hadzhyiev[2,†], Hanna Okhotko[3,†], Yehor Bondar[2,†] and Yuriy Netrebin[4,†]

[1] National Aerospace University – Kharkiv Aviation Institute, Chkalova street 17, Kharkiv, 61000, Ukraine
[2] Kharkiv National University of Radio Electronics, Nauki avenue 14, Kharkiv, 61166, Ukraine
[3] Odesa I. I. Mechnykov National University, Pastera street 42, Odesa, 65082, Ukraine
[4] INTIVE Limited, O'Connell Bridge House, D'Olier Street 2, Dublin, D02RR99, Ireland

Abstract

In this paper we presented the usage of OpenAPI specification in distributed microservices-oriented information system for astronomical data processing. A common goal of all scientific and technological algorithms and methods is to automate as much as available processes without any human actions. In general cases it can be done by the different astronomical distributed microservices-oriented information system. In these pipelines the various data mining and knowledge discovery in databases (KDD) tasks are used for speeding up and optimizing the astronomical data processing. Suggested using of the OpenAPI specification in a distributed microservices-oriented information system for astronomical data processing significantly improves the system's interoperability, scalability, and maintainability. The developed skeleton of the real example of astronomical data-processing system is implemented using .Net Core framework and C# programming language. Implementing Swagger in a microservices architecture presents numerous benefits, significantly enhancing both the development and maintenance phases of service-oriented applications. The developed skeleton and the proposed approach will be useful for the different microservices-oriented information system for astronomical data processing. It can be used for all kind of processing astronomical images using the different mathematical methods and algorithms implemented as a tool, module, or service. Another one good example of application the proposed skeleton is a realization of the Virtual Observatory (VO) concept or integration with CI/CD tools.

Keywords

Information system, client-server architecture, microservices-oriented architecture, scalability, processing pipeline, data mining, knowledge discovery in databases, astronomical observations, image processing, Solar System objects, Swagger, OpenAPI, REST API, JSON, .NET, C#, RabbitMQ

1. Introduction

The asteroid-comet hazard becomes a huge potential problem in the XXI century [1], which can cause the global destructions, collisions with geostationary artificial satellites [2], space debris, etc. To avoid such situation the humanity is continuously developing and improving mathematical methods [3] and algorithms for the astronomical scientific direction like an astronomical image processing and computer vision [4], which includes the background alignment [5], brightness equalization [6], astrometric reduction [7], photometric reduction [8], detection of moving objects in series of frames, or even discovery of the Solar System objects (SSOs) [9], like comets, asteroids [10], small planets, galaxies, stars, etc.

All astronomical scientific observations are created by the charge-coupled device (CCD) [11] that

ICST-2024: Information Control Systems & Technologies, September 23-25, 2023, Odesa, Ukraine
s.v.orlov@student.khai.edu (S. Orlov); tetiana.trunova@nure.ua (T. Trunova); emil.hadzhyiev@nure.ua (E. Hadzhyiev); hannaokhotko@gmail.com (H. Okhotko); yehor.bndr@gmail.com (Ye. Bondar); yuriy.n.netrebin@gmail.com (Yu. Netrebin)
0009-0008-0680-206X (S. Orlov); 0000-0003-2689-2679 (T. Trunova); 0009-0003-6752-7827 (E. Hadzhyiev); 0009-0002-1403-4298 (H. Okhotko); 0009-0001-5309-0084 (Ye. Bondar); 0009-0001-8778-3241 (Yu. Netrebin)

are used as a main equipment in telescopes or any other optical system at the observatories. Such astronomical scientific observations are collected during the specified observational period of the investigated small celestial SSOs [12], as well as the artificial satellites. After performing the series of observations of the investigated SSOs it is required to analyze the results of observation, which can include the period and shape of rotations determining of such investigated SSOs. It means that there are a lot of astronomical big data and to process it we need to apply the different approaches of the information technologies.

The astronomical scientific information is also can be collected from the different historical clusters, archives, Virtual Observatories [13], data clouds, astronomical astrometric and photometric catalogues [14], different servers and other storages.

A common goal of all scientific and technological algorithms and methods is to automate as much as available processes without any human actions. In general cases it can be done by the different astronomical scientific information systems. In these information systems the various data mining [15] and knowledge discovery in databases (KDD) [16] tasks are used for speeding up and optimizing the astronomical data processing.

In case if the astronomical scientific information system is a very complex and consists of the different mathematical modules and libraries it becomes distributed microservices-oriented information system for astronomical data processing.

Microservices, also known as the microservice architecture, is an architectural style that structures an application as a collection of loosely coupled services, each of which implement business capabilities. The microservice architecture enables the continuous delivery and deployment of large, complex information systems. It also enables an organization to evolve its technology stack, scale and be more resilient with time. Microservice architecture advocates for developing a single information system into a collection of loosely associated services. These units also enable the continuous delivery and deployment of large, monolithic information systems with minimal need for centralization.

As microservices architecture [17] continues to grow in popularity, the complexity of managing multiple, interrelated services increases. Documentation becomes essential not just for external users but also for internal developers who need to understand the APIs provided by each service. That's where Swagger comes into play. Swagger, now known as the OpenAPI Specification, is a powerful tool for describing, producing, consuming, and visualizing RESTful web services.

Swagger simplifies API development and maintenance by providing a language-agnostic interface to REST APIs [18]. With Swagger, you can generate client libraries, server stubs, and API documentation that facilitates clear communication amongst your development team and beyond. It ensures that all microservices speak the same 'language' when it comes to API endpoints, parameters, and data models.

This paper aims to the analysis of main focuses and features of OpenAPI specification for microservices development. Real examples of astronomical data-processing system are implemented using .Net Core framework and C# programming language, which is perfectly designed for the developing of distributed microservices-oriented information system. Section 2 presents the several technologies related to our work for solving of API documentation task. Section 3 elaborates the system architecture based on microservices architecture style, presents the integration of Swagger OpenAPI specification in real microservice implementation. Integrated data models for astronomical data-processing system are also presented in this section as well as the result of execution is illustrated in it. This section aldo aims to the discussions about advantages of the proposed usage of OpenAPI specification in distributed microservices-oriented information system for astronomical data processing. The paper ends with a conclusion in section 4, which illustrates the conclusions and outlines of future work as well as possibilities for future investigations and enhancements.

2. Related Works

Each SSO in a digital frame has a typical form of its image [19]. The common methods for the image

processing [20] and machine vision [21] are developed for detection/recognition such images of SSOs and an estimation of their positional and motion parameters [22]. Such methods are based on the analysis of only those pixels that potentially belong to the investigated object. The disadvantages of such methods are very low accuracies when the typical form of object has a different shape [23].

The methods for assessing the aperture brightness [24] of object's images will work only with a single image of each SSO. Any methods for the matched filtration [25] and high-frequency filtration, which are devoted to the improving the quality of corrupted images are very resource consuming. The disadvantages of the methods are the big complexity and low accuracy during the astronomical data processing, when an object's image has a several peaks of magnitude.

Methods for the Wavelet analysis [26] or even time series analysis [27] are not so effective, because we do not have a big volume of the input data to be analyzed. Also, the disadvantage of such algorithms is the corrupting of the general statistics and possibility to process only clear measurements without any deviations in the typical form of image.

Any methods for the deep learning and pattern recognition [28] also require a big amount of astronomical data for training. The problem of such methods that astronomical image has a lot of artifacts, so there are a lot of false objects are detected in series of frames.

Almost all information systems related to the astronomical image processing are desktop applications with monolith structure. They require very careful installation, dependencies on 3rd party libraries, operational systems, frameworks, drivers. The processing time is fully depending on abilities of computers, on which the desktop application is installed. Such information systems don't have scalability and splitting on processes, so if any algorithm fails the whole system will be stuck. In this case to work with resource consuming mathematical algorithms, methods, and modules, which implement them, the distributed microservices-oriented information system architecture for astronomical data processing is required. And OpenAPI specifications is a good approach for such purposes.

There are several alternatives to Swagger for implementing OpenAPI specifications, each offering unique features and benefits that might be more suitable depending on your specific requirements. Here are some notable alternatives mentioned below.

Postman is a versatile tool for API development and testing [29]. Postman enables automated testing, team collaboration, and integration with various CI/CD tools. It also includes features like mock servers and interactive API documentation, making it a comprehensive solution for managing the API lifecycle. Author describes microservice architecture as a scalable method for designing and implementing online applications. Due to their network-based nature, microservice applications require testing within a network environment. Automating these tests involves generating artificial network traffic, typically in the form of HTTP requests to APIs such as REST APIs. These topics are explored from the perspectives of test design and implementation, alongside key features of microservice architecture and automated testing in general. The core of this thesis details the process of designing and implementing a test automation framework for Intel Insight, an automatic image storage and photogrammetry processing platform built as a microservice system.

The Stoplight platform excels in the areas of API design, documentation, and governance [30]. It features a user-friendly interface for creating API specifications with OpenAPI or RAML, and includes capabilities such as interactive documentation, code generation, and API governance tools. Notably, Stoplight stands out for its strengths in visual API design and its integration with development tools like GitHub and Jira. In the mentioned article author covers the problem occurring during creation and maintaining of OpenAPI standards for REST API testing.

A special tool called Respector was introduced as a first technique to employ static and symbolic program analysis to generate specifications for REST APIs from their source code [31]. Provided experiments showed that Respector successfully detected numerous missing endpoint methods, parameters, constraints, and responses, as well as identified several discrepancies between developer-provided specifications and actual API implementations. Moreover, Respector outperformed other techniques that deduce specifications from API annotations or by invoking the APIs.

With the rise of object-oriented languages and the portability of Java APIs, the development and utilization of reusable software components are becoming increasingly feasible [32]. The effectiveness of component reuse relies heavily on the reliability of these components, which is achieved through comprehensive testing. However, the literature lacks practical approaches for generating inputs and verifying outputs for the numerous test cases required. Author introduces the "Roast" tool and associated techniques for testing Java APIs. The practicality and effectiveness of these methods are demonstrated using two complex components, with quantitative results provided to validate the approach. Each of these papers describes different strengths, whether it's in collaboration, integration, interactive documentation, or API management. Depending on the astronomical project specific needs for astronomical data processing, one of these alternatives might serve as a better fit than Swagger for implementing OpenAPI specifications.

3. Distributed microservices-oriented information system for astronomical data processing

3.1. System design and architecture

Designing a HTTP API service related to astronomical data processing involves creating endpoints that allow clients to interact with and retrieve data about celestial objects, astronomical phenomena, and other relevant information. On the diagram below high-level architecture of designed system is presented. It consists of multiple architectural components including client applications, back-end API aggregators and domain microservices.

Microservices provides multiple communication channels including asynchronous and synchronous ways. Synchronous communication channel is implemented via exposing HTTP APIs for reading data model. Any data model is performed in asynchronous way via message bus (RabbitMQ in a current case). Since data fetching is performed via HTTP request, we can see the importance of OpenAPI and Swagger instrument. Once data is updated via message broker in asynchronous way, Swagger allows to access HTTP REST API in synchronous way by making a direct HTTP call to the microservice accessing the data storage and checking its saved information. The provided diagram in Figure 1 illustrates a high-level architecture for an astronomy-related system using microservices.

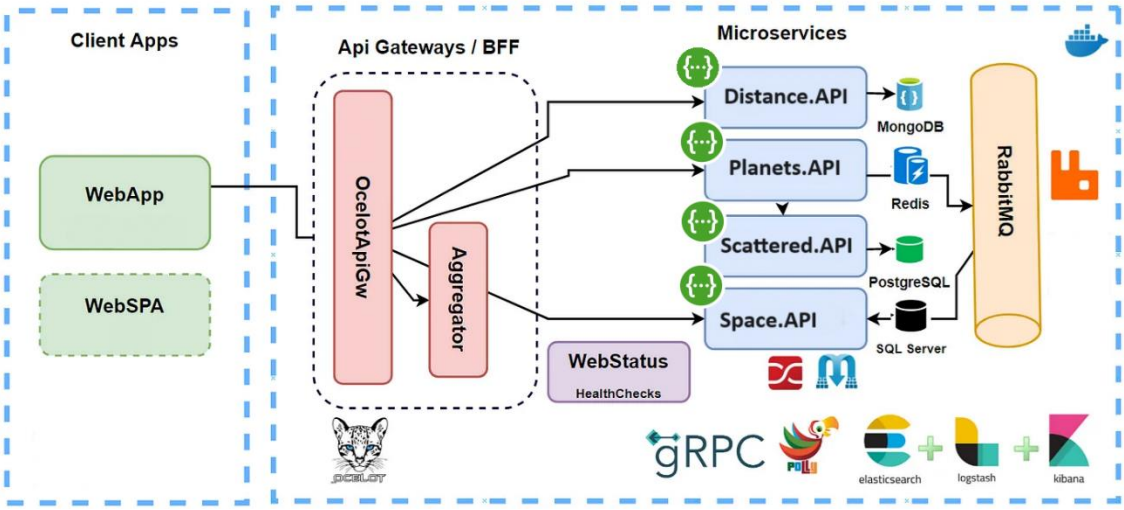

Figure 1: High-level architecture for an astronomy-related system using microservices

Here's a detailed description of each component and their interactions:

1. Client Apps:

    a. WebApp: A traditional web application interface that interacts with the API Gateway.
    b. WebSPA: A Single Page Application (SPA) that provides a more dynamic user experience, also interacting with the API Gateway.
2. API Gateways / BFF (Backend for Frontend):
    a. OcelotApiGw: Ocelot is an API Gateway that handles requests from client applications and routes them to the appropriate microservices. It provides functionalities like authentication, authorization, request aggregation, and more.
    b. Aggregator: This component aggregates data from multiple microservices into a single response, optimizing the number of calls needed by client applications.
3. Microservices:
    a. Distance.API: Handles operations related to astronomical distances. It uses MongoDB for data storage, providing flexible and scalable storage of distance data.
    b. Planets.API: Manages data related to planets. It leverages Redis, an in-memory data store, to enhance the speed of data access and caching.
    c. Scattered.API: Likely deals with scattered objects in space such as asteroids or comets. It uses PostgreSQL, a powerful, open-source relational database.
    d. Space.API: Manages general space-related data. It relies on SQL Server, a robust relational database system from Microsoft.
4. RabbitMQ as a message broker used for asynchronous communication between microservices. It enables event-driven architecture, where services can publish and subscribe to events without tight coupling.
5. Additional Components:
    a. WebStatus (HealthChecks): A service that monitors the health status of various microservices, ensuring they are running optimally. It can provide insights into service uptime and performance.
    b. gRPC: A high-performance, open-source RPC framework that can be used for communication between microservices, offering advantages such as language-agnosticism, low latency, and efficient data serialization.
    c. Polly: A .NET resilience and transient-fault-handling library that allows developers to express policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback.
    d. ELK Stack (Elasticsearch, Logstash, Kibana): A set of tools for logging, searching, and visualizing data:
        i. Elasticsearch: A search and analytics engine.
        ii. Logstash: A data processing pipeline that ingests data from multiple sources, transforms it, and then sends it to a stash like Elasticsearch.
        iii. Kibana: A visualization tool used to explore data stored in Elasticsearch, providing graphical representations and dashboards.
6. Data Flow:
    a. Client Interaction: Users interact with the WebApp or WebSPA, which sends requests to the OcelotApiGw.
    b. API Gateway Routing: The API Gateway routes these requests to the appropriate microservice (Distance.API, Planets.API, Scattered.API, Space.API).
    c. Data Aggregation: For complex queries needing data from multiple sources, the Aggregator compiles the necessary information.
    d. Database Operations: Each microservice interacts with its respective database (MongoDB, Redis, PostgreSQL, SQL Server) to perform CRUD operations.
    e. Asynchronous Communication: Microservices communicate asynchronously through RabbitMQ, allowing for scalable and decoupled architecture.
    f. Health Monitoring: The WebStatus service continuously monitors the health of all services.

g. Logging and Visualization: Logs and metrics are collected, processed, and visualized using the ELK Stack, facilitating monitoring and debugging.

This architecture demonstrates a robust and scalable approach to managing an astronomy-related system using microservices, an API Gateway, asynchronous communication, and comprehensive health monitoring and logging capabilities. It leverages modern technologies to ensure high performance, resilience, and maintainability.

As we can mention from the diagram above, Microservices represented by API HTTP services provide OpenAPI documentation by exposing Swagger endpoints.

3.2. OpenAPI specification

The provided OpenAPI specification describes an API for an astronomy-related service with several endpoints for managing and retrieving data about distances, planets, scattered disks, space, and reference stars [33]. Below is a detailed breakdown of each part of the specification.

Specification provided using OpenAPI Version: 3.0.1. Specification is implemented via open-source tool called Swagger UI and examples are provided below using that API tool.

The first section of the specification (*/api/Distance*) is related to astronomical distance measurement. Existing HTTP endpoints accepts HTTP GET and POST request to the service, allowing to enter a record regarding any distance as well as fetch already existing information. The OpenAPI specification can be found below:

- GET: Retrieves a list of distances.
  - Tags: `Distance`
  - Responses:
    - `200`: Success returns an array of `Distance` objects in `text/plain`, `application/json`, or `text/json` formats.
- POST: Creates a new distance entry.
  - Tags: `Distance`
  - Request Body: Accepts a `Distance` object in `application/json`, `text/json`, or `application/*+json` formats.
  - Responses:
    - `200`: Success returns the created `Distance` object.

The Swagger specification (*/api/Planets*) is related to the planets in the universe providing endpoints for accessing all the information including names, ordering and planetary system. Existing contracts allows retrieve existing list of planets and record a new planet entry which has been discovered recently. The OpenAPI specification can be found below:

- GET: Retrieves a list of planets.
  - Tags: `Planets`
  - Responses:
    - `200`: Success returns an array of strings representing planet names in `text/plain`, `application/json`, or `text/json` formats.
- POST: Creates a new planet entry.
  - Tags: `Planets`
  - Request Body: Accepts a string in `application/json`, `text/json`, or `application/*+json` formats.
  - Responses:
    - `200`: Success

The scattered disk is a distant region of the Solar System that extends beyond the orbit of Neptune. It is populated by a group of small icy bodies known as scattered disk objects (SDOs). These objects have highly elliptical orbits that take them far from the Sun at their aphelion (the point in their orbit

farthest from the Sun) and closer to the Sun at their perihelion (the point in their orbit closest to the Sun). Key characteristics of the scattered disk include orbital characteristics, origins, composition, known Objects. Listed characteristics are covered by the OpenAPI specification *(/api/ScatteredDisk)* listed below:

- GET: Retrieves a list of space-related objects.
  - Tags: `Space`
  - Responses:
    - `200`: Success returns an array of strings in `text/plain`, `application/json`, or `text/json` formats.
- POST: Creates a new space entry.
  - Tags: `Space`
  - Request Body: Accepts a string in `application/json`, `text/json`, or `application/*+json` formats.
  - Responses:
    - `200`: Success

The Figure 2 illustrates the effectiveness of Swagger usage in a context of astronomical data processing. On the image below, we can see an example of transforming OpenAPI JSON specification to user-friendly GUI via Swagger tool.

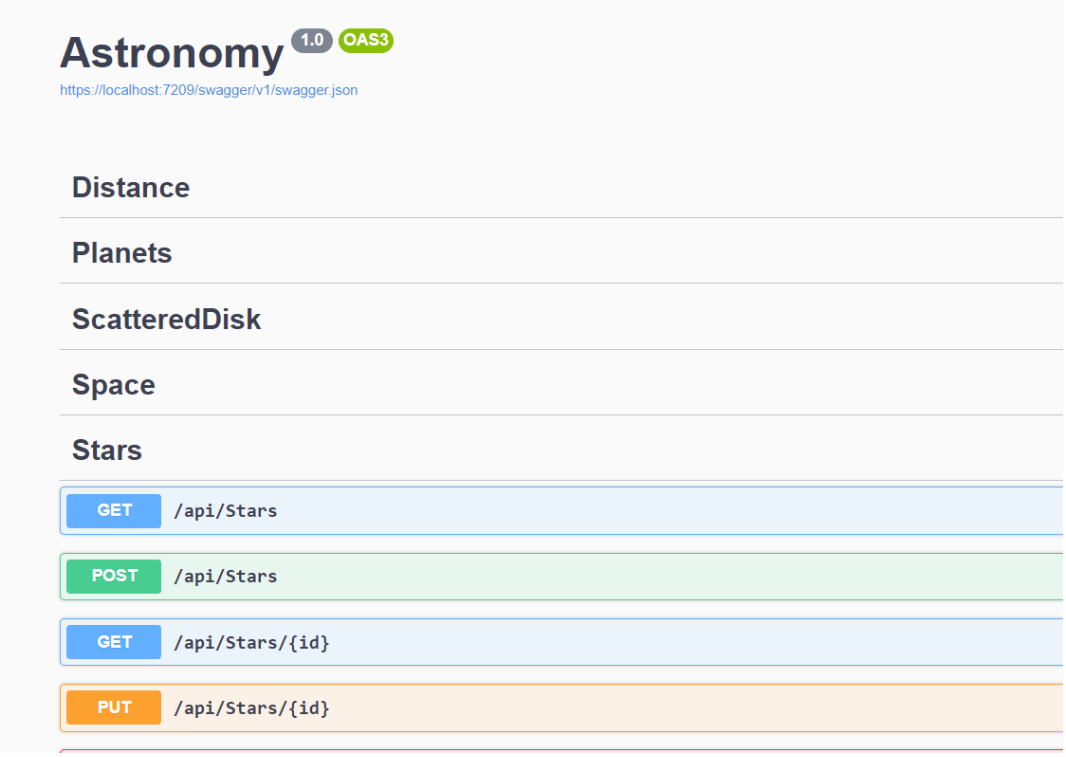

Figure 2: Transforming OpenAPI JSON specification to user-friendly GUI via Swagger tool

3.3. Data Models

Data models define the structure of your data entities in C#. For an astronomy API, these models represent celestial objects and their attributes. The purpose of the following architecture component is to define domain model and main attributes are required during the astronomical data processing. The following important attributes should be defined inside the astronomical domain model of SSOs [34]: mass, radius, identifier (name), etc.

Instances of these models are used throughout your application to represent and manipulate data related to stars. The visual representation of the database models highlights the different structures and technologies used for each microservice. The following diagram in Figure 3 showcases these

models.

The visual representation of the database models highlights the different structures and technologies used for each microservice. The following diagram showcases these models.

- Distance.API (MongoDB): A document collection with various fields for distance data. This collection will store information about the distances between different astronomical objects. Each document will represent a specific distance measurement, including the source and destination of the measurement, the distance value, and the unit of measurement.
- Planets.API (Redis): In-memory data structures for storing planet data. In Redis, each planet will be stored as a hash where the key is a unique identifier for the planet (e.g., planet:1) and the value is a hash containing various attributes of the planet such as name, mass, radius, orbital period, distance from the sun, and atmosphere composition. Redis is used here for its fast read and write operations, which are beneficial for frequently accessed data.
- Scattered.API (PostgreSQL): A relational table with fields for scattered object data. This table will store data about scattered astronomical objects like asteroids and comets. Each row represents an object with attributes including its ID, name, type, mass, radius, orbital period, and discovery date. PostgreSQL is chosen for its ACID compliance and powerful querying capabilities.
- Space.API (SQL Server): A relational table for space entity data with comprehensive fields for detailed information. This table will store general information about various space entities such as stars, galaxies, and nebulas. Each row represents an entity with attributes including its ID, name, type, mass, radius, distance from Earth, and a description. SQL Server is used here for its enterprise features and robust performance.

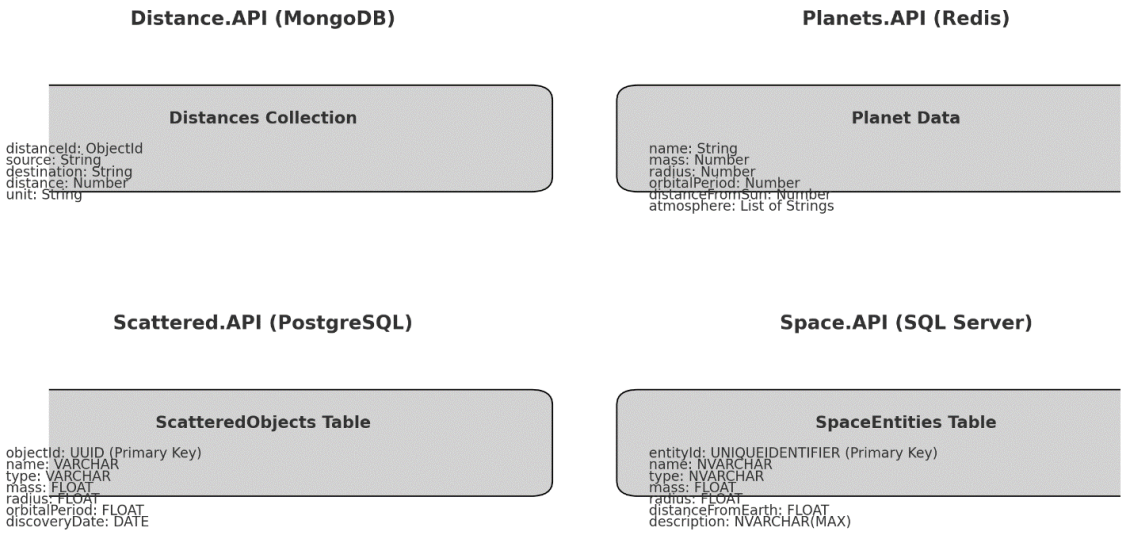

Figure 3: Data models for an astronomy-related system using microservices

Distance JSON model is represented by NoSQL (Document-Oriented) Database. Database collection is called Distances, and it includes following set of fields.

```
{
  "distanceId": "60c72b2f4f1a4e3d5c8b4567",
  "source": "Earth",
  "destination": "Mars",
  "distance": 0.52,
  "unit": "AU"
}
```

Planets JSON model is represented by NoSQL (Key-Value storage) Database. Since it is a key-value storage, data should be stored via single string by hashing or serializing into JSON string.

```
{
  "name": "Earth",
  "mass": 5.972e24,
  "radius": 6371,
  "orbitalPeriod": 365.25,
  "distanceFromSun": 1.00,
  "atmosphere": ["Nitrogen", "Oxygen", "Argon", "Carbon Dioxide"]
}
```

Scattered objects are stored in relational SQL database with its unique identifier as primary key for each scattered object and the list of related attributes.

```
CREATE TABLE ScatteredObjects (
  objectId UUID PRIMARY KEY,
  name VARCHAR(255),
  type VARCHAR(50),
  mass FLOAT,
  radius FLOAT,
  orbitalPeriod FLOAT,
  discoveryDate DATE
);
```

Space SQL model is represented as relational database table as well with corresponding primary attribute and the list of attributes assigned.

```
CREATE TABLE SpaceEntities (
  entityId UNIQUEIDENTIFIER PRIMARY KEY,
  name NVARCHAR(255),
  type NVARCHAR(50),
  mass FLOAT,
  radius FLOAT,
  distanceFromEarth FLOAT,
  description NVARCHAR(MAX)
);
```

This architecture leverages the strengths of each database technology, ensuring optimal performance, scalability, and flexibility for handling diverse data requirements in an astronomy-related HTTP service.

4. Results

A skeleton of the proposed OpenAPI specification in distributed microservices-oriented information system for astronomical data processing was tested in scope of the Lemur software of the Collection Light Technology (CoLiTec) project (https://colitec.space) [35]. The specific modules and services related to the mathematical methods and algorithms in the Lemur software are:

- automated frame calibration;
- cosmetic frame correction;
- track-and-stack feature;
- brightness equalization;
- background alignment;
- astronomical image filtering;
- determining the contours of objects;
- fully automated robust method of the astrometric reduction;
- fully automated robust method of the photometric reduction [36];

- support of the multi-threaded processing;
- transferring of astronomical data with intermediate storage;

More extended details about the Lemur software of the CoLiTec project are presented in these papers and research [37, 38, 39]. The example of JSON data implemented in scope of the distributed microservices-oriented information system for astronomical data processing for the Lemur software represents a response for an API that provides distance-related information between celestial bodies mentioned below.

```
{
  "origin": {
    "name": "Earth",
    "type": "Planet"
  },
  "destination": {
    "name": "Mars",
    "type": "Planet"
  },
  "distance": {
    "unit": "AU",
    "value": 1.52
  },
  "travelTime": {
    "unit": "days",
    "value": 300
  },
  "metadata": {
    "requestTime": "2024-06-18T12:34:56Z",
    "responseTime": "2024-06-18T12:34:57Z",
    "service": "DistanceAPI"
  }
}
```

Presented JSON structure contains following valued information:

- origin: Information about the starting point of the distance calculation, including the name, type (e.g., planet, star).
- destination: Information about the endpoint of the distance calculation, similar to the origin.
- distance: The calculated distance between the origin and destination, along with the unit of measurement (e.g., Astronomical Units - AU).
- travelTime: An estimated travel time to cover the distance, along with the unit of measurement (e.g., days).
- metadata: Additional information about the API request, including the request and response times and the name of the service that provided the data.

This JSON structure is designed to be comprehensive and can be extended further based on the specific requirements and additional attributes that might be relevant for the Distance API in a microservices architecture.

5. Conclusions

We presented the usage of OpenAPI specification in distributed microservices-oriented information system for astronomical data processing. A common goal of all scientific and technological algorithms and methods is to automate as much as available processes without any human actions.

In general cases it can be done by the different astronomical distributed microservices-oriented information system. In these pipelines the various data mining and knowledge discovery in databases

tasks are used for speeding up and optimizing the astronomical data processing. Suggested using of the OpenAPI specification in a distributed microservices-oriented information system for astronomical data processing significantly improves the system's interoperability, scalability, and maintainability. The developed skeleton of the real example of astronomical data-processing system is implemented using .Net Core framework and C# programming language. The modern international astronomical astrometric and photometric catalogues are available now in cloud, so any interactions with such data from them require services integration for processing. In comparison with existed information systems related to the astronomical image processing the developed micro-serviced architecture show more stability and scalability points. Such architecture will be also very helpful in the complex information systems for astronomical data processing with integration of the Continuous Integration/Continuous Delivery (CI/CD) principles.

The further research will be conducted on integrating proposed OpenAPI specification in distributed microservices-oriented information system for astronomical data processing in scope of the Lemur software of the Collection Light Technology (CoLiTec) project.

References

[1] L. Wheeler, et al., Risk assessment for asteroid impact threat scenarios, Acta Astronautica 216 (2014) 468–487. doi: 10.1016/j.actaastro.2023.12.049.

[2] V. Akhmetov, et al., Cloud computing analysis of Indian ASAT test on March 27, 2019, in: International Scientific-Practical Conference: Problems of Infocommunications Science and Technology, 2019, pp. 315–318. doi: 10.1109/PICST47496.2019.9061243.

[3] V. Savanevych, et al., Mathematical methods for an accurate navigation of the robotic telescopes, Mathematics 11 10 (2023) 2246. doi: 10.3390/math11102246.

[4] R. Klette, Concise computer vision. An Introduction into Theory and Algorithms, London: Springer, 2014.

[5] V. Vlasenko, et al., Devising a procedure for the brightness alignment of astronomical frames background by a high frequency filtration to improve accuracy of the brightness estimation of objects, Eastern-European Journal of Enterprise Technologies 2 2 (2024) 31–38. doi: 10.15587/1729-4061.2024.301327.

[6] Š. Parimucha, et al., CoLiTecVS – A new tool for an automated reduction of photometric observations, Contributions of the Astronomical Observatory Skalnate Pleso 49 2 (2019) 151-153.

[7] V. Akhmetov, et al., Astrometric reduction of the wide-field images, Advances in Intelligent Systems and Computing, 1080 (2020) 896–909. doi: 10.1007/978-3-030-33695-0_58.

[8] V. Kudak, V. Epishev, V. Perig, I. Neybauer, Determining the orientation and spin period of TOPEX/Poseidon satellite by a photometric method, Astrophysical Bulletin 72 3 (2017) 340-348. doi: 10.1134/S1990341317030233.

[9] S. Khlamov, et al., Big astronomical datasets and discovery of new celestial bodies in the Solar System in automated mode by the CoLiTec software, Knowledge Discovery in Big Data from Astronomy and Earth Observation, Astrogeoinformatics (2020) 331–345. doi: 10.1016/B978-0-12-819154-5.00030-8.

[10] V. Troianskyi, V. Godunova, A. Serebryanskiy, et al., Optical observations of the potentially hazardous asteroid (4660) Nereus at opposition 2021, Icarus 420 (2024) 116146. doi: 10.1016/j.icarus.2024.116146.

[11] G. Adam, P. Kontaxis, L. Doulos, E.-N. Madias, C. Bouroussis, F. Topalis, Embedded Microcontroller with a CCD Camera as a Digital Lighting Control System, Electronics 8 1 (2019). doi: 10.3390/electronics8010033.

[12] D. Oszkiewicz, et al., Spins and shapes of basaltic asteroids and the missing mantle problem, Icarus 397 (2023) 115520. doi: 10.1016/j.icarus.2023.115520.

[13] I. B. Vavilova, Y. S. Yatskiv, L. K. Pakuliak, et al., UkrVO astroinformatics software and web-services, in: Proceedings of the International Astronomical Union 12 S325 (2016) 361-366. doi: 10.1017/S1743921317001661.

[14] V. Akhmetov, et al., Fast coordinate cross-match tool for large astronomical catalogue, Advances in Intelligent Systems and Computing 871 (2019) 3–16.

[15] K. D. Borne, Scientific data mining in astronomy, Next Generation of Data Mining, Chapman and Hall/CRC, 2008.
[16] Y. Zhang, Y. Zhao, Astronomy in the big data era, Data Science Journal 14 (2015).
[17] N. Raychev, Test Automation in Microservice Architecture, IEEE Spectrum 8 7 (2020) 15 p.
[18] R. Huang, M. Motwani, I. Martinez, A. Orso, Generating REST API Specifications through Static Analysis, in: IEEE ACM 46th International Conference On Software Engineering (ICSE), 107, 2024, pp. 1–13. doi: 10.1145/3597503.3639137.
[19] V. Savanevych, et al., Formation of a typical form of an object image in a series of digital frames, Eastern-European Journal of Enterprise Technologies 6 2 (2022) 51–59. doi: 10.15587/1729-4061.2022.266988.
[20] W. Burger, M. Burge, Principles of digital image processing: fundamental techniques, New York, NY: Springer, 2009.
[21] S. Khlamov, I. Tabakova, T. Trunova, Z. Deineko, Machine Vision for Astronomical Images Using the Canny Edge Detector, CEUR Workshop Proceedings, vol. 3384, pp. 1–10, 2022.
[22] V. Troianskyi, P. Kankiewicz, D. Oszkiewicz, Dynamical evolution of basaltic asteroids outside the Vesta family in the inner main belt, Astronomy and Astrophysics 672 (2023). doi: 10.1051/0004-6361/202245678.
[23] D. Oszkiewicz, et al., Spin rates of V-type asteroids, Astronomy & Astrophysics 643 (2020). doi: 10.1051/0004-6361/202038062.
[24] V. Savanevych, et al., CoLiTecVS software for the automated reduction of photometric observations in CCD-frames, Astronomy and Computing 40 (2022) 100605.
[25] S. Khlamov, et al., Development of the matched filtration of a blurred digital image using its typical form, Eastern-European Journal of Enterprise Technologies 1 9 (2023) 62–71. doi: 10.15587/1729-4061.2023.273674.
[26] M. Dadkhah, et al., Methodology of wavelet analysis in research of dynamics of phishing attacks, International Journal of Advanced Intelligence Paradigms 12 (2019) 220-238. doi: 10.1504/IJAIP.2019.098561.
[27] L. Kirichenko, A.S.A. Alghawli, T. Radivilova, Generalized approach to analysis of multifractal properties from short time series, International Journal of Advanced Computer Science and Applications 11 5 (2020) 183–198. doi: 10.14569/IJACSA.2020.0110527.
[28] S. Khlamov, et al., The astronomical object recognition and its near-zero motion detection in series of images by in situ modeling, in: Proceedings of the 29th IEEE International Conference on Systems, Signals, and Image Processing, IWSSIP 2022, Sofia, Bulgaria, June 1st – 3rd, 2022. doi: 10.1109/IWSSIP55020.2022.9854475.
[29] D.N.N. da Costa, Guidelines for Testing Microservice-based Applications, Dissertation for obtaining the master's degree, 2022.
[30] Stoplight. Available at: https://stoplight.io/solutions.
[31] A. Golmohammadi, M. Zhang, A. Arcuri, Testing Restful Apis: A Survey, ACM Transactions on Software Engineering and Methodology 33 (2023) 1-41.
[32] D. Hoffman, P. Strooper, Tools and Techniques for Java API Testing, in: Australian Software Engineering Conference, 2002, pp. 235-245. doi: 10.1109/ASWEC.2000.844580.
[33] S. Khlamov, et al., Automated data mining of the reference stars from astronomical CCD frames, CEUR Workshop Proceedings, vol. 3668, 2024, pp. 83–97.
[34] V. Troianskyi, V. Kashuba, O. Bazyey, et al., First reported observation of asteroids 2017 AB8, 2017 QX33, and 2017 RV12, Contributions of the Astronomical Observatory Skalnaté Pleso 53 (2023) 5-15. doi: 10.31577/caosp.2023.53.2.5.
[35] S. Khlamov, et al., Machine vision for astronomical images using the modern image processing algorithms implemented in the CoLiTec software, Measurements and Instrumentation for Machine Vision, Chapter 12: CRC Press, Taylor & Francis Group, 2024, pp. 269-310. doi: 10.1201/9781003343783-12.
[36] I. Kudzej, et al., CoLiTecVS – A new tool for the automated reduction of photometric observations, Astronomische Nachrichten 340 (2019) 68–70. doi: 10.1002/asna.201913562.
[37] V. Savanevych, et al., Comparative analysis of the positional accuracy of CCD measurements of small bodies in the solar system software CoLiTec and Astrometrica, Kinematics and Physics of Celestial Bodies 31 6 (2015) 302–313.

[38] V. Savanevych, et al., A new method based on the subpixel Gaussian model for accurate estimation of asteroid coordinates, MNRAS 451 3 (2015) 3287–3298.
[39] V. Savanevych, et al., A method of immediate detection of objects with a near-zero apparent motion in series of CCD-frames, A&A 609, 2018.