Designing a microservice architecture on the example of an automated application for the HR department

Natalya Kolyeva, Liudmila Kortenko, and Elena Radkovskaya

1 Ural State University of Economics, 620144 Yekaterinburg, Russia

Abstract. This article discusses the use of microservices in the context of HR processes and proposes a specific architecture to achieve this goal. The article is of interest to everyone who is interested in improving processes using the latest technologies.

1 Introduction

Microservice architecture is one of the popular approaches to designing modern applications, including automated applications. This approach is to break the application into small, autonomous and interacting services. When designing, it is important to define the boundaries of services: which functions should be separated into separate services, and which can be combined. Service boundaries should be clearly defined to avoid too many dependencies between them.

A microservice is an architectural approach in which an application is broken down into small, loosely coupled and independent units (services) that perform separate functions. Each microservice can have its own data set, storage, business logic, API and UI. A client can be any software that requests services or data from a server (principle of client-server architecture). This can be a web browser, mobile application, desktop application, game console, payment terminal, and many other devices and software products. The main requirement is the ability to initiate a request to the server and process the response that the server will return. The client usually has a graphical interface for user interaction and provides a convenient way to access functionality and resources that are stored on the server. The number of clients in a client-server system can be different and depends on the specific implementation and system requirements.

A human resources department that does not use a process automation system may encounter problems such as:

1. Low speed of document processing. The system allows you to quickly process documents, monitor their movement, notify of the need for execution and control execution. This makes it possible to avoid delays in the process of work and reduce the number of errors.
2. Long search for documents. The system stores all documents in a centralized electronic archive and allows you to quickly find the desired document using filters by various parameters.

3. Difficulty in monitoring the execution of documents. The system allows you to control who and when executes the document, which reduces the likelihood of errors and increases the responsibility of performers.

4. Interactions between departments. The system allows you to simplify and speed up the transfer of information and documents between departments, which helps to increase the efficiency of the entire company.

From the listed problems, we single out two main needs of the personnel department, which the system should solve:

5. Control the movement and execution of the document.


To control the movement of a document, you need to determine which stages the document goes through and what actions are available at each stage.

In each business process of the HR department, a common scenario for document movement is highlighted:

7. Entrance/Exit of a business process (creation/registration of a document).


10. Business process results/indicators (statuses, date of creation, date of change).

11. Related documents.

2 Research results
request to another microservice via HTTP commands such as GET, POST or PUT. In response, it receives the status code of the sent request. Consider the interaction scenarios that must be performed in the application.

The client sends a request to create a document to idoc-gateway-api and receives a response with an HTTP_OK (200) status. The request was successfully processed.

Gateway-api routes the request to the idoc-events-adapter-api service, which acts as a producer. It processes the request and outputs the CreateDocumentEvent event to the queue (message broker).

Idoc-documents-event-bus-api acting as a consumer polls the message broker for new events. Upon receiving the event, it generates a request to create a document in idoc-documents-api.

Figure 1 shows a sequential diagram of the interaction of microservices when creating a document.

**Fig. 1.** Sequential scheme of interaction of microservices when creating a document

Document update.

The UpdateDocumentEvent event is passed to idoc-documents-event-bus-api in the same way as CreateDocumentEvent.

When the executor of a document changes, all its participants (author, current executor, and all previous executors) receive an event notification. Therefore, it is necessary to obtain the data that has been changed and determine whether there is an executor among them. To do this, idoc-documents-event-bus-api first sends a GET request to /api/documents{id} and gets the version of the document before the change. Next, a PUT /api/documents{id} request changes the document data. It is important to note that in this case it is correct to use PUT, because it is idempotent and multiple calls to this method, with the same set of data, will have the same execution result (no side effects). After the request is successfully completed, the document data is compared before and after the change.

If the performer has not been changed, then the interaction of microservices ends. If the executor has been changed, then the author of the document and all previous executors (including the current one) should receive an email notification.

GET request /api/documents/{id}/revisions, returns all previous states of the document including id of users who were previously executors. A GET request to /api/users/{id} for the idoc-users-api service returns user information that stores email. Finally, idoc-documents-event-bus-api sends a POST request to idoc-notification-api to send notifications to users whose emails were specified in the profile.


3 Conclusion

This approach to application design makes it easy to scale and change the application, since each service can be changed and scaled independently of other services. This simplifies the process of developing and maintaining the application.

References

5. T. Pyatkova, N. Kolyeva, M. Panova, E3S Web of Conferences 376, 05041 (2023) E3S Web of Conferences https://doi.org/10.1051/e3sconf/202337605041
7. J. Streimikis, L. Kortenko, M. Panova, M. Voronov, E3S Web of Conferences 301, 05002 (2021)