info@apriorit.com
www.apriorit.com

Case study

Building a Microservices SaaS Solution for Property Management

Our client is a large US-based provider of property management software who had a monolithic SaaS platform written in Ruby. To attract new customers, optimize platform maintenance costs, and make it easier to implement new features, they decided to improve the platform design and replace their monolithic solution with a flexible solution based on microservices.

As a result of our cooperation, the dedicated Apriorit team prepared a new design and a microservices architecture for the product. Then our specialists implemented the necessary components of the new platform and planned its expansion.

The client

Our client is an American provider of SaaS property management solutions. The company offers a wide selection of cloud tools and services to comfortably manage residential and commercial properties. The tools and services provided cover automated accounting, billing, fee collection, and tenant screening.

The challenge

The client wanted to make their platform more flexible, resilient, and comfortable to work with. To achieve this, they decided to update the platform's design and replace the current monolithic platform with a platform based on microservices.

At first, the client attempted to prototype and design the new solution on their own. However, their internal development team lacked the needed skills and experience for SaaS and microservices development. In particular, they needed competent Angular and Golang developers as well as a DevOps specialist for configuring the AWS infrastructure.

As Apriorit has extensive expertise working with these languages as well as building microservices SaaS solutions, the client challenged us with this task.

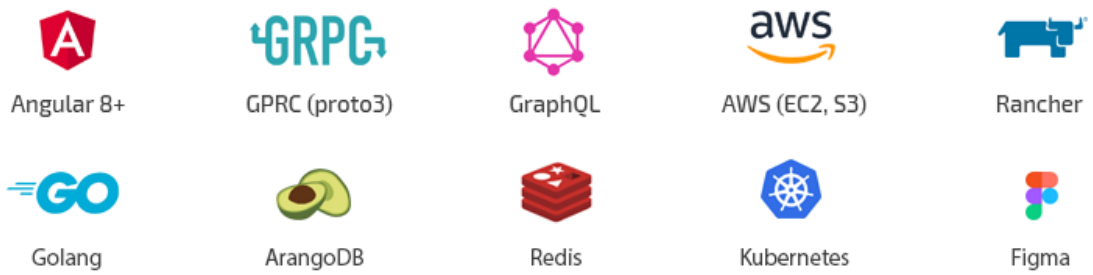
Our approach

Apriorit formed a dedicated team that consisted of a business analyst (BA), a UI/UX designer, developers, a DevOps specialist, and a project manager. Our team was supposed to work alongside the client's developers, so we established a clear communication workflow that helped us eliminate misunderstandings between the two teams.

After discussing the client's vision of the new platform's operation, we finalized the requirements for the end product, planned a new microservices-based architecture for the platform, and moved to the implementation stage.

APRIORIT TEAM

KEY TECHNOLOGIES USED FOR THE PROJECT



The result

We successfully implemented a microservices-based SaaS platform with an updated design. The new platform offers better scalability, easier code maintenance, and a more enjoyable user experience.

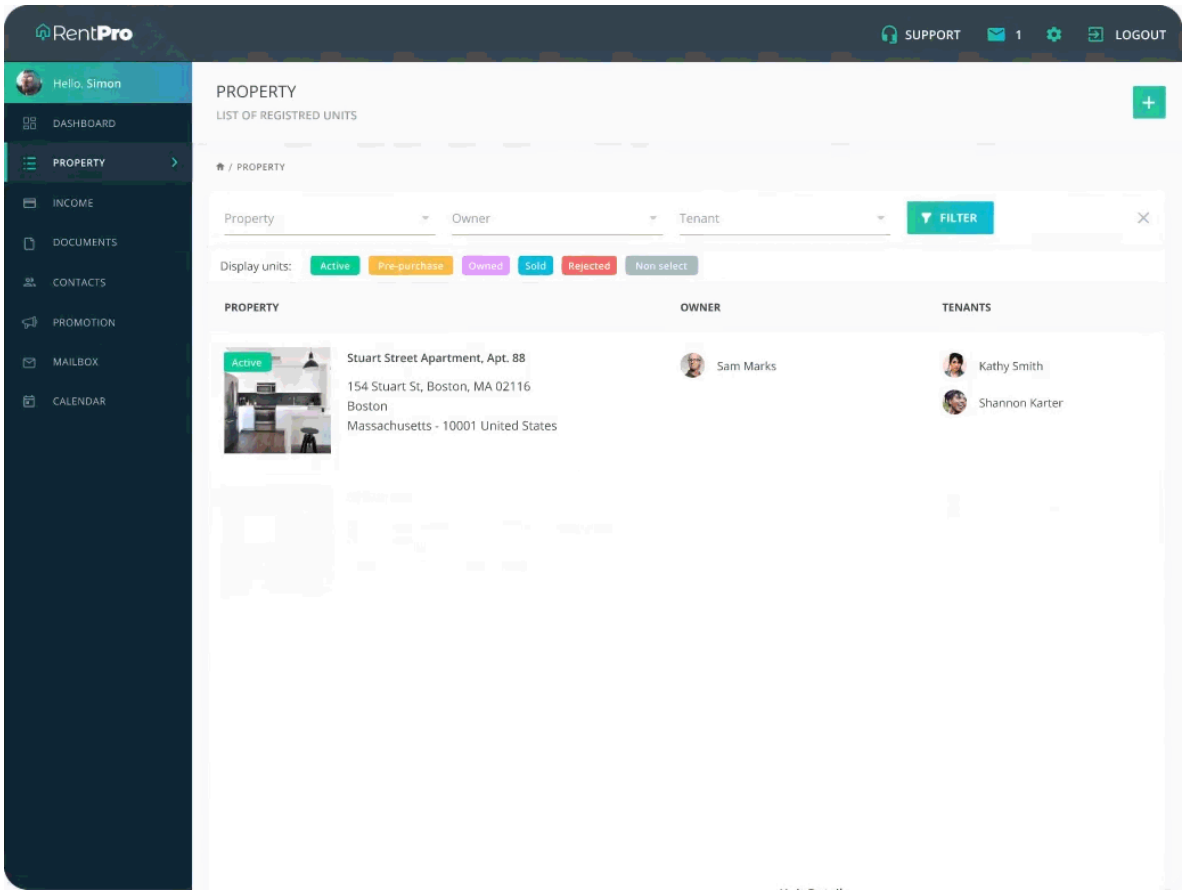
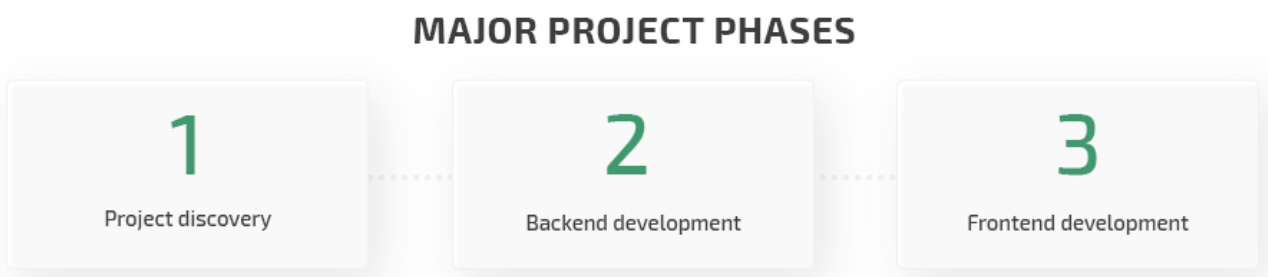


Figure 1. Property management view

Our approach step by step

The project has gone through three key phases so far:



Let’s go over each of these phases.

Phase 1: Project discovery

In order for our team to successfully realize the client's vision of the final product, we started our work with project discovery. This consisted of four key stages:

The client provided our business analyst with a large presentation containing their preliminary product requirements. After processing these requirements, our BA started preparing a set of user stories — comprehensive descriptions of features and functionalities. All documentation was approved with the client's representative.

At the same time, our UI/UX designer started working on mockups for the new look of the client's platform.

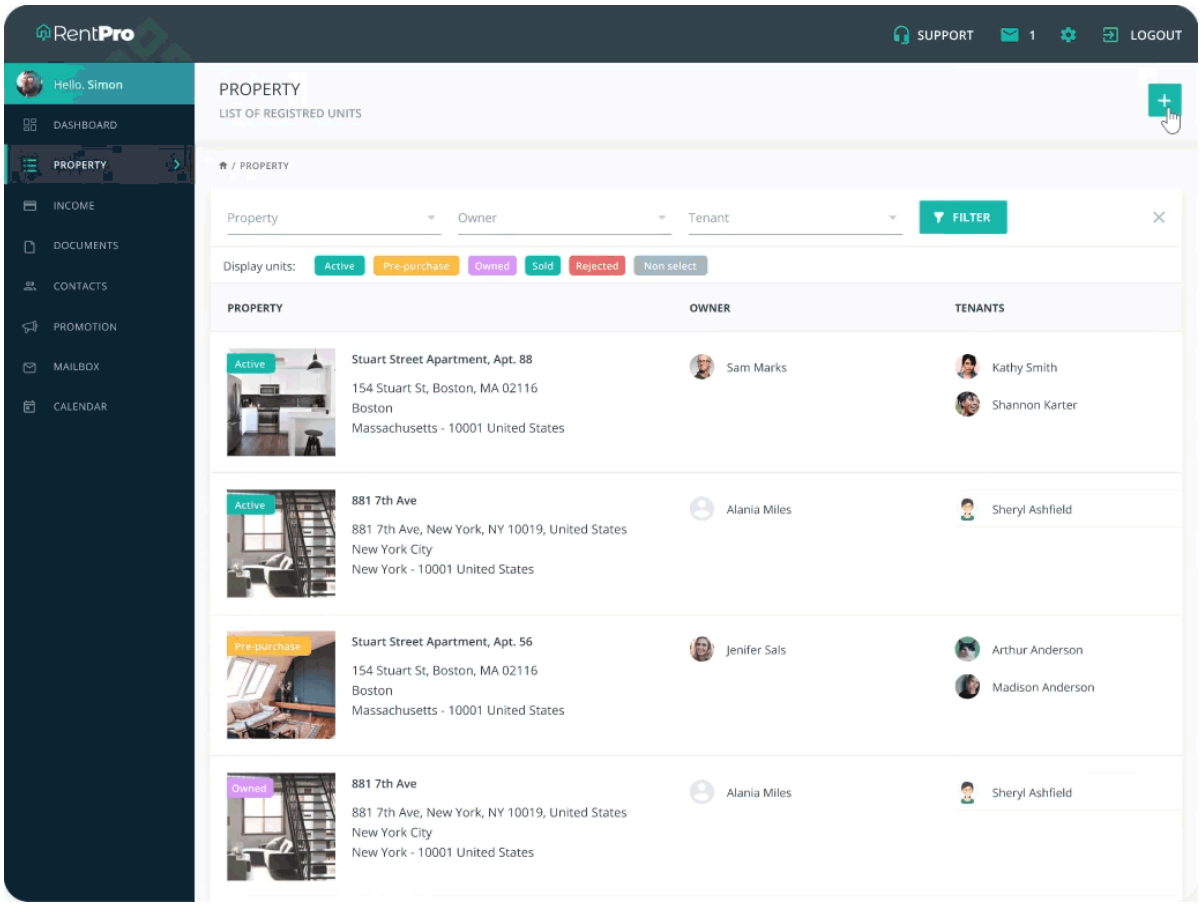


Figure 2. Property units view

The client wanted to follow Material Design principles to create truly functional visuals. Similarly to finalized requirements and user stories, all design improvements were regularly reviewed and approved by the client.

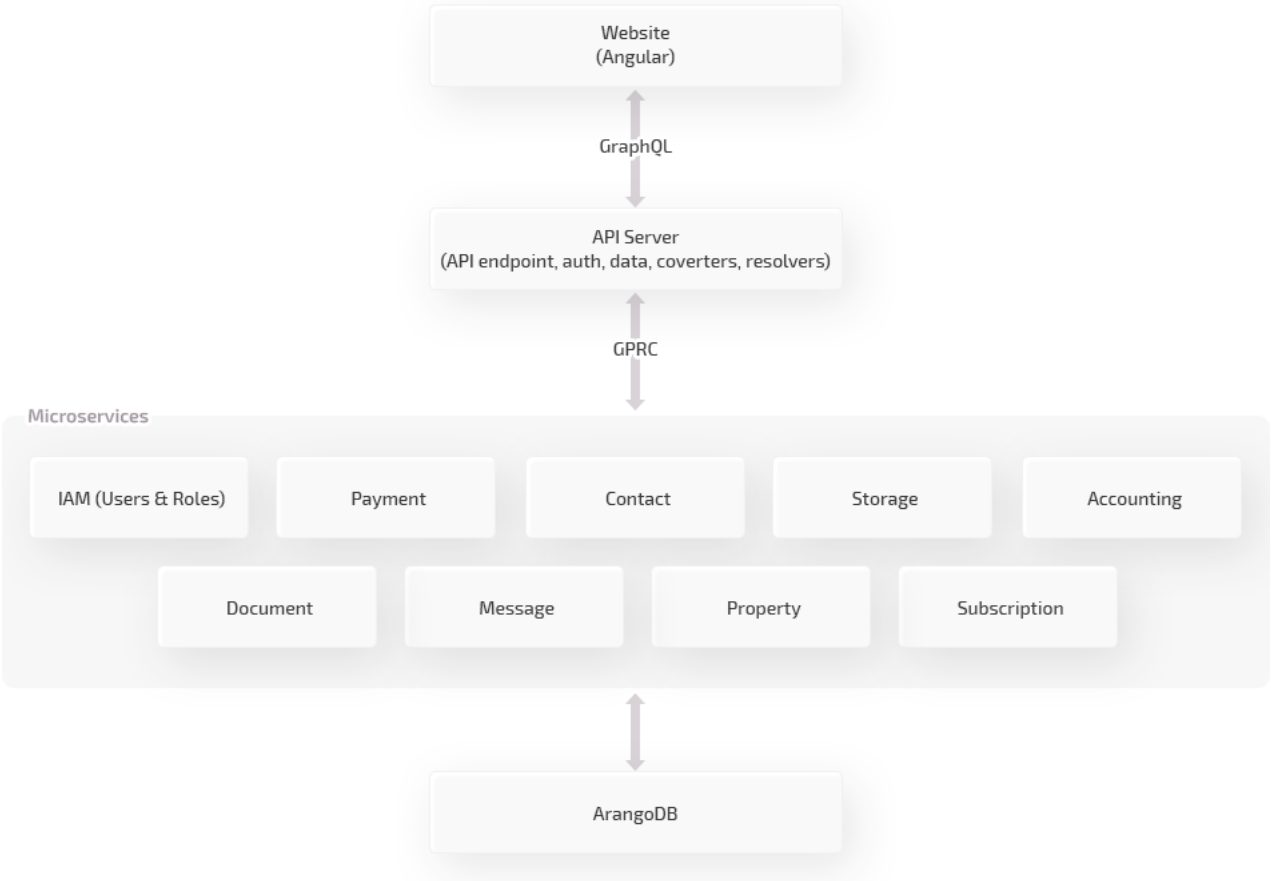
To make the process of collaborating on design changes more efficient — both within the team and with the client — we used the Figma UI tool.Work on new user stories and design variations based on them continued during the following project phases.

The third stage of the project discovery phase was creating the new microservices architecture. First, our backend developer analyzed all data processed by the current monolithic system to outline features and functions that could be grouped into separate microservices. At first, we outlined six key blocks that could be used as the basis for the new microservices architecture:

- Accounting
- Contact
- Identity and access management (IAM)
- Document
- Message
- Property

Later, we added three more blocks: Payment, Storage, and Subscription. The final version of the created SaaS solution microservices architecture looked like this:

Microservices architecture of the SaaS property management solution



We also paid special attention to establishing transparent and efficient communication with the client. Every week, the client sees practical demonstrations of our progress during Zoom calls and receives detailed written reports via email. Daily communications are organized through meetings and GitLab pull requests. Additionally, as the client uses the Dedicated Team outsourcing model, their CTO can communicate directly with our developers when needed.

Once the client's CTO approved the preliminary architectural model, we started planning our first Agile sprint and finally moved to the actual development. The next phase was creating the product's back end.

Phase 2: Starting backend development

During this phase, our team focused on four main tasks:

1. Ensure proper database integration
2. Create models for connecting microservices
3. Create GraphQL models for communication with the platform front end
4. Implement the first set of microservices

We used ArangoDB as the core database for the new platform. For testing the services that communicated with the database, we decided to use integration tests that would set up ArangoDB in a container using the Testcontainers package.

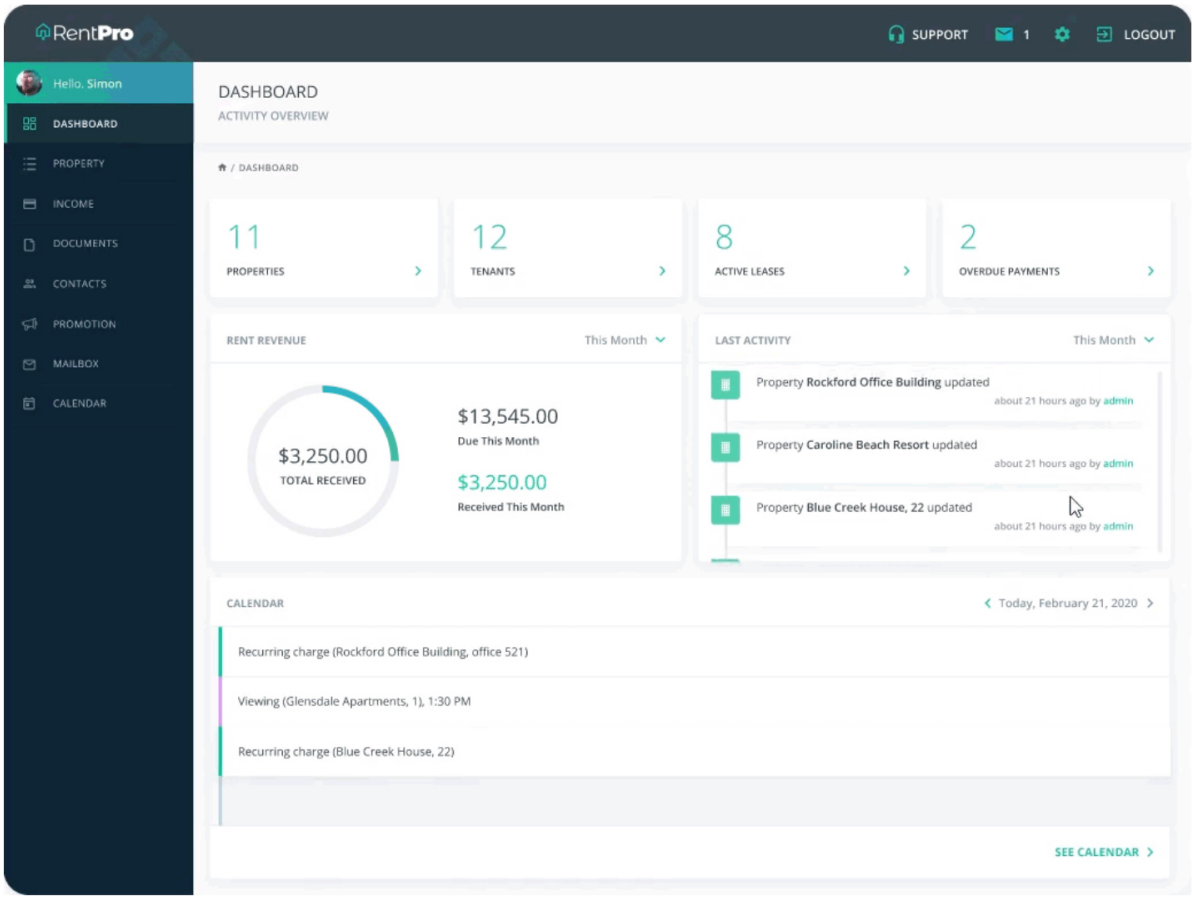


Figure 3. Dashboard view

To connect microservices and establish flawless communication between them, we used the gRPC framework. And for communication with the front end, we built GraphQL models in the platform's API server.

Finally, we started implementing separate services that will later form the new platform. The first set of developed services included:

- A service for creating and storing property data
- A user authorization service
- An IAM service for storing user data and roles
- A service for integration with the payment system

Regular unit and integration tests allowed us to ensure the proper quality of the developed features. After building the solution's backend core, we finally moved to developing the front end.

Phase 3: Starting frontend development

In close cooperation with our business analyst and designer, the frontend developer started building the client side of the application's web interface.

We also continued building new microservices for the platform. In particular, our team focused on creating the following services:

- Contact service for storing user contacts
- Message exchange service
- Subscription service
- Document service for storing data on lease agreements

At first, we planned separate scopes of work for the frontend and backend developers. However, as we progressed with the backend–frontend integration, it became obvious that this approach wasn't efficient enough. As frontend and backend developers were focused on different features, we couldn't detect possible flaws in the code or address integration issues early on.

This is why we decided to revise our plans and switch from implementing separate functionalities to working on full user stories from different angles. In this way, the whole project team could work on the same scope of features during the same sprint. This allowed us to ensure fast detection and resolution of any integration issues and release a complete feature according to the user story.

Currently, we're working on implementing the front end. Our plans for the near future include developing new features and services, including:

- Tenant screening
- Automated accounting and billing
- Financial report generation
- Property portfolios
- User documentation library
- Webflyers

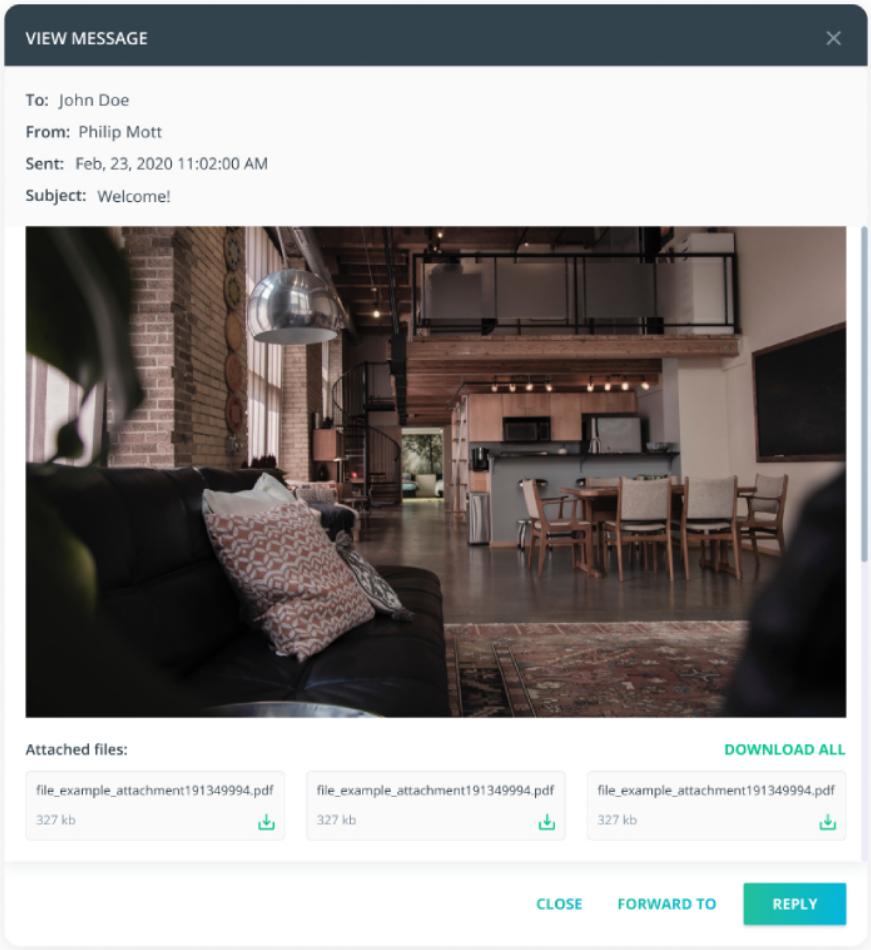


Figure 4. Message view

Responding to arising project issues

While working on this project, we faced the following issues:

Environment setup — While the client didn't request a DevOps specialist at the beginning of the project, we found ourselves needing to employ one when it came to setting up the environment for the new platform.

Luckily, Apriorit has internal DevOps specialists, so after getting the client's approval, we were able to engage an expert with all the needed competences. Working with an internal DevOps specialist allowed us to ensure timely environment setup without delays and bottlenecks.

Parallel work on features — The project had strict deadlines, which is why we sometimes had to prepare feature specifications and design mockups and develop the actual feature almost in parallel. This made it difficult to keep team members up to date with the progress and make sure that any last-minute changes didn't contradict previously agreed requirements.

Having a highly competent BA and project manager on our team helped us improve our communication process and maintain a well-organized workflow.

Communication adjustments — At the beginning of the project, we used a common

communication model based on daily and weekly meetings. However, this approach turned out to be ineffective for tracking the progress and current state of a specific feature or functionality.

After discussing different options, we settled on using a shared table where specialists from both teams would mark status updates for every feature.

The Apriorit team managed to effectively solve these issues and provide the client with satisfactory results.

The impact

Our client's monolithic SaaS platform presents many challenges including increasingly complex legacy code maintenance and high interdependence of system features. All this negatively affects not only the performance and stability of the platform but also the experience of its end users.

While the new platform is still in development, it's already obvious that using microservices to customize this SaaS solution will significantly improve the platform's scalability. The use of microservices also increases the flexibility of the platform's architecture and simplifies the process of adding new features to it.

Finally, the modernized design that's both more appealing and more functional is expected to significantly improve the user experience. As we continue working on new features that will boost the platform's performance, both our client and the Apriorit team are excited about the future of this project.

Want to build a highly efficient SaaS solution based on microservices?
Dedicate this task to Apriorit!