Qulix

How to move your Mobile Banking App into micro frontend?

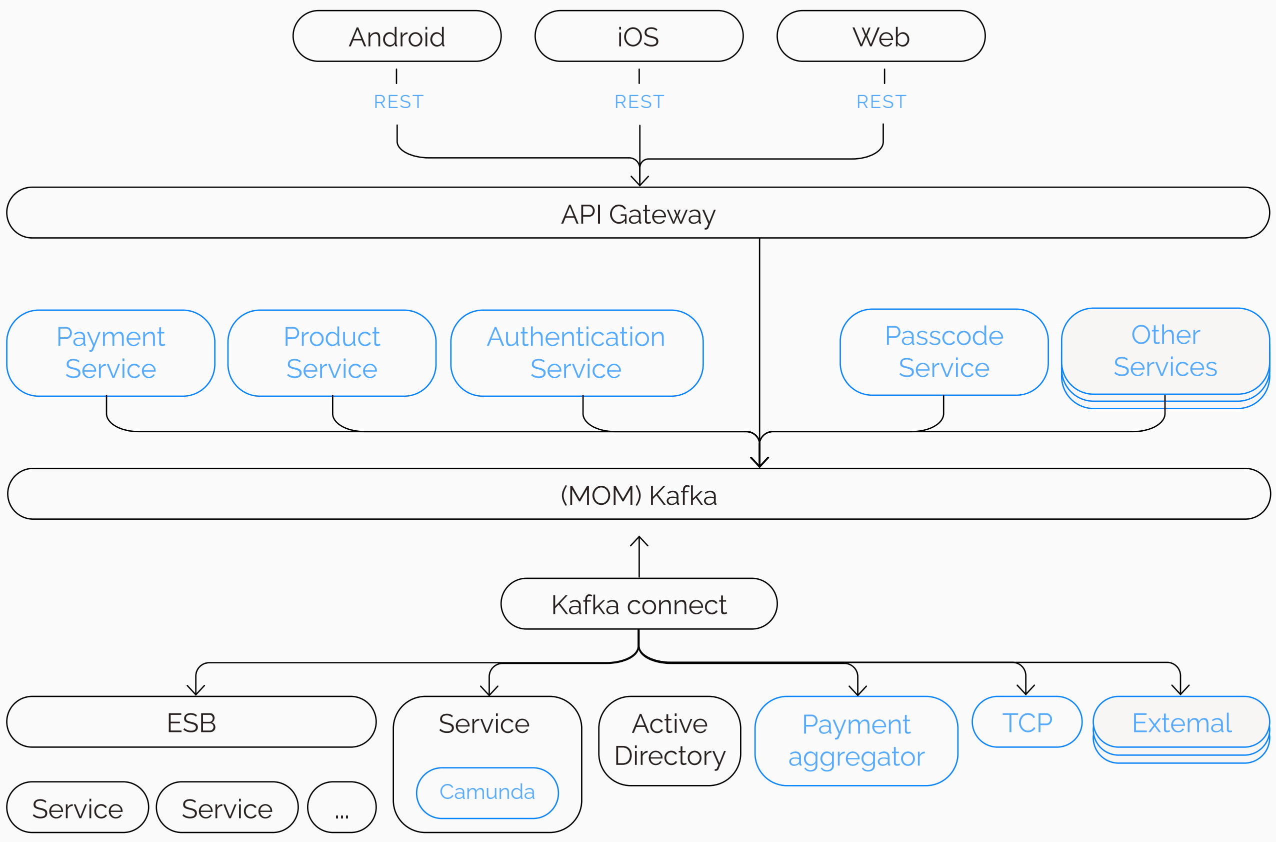
qulix.com

Disclaimer

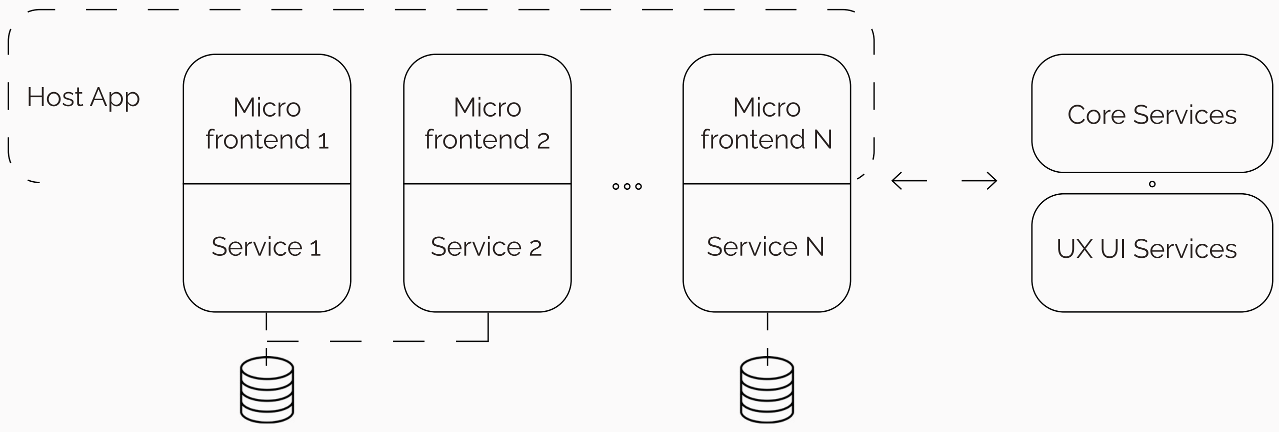
Table of Contents

Introduction

Microservices have been gaining a huge momentum recently in the banking sector. It comes as no surprise in 2021 when a well-established bank decides to migrate its application to an architecture like the one below:



However, in an ideal situation the frontend should be decoupled, too. Thus, the above architecture can be revised to include *micro frontends* (see below):



If building micro frontend-based web applications is a tough, yet a doable project, a mobile app with a micro frontend architecture is way more difficult to build.

In this whitepaper we will tell you how to implement a micro frontend approach in your mobile banking application.

N.B. Mind please, that the notion of micro frontends refers mostly to web development (client applications). With mobile applications, we use *plugin/modular architecture* implying what in web development is known as micro frontends. This is due to the implementation differences, while the concept (both in micro frontends and plugin/modular architecture) is the same - the monolith is decoupled into a set of autonomous parts and the application is delivered by a number of independent feature-teams.

This whitepaper addresses the following issues that you might face during your micro frontend project:

1. Why should banks move to the micro frontend architecture? Is it the right solution for you?
2. What is the best approach for a seamless migration to micro frontends?
3. What steps should the Bank's IT team take to migrate from a monolithic to micro frontend architecture?
4. What IT team composition will suit better for the migration needs?

Thus, we highlight all the typical questions of a Bank willing to migrate to a new, modular architecture, with both backend and frontend sides decoupled.

What are micro frontends and why do banks need them?

This Section provides an overview of the topic. It contains basic definitions, problem statement and a high-level description of the solution.

1. What are micro frontends?

The term *microservices* has been known for over a decade by now. It's widely used in server-side development nowadays. For frontend development, *micro frontends* are the alternative to the old-school monolithic architecture.

Martin Fowler gives the following high-level definition of micro frontends:

> *An architectural style where independently deliverable frontend applications are composed into a greater wholeю.*

Thus, similarly to microservices, in micro frontends

1. Application is delivered by a number of autonomous teams
2. Small and relatively independent chunks of work make up a single whole.

Simply put, micro frontend-based application means that a number of independent teams are working on it, each delivering its own piece of application separately.

2. Why is micro frontends/plugin architecture the technology of choice for the banks?

Web banking applications as well as mobile banking applications comprise a huge variety of sections and functionalities. New sections appear, while the old ones go through dramatic upgrades. Oftentimes each section is the area of responsibility of an autonomous team.

This is why it's crucial that such autonomous teams should be able to develop, test and deploy their chunk of work independently, under their own schedule.

3. Why implementing micro frontends concept in mobile applications is a difficult task?

First and foremost, mobile applications are monoliths by nature. So micro-anything (be it services or front-ends) simply don't fit in when it comes to mobile development.

In a web application, as a contrast to a mobile one, various sections and functionalities can be easily deployed separately. On the contrary, a mobile application is always deployed as a single file which includes all the functionality. It cannot download part by part or be installed partially. When an update is available, a user has to download the whole application once again to get access to the new functionality.

So, given this, how at all the notion of decoupling can be applied to mobile development? As we've already mentioned, a mobile application is *deployed* as a monolith. Yet, it can be *developed* as a multi-section entity. Thus, when we talk about micro frontends (aka plugin architecture) in mobile development, we mean the process of *development*, not *deployment*.

Due to the changes in the development process, the micro frontend benefits become available to us. This is the key business objective behind the transition, although mobile applications still continue to be deployed as monoliths.

How to move your mobile banking app into micro frontend-based application?

This Section briefly describes the migration approach that Qulix follows in its micro fronted projects. Here we also share with you our step-by-step roadmap for a smooth system migration to a new modular architecture.

1. What approach does Qulix use to migrate mobile applications to micro frontends?

Mobile development cannot duplicate the schema of delivery that is used in web development due to various implementation differences. However, the key principles can very well be shared. Thus *Modular* or *Plugin Architecture* was born.

> *Plugin Architecture is an approach to mobile application development where a monolithic application is decoupled into a host application and several autonomous modules (or plugins).*

Hence, this is one of the approaches through which the concept of micro frontends is realized in mobile applications and the one that Qulix uses in its projects.

Let's enumerate some of the key benefits that plugin architecture brings to the project:

1. smaller codebases that are easier to develop and maintain
2. autonomous teams
3. higher scalability due to higher autonomy of the scalable elements
4. shorter time-to-market.

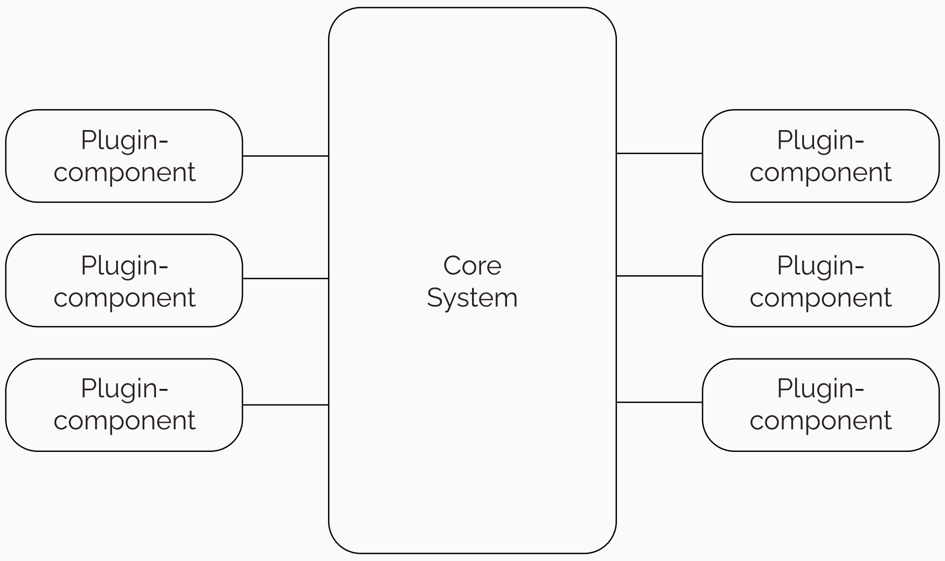
Likewise any technology, plugin architecture has its drawbacks, which include overhead and complexity. We'll consider them in more detail in the next paragraph.

2. Plugin Architecture. General overview

Plugin Architecture implies that a mobile application contains a host application and several independent modules (plugins).

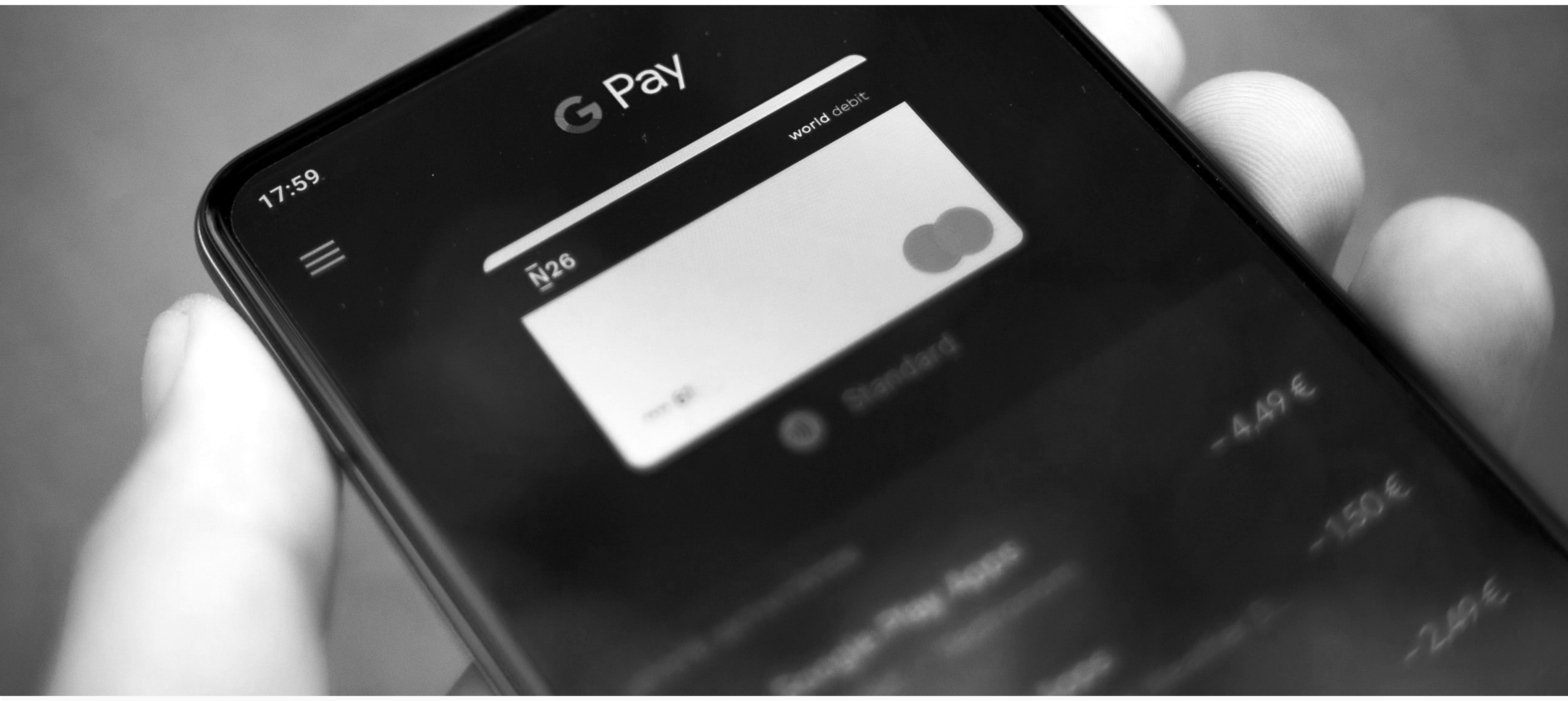
A host application is a frame application which incorporates other modules. A mobile application that users download from the markets is basically a host application after all the modules (a module per functionality) are ready and deployed as a single whole.

See the picture below for example.



This application can be viewed as an example of the micro frontends concept implementation. Each plugin component is an autonomous module that plugs into the host application.

When a feature team (for instance, Payments team) is ready to release its module, the module is incorporated into the host application. The corresponding dependency is added and after the application is deployed, the user sees the new Payments section in its newly downloaded application.



As you see, the application is no longer developed as a monolith, which is a great advantage. At the same time, new/updated functionality is not deployed independently. Each new piece should go through all the SDLC stages, similarly to the monolith development:

1. Develop and test a new module
2. Incorporate/update the module in the host-application
3. Build and deploy the host application
4. Download a new version of the host application to the market
5. Download the application and install it on a user's device.

N.B. To update the user on new application versions, use push-notifications. The push-notifications are sent from the server according to your mailing list.

Example of a push-notification:

A new update is available for your banking application. Download a new version of the application now?

YES **NO**

However, this great approach comes at a cost. Let's see what's on the flipside of micro frontends.

3. Drawbacks of Micro frontends/Plugin Architecture

A. Complexity

Prior to the project start a micro frontend team has to answer to a pretty impressive list of questions, which include:

- Organizational issues (workflows);
- Configuration;
- Environment;
- Processes.

Look at some questions below that your team should discuss before the project start as an example:

- How will the host application function and connect to the plugin modules?
- Will the plugin modules communicate with each other and how, if so? What will be the communication contract?
- How each separate application (plugin) is going to be built (environments, configuration, etc.)?
- Which core components will be developed separately (by the core team)? Is there going to be a core team on the project at all?
- Is there going to be a system design team?

All in all, the complexity of the micro frontends architecture has nothing to do with the development. Rather, it is the processes and settings that should make a multicomponent micro frontends machine work as a single whole. A monolithic application often results to be much easier to build and deploy.

B. Overhead

Implementing micro frontends architecture implies parallel work of several teams.

In general, your micro frontend team will include 3-5 independent feature-teams and 2-3 service teams. Overhead is compensated by the speed of development and generally can be viewed more as a specifics, not as a disadvantage of the micro frontend architecture.

See Section III *How to build a Micro frontend team?* for more details.

4. How to migrate to micro frontends: A step-by-step roadmap

In this Section, we will share with you our step-by-step roadmap for a seamless migration from a monolithic mobile banking application to micro frontends.

N.B. Assumptions

Qulix recommends you the following roadmap assuming that:

- Your system architecture follows microservices design principles
- The system is developed by several independent, vertically divided full-stack teams
- Each team may produce full-stack feature modules: frontend and backend (see Section III for more on team structure).

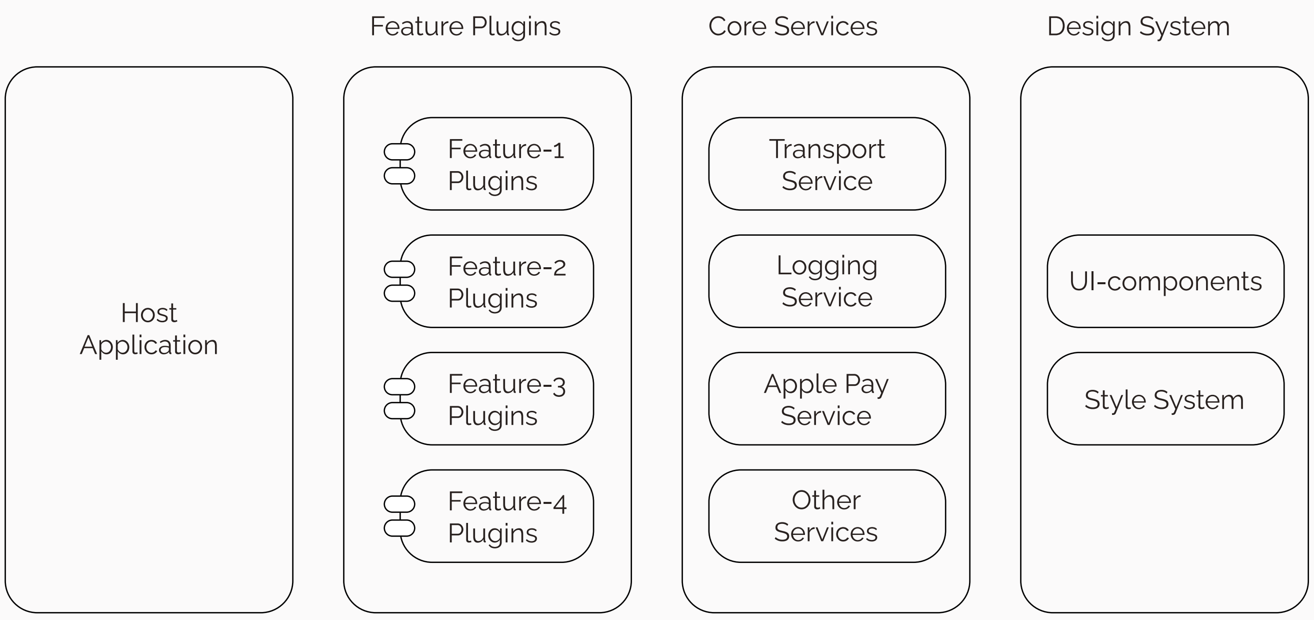
Step 1. System audit

To start with, conduct a detailed system audit. You can hire a vendor to do it for you or carry out the audit by yourself.

Feel free to contact us for auditing or consulting services. We'll be happy to get your system ready for a dramatic update.

As a result of the system audit, you should be able to distinguish specific components within your system that will pertain to:

- Host-application
- Feature-plugins
- Core-services *
- Design system (a library of UI-components, style system) **



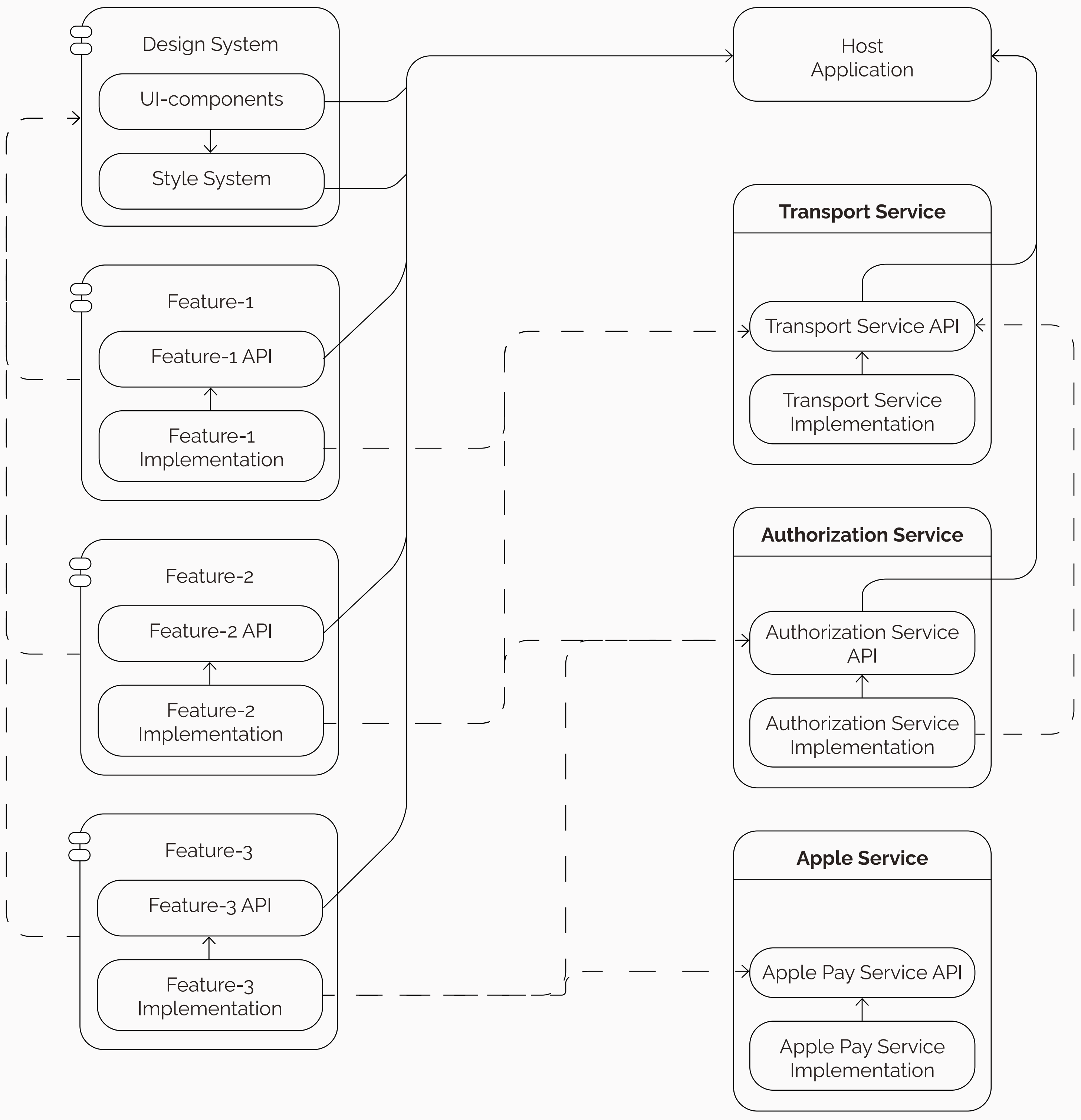
*Each feature team uses a specific set of services (server service, database service, etc.). Oftentimes, those services overlap in different teams. Thus, the core team develops the modules (or libraries) that all the teams can use.

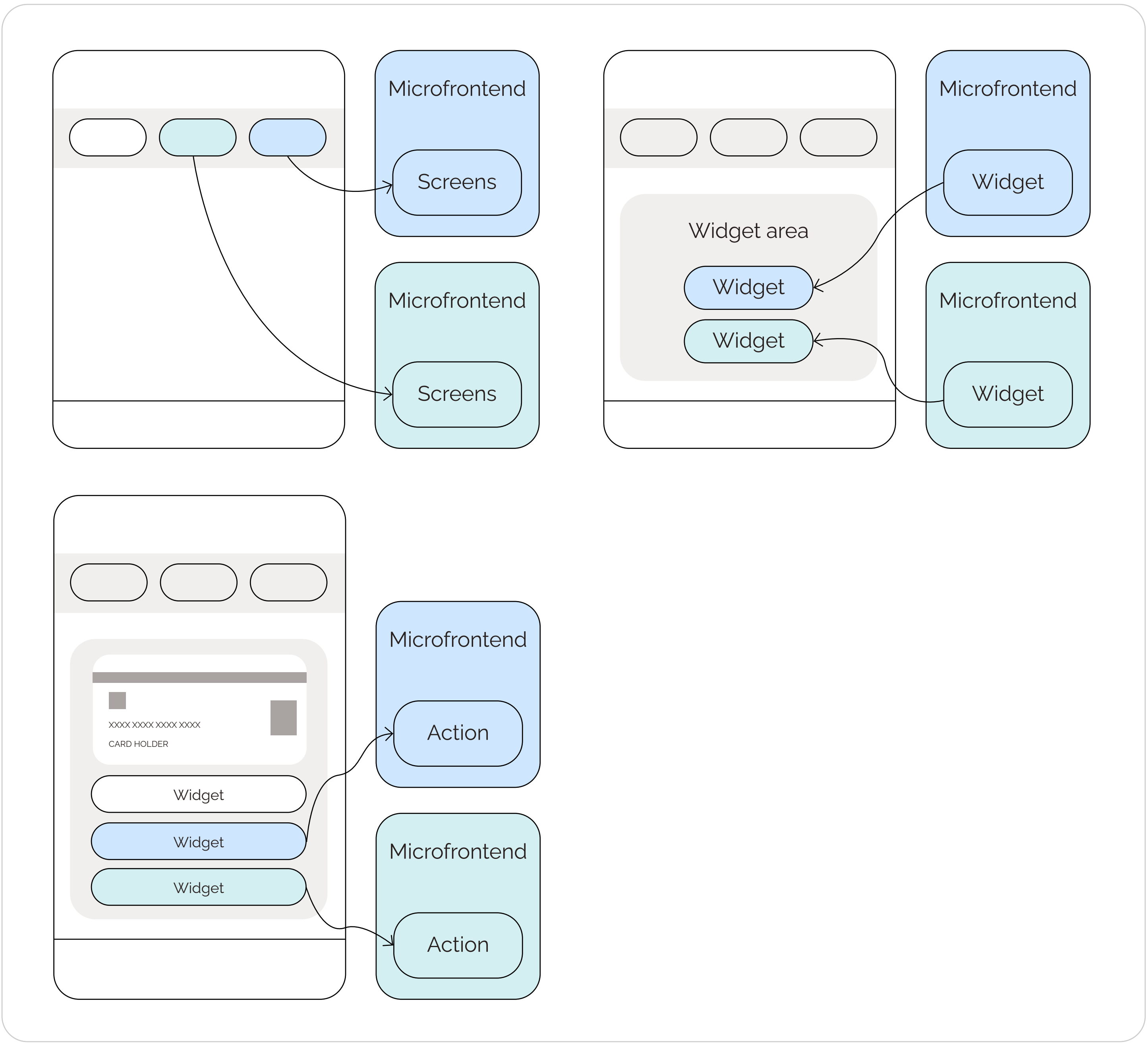
** The design system team is in charge of the style integrity throughout all the application modules (buttons, text fields, other components). In essence, it does the same job as the core team - develops the modules/ libraries that other teams will use to enable integrity and avoid duplication.

Check out Section III for more on micro frontend teams.

Step 2. Interconnection architecture design

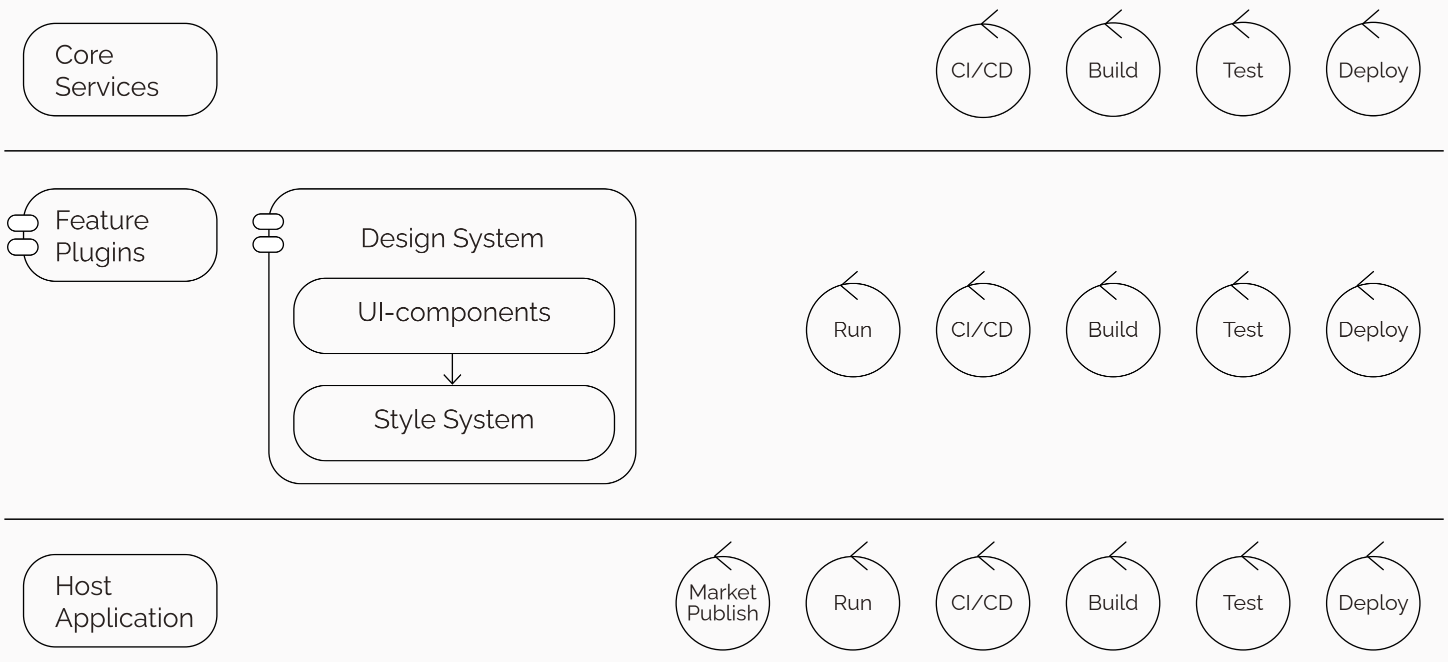
- Reflect the host-application and all the necessary plugins and services on the diagram.
- Design the API for plugins and services that will enable their interaction with the host-application or other plugins/services.
- Build a dependency graph encompassing all the system components.
- Define the mount points for the plugins.



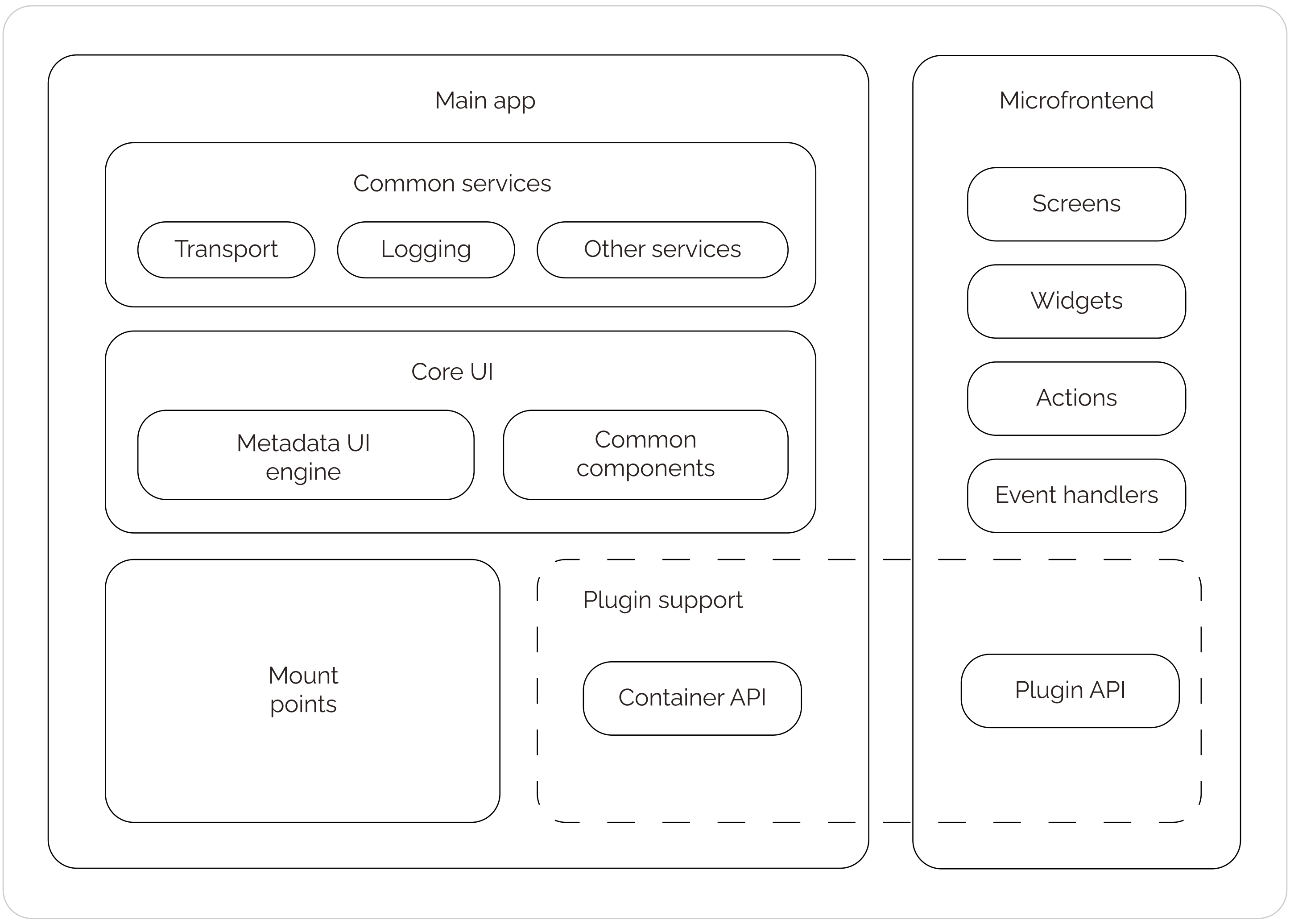


Step 3. Infrastructure and CI/CD processes

- Set-up the infrastructure for configuration management in the host-application and feature-plugins.
- Plan and design the frames for feature-plugins and core-services. In future, all the teams engaged on the project will be able to use those frames for an easy development start.
- Integrate CI/CD processes for plugins and services to enable smooth build, start testing and artifacts publishing.
- Integrate CI/CD for the host-application to start testing, enable dependencies/plugins installation, build and upload the ultimate application to the market.



Step 4. Get to the target micro frontend-based architecture



- Each feature team follows their own development cycle and release schedule
- At release time, the feature team publishes artifacts to the company's Maven repository (Nexus, Artifactory, etc.)
- Feature release event triggers application assembly stage using latest versions of features.
- Application is published internally and later may be uploaded to markets

With this approach you can achieve independent feature delivery without independent deployment. We believe that this is a good compromise given the mobile platform vendors' limitations.

N.B. What are mount points?

In the picture above you have seen one of the crucial terms of the micro frontends concept, which is **mount points**. The term 'mount point' means a place in an application where micro frontends may be attached.

What can serve as a mount point?

- Main menu (or similar UI entity) with dynamic entry list.

Each list entry activates a particular micro frontend capability (open screen, produce event, etc.)

- Widget hosting views (dashboard, etc.).

Microfrontend may provide widgets, users may place it on the dashboard.

- Custom business entity actions.

For example dynamic credit card action list based on micro frontend's abilities to handle such events.

5. Special points to consider

System testing

During the development stage, feature teams cover the system with unit tests.

Prior to deploying the micro frontend-based system, it also goes through exhaustive testing procedures including functional, load-testing, and integration testing. During the testing phase, the environment is created that replicates the real-life environment to the fullest extent possible.

To create such an environment at Qulix we use stub systems - server emulators that generate test data.

Thus, the application goes through the testing stage at two levels - at the feature team level and at the project level. Regardless of how thoroughly the testing was done at the feature-team level, the tests are duplicated at the project level (host-application level).

N.B. Core modules testing

At the testing phase, give especial attention to the core components of your micro frontend application (if such a separate module exists in your project). The core components are those that all the project teams will use heavily, so make sure they undergo the most stringent checkings.

Automation

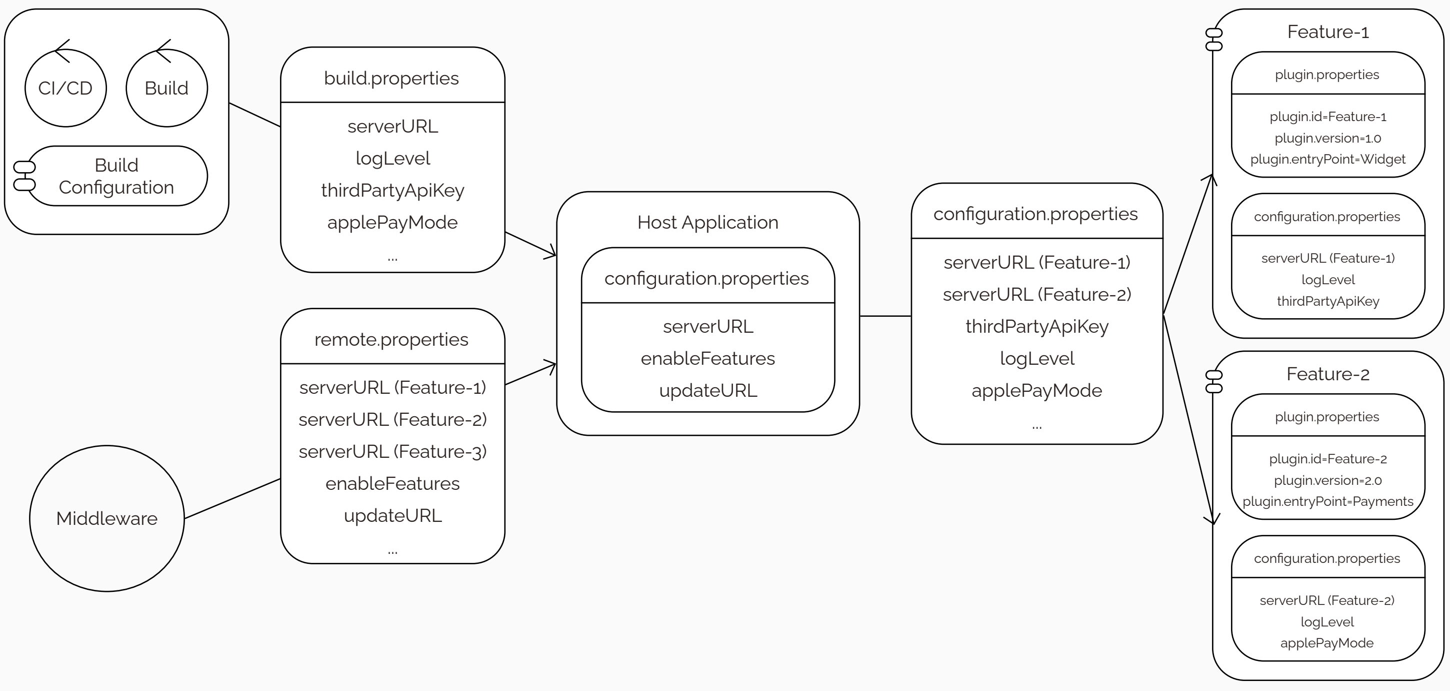
Automation is crucial for correct configuration and infrastructure settings, and those two are crucial for seamless build and deploy of a micro frontend-based application.

If in case of a server side applications Docker is a solution #1 for trouble-free configuration, it is of no help for mobile applications.

At Qulix, to enable configuration automation for mobile applications we have developed our own solution.

All the configuration data are stored in a Common Source of Knowledge (CSK). Thus, the CSK configures the whole product module all at once. The CSK holds all the proper settings for all the features within such a product module. In such a way we minimize manual settings relying on automation, wherever possible.

Ideally, with the CSK in place, the feature build is performed according to the following algorithm:



```
/**
An interface representing microfrontend plugin.
*/
public interface Plugin {

   /**
    The main plugin initializer.
    Plugin initializer needs a configuration and a dependency container.
    If the required configuration / dependency was not found during initial-
ization, an exception will be thrown.

     - Parameters:
       - configuration: key-value configuration.
       - container: dependency container.

      - Throws: PluginInitializationException and its subclasses.
    */
   init(configuration: PluginConfiguration, container: InjectionContainer)
throws

   /**
    Returns plugin entry points structure.
    */
   func getEntryPoint() -> PluginEntryPoint
}
```

Ideally, with the CSK in place, the feature build is performed according to the following algorithm:

start of a command -> configuration checking -> configuration update -> feature build

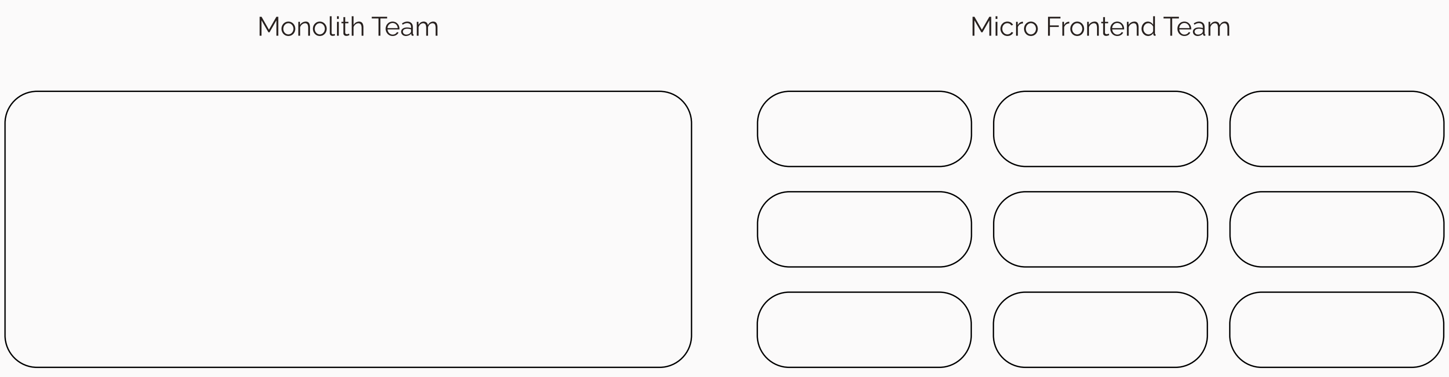
How to build a micro frontend team?

This Section provides a micro frontend team definition and monolith vs micro frontend team comparison. It also contains highlights on project team structure and segregation of duties within it.

1. Monolith vs Micro frontend team structure. Feature teams

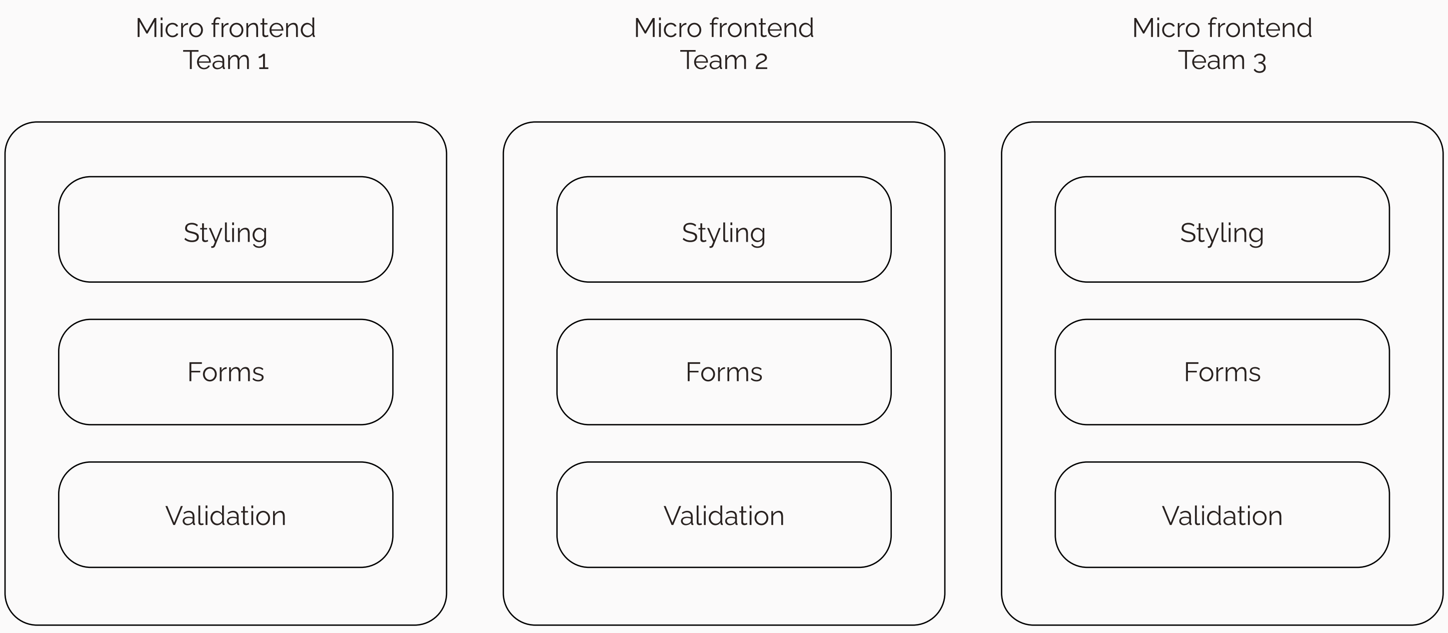
To implement micro frontends requires a different approach to team building.

In a conservative monolithic team, for example, 20 experts are engaged in delivering one common project (horizontal teams). In micro frontend-based applications, the structure of a team includes several autonomous vertical feature-teams, each in charge of its own functionality.



A *feature team* is a full-stack team in charge of all the SDLC for a specific feature. Thus, it includes a backend developer, front-end (iOS, Android), BA, PM, and a QA. The team may also include a product owner, although he/she generally works on the customer's side.

A Product owner develops a business concept, provides a functionality overview, and elaborates on UX/UI details with the UX/UI designer. Both the Product owner and UX/UI designers communicate directly with a the BA. The BA translates the PO's and UI/UX designer's vision into the software development requirements.



N.B. Micro frontend team competencies

Similarly to the microservices, micro frontend architecture requires higher qualifications from a team because of the system's complexity. However, you may very well hire regular **junior developers/ junior QAs**. For these two positions special skills are optional (development according to specifications, heavy components usage).

For a Team lead/Platform lead micro frontend skills are a must. Those include:

- modular application architecture, APIs, communication protocols

Communication issues in a multi-component app require top efforts from the team and advanced skills, as a result.

- advanced testing skills

This is crucial for adequate unit testing of separate plugins and creation of a testing environment that generates test data correctly.

Contact us for additional information on micro frontend team qualifications. We'll share with you our talents' CV templates and discuss their key skills and abilities.

2. Service teams

Moreover, in micro frontend-based architecture, it's important to incorporate a few other teams that will make the development process smooth and seamless.

These are:

- Core team
- System design team
- Platform team

Let's have a closer look at their duties.

a. Core team

The core team develops common modules that are used by all the feature teams (server service, database service, etc.). In some projects the core team also prepares UI components, although generally those are developed by the System design team.

b. System design team

System design team is responsible for the style integrity throughout the whole application (buttons, text fields, headers and footers, etc.).

Likewise in the case of a core team, the System design team creates the shared components so that feature teams can freely use them, if needed.

For UX/UI purposes on a micro frontend project a components showcase is created. A components showcase is a list of all the existing components for each platform. It looks and feels like a real showcase, so a developer can open this section/screen, scroll down and find the component he/she needs. Alongside the visuals, this section contains the names and the examples of code that demonstrate how these components can be integrated into the project. Thus, every developer on a project adds the System design module to his/her list of libraries and uses them, as necessary.

UI components

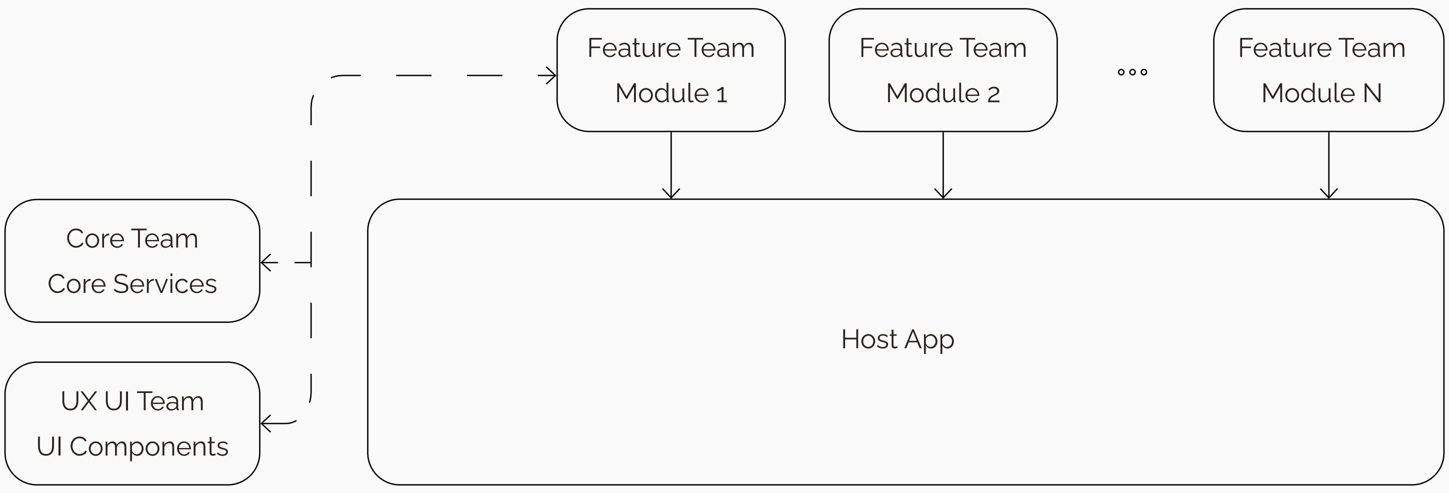
SystemKit	CommonKit	ProductKit
NavigationBar	ProgressBar	AccoutRequisitesView
ToolBar	ActivityView	TransactionListView
CollectionView	GradientHeader	PaymentDetailView
StackView	TextField	ProfileImageView

c. Platform team

The platform team is in charge of the healthy platform (Android, iOS) operation (correct modules connection and seamless builds, settings, appropriate functioning of the platform infrastructure, etc.).

Thus, a micro frontend team is an operational unit comprising several vertical feature-teams (full-stack teams delivering the whole module - from backend to frontend) and several service teams - core team, platform team, system design team, etc.

Every company may have its own unique set of teams for micro frontend projects. However, at Qulix we have come to these four teams that work fine for us: **feature teams, platform team, core team, system design team**.



N.B. Knowledge sharing

Although in a micro frontend application several vertical teams are running autonomously, knowledge sharing is paramount for smooth project delivery. This is especially important for BAs, PMs, and UI/UX designers.

In case a component (validation module, for example) has been used, it should be added to the Design system. A BA, in his/her turn, should also indicate it in the specifications and promote its usage among other teams.

Conclusion

A quick summary on the topic, final thoughts and strategic recommendations before you go.

In this whitepaper, we shared with you our insights on the monolith migration to the micro frontend architecture. There are several highlights from this whitepaper that we would like to give special focus before you go.

Why should banks migrate to micro frontends?

Advantages of this new architecture for the banks include short time-to-market, easy system development and maintenance and higher scalability. A great combination for the large entities willing to become more market and user friendly.

If your business is struggling with a bulky large-scale application where dependencies are unclear and system upgrades are painful and take ages to perform, micro frontends are an option for you.

What migration approach to choose?

At the moment, several approaches exist. At Qulix we use the Plugin Architecture and recommend it to our customers. Our step-by-step roadmap described herein is based on the plugin approach.

Do I have to change my approach to team building to launch a micro frontend project?

Most probably, yes. A micro frontend team comprises several vertical feature-teams (full-stack teams delivering the whole module - from backend to frontend) and several service teams - core team, platform team, system design team, etc. So, if you are following the monolithic practices, a shift to micro frontends will require a new team building approach.

Do micro frontends make the 'plug and play' concept real for mobile development?

No, but they imitate the concept in a very realistic manner.

By now, there is no approach in mobile development that allows using the 'plug and play' concept. A new/updated feature can be introduced to the user only after the whole application was downloaded repeatedly and installed on the user's device. Nevertheless, it can be *developed* as a multi-section entity. Due to the changes in the development process, the micro frontend benefits become available to us, although mobile applications still continue to be *deployed* as monoliths.

This new approach opens up a whole lot of benefits that businesses will perceive in a relatively short time perspective. **We recommend micro frontends to our banking and finance clients willing to become more flexible and client-oriented**.

Mind that this architecture will impose **higher qualification requirements** mainly for the project leaders (tech leads, architects, senior developers). Additionally, despite running autonomously, the teams should cooperate as a single whole, which will require certain delivery **workflow modifications**.

Thus, a to-do list before the project start will include:

- Comprehensive system audit

Is your application going to grow fast soon? What are your scalability requirements?

- Team revision

Check out which skills your team has and which ones should be replenished.

- Workflow modification

Revise your workflows and make necessary modifications to enable autonomous teams running. At the same time, pay attention to the teams communication and knowledge sharing.

Micro frontends are a new promising movement in the mobile development industry. Despite the implementation difficulties, its benefits make it a great option for the market players that seek ultimate flexibility. If you are among them, don't hesitate to give micro frontends a try.

About Qulix

Custom Software Development Company Operating Globally

Qulix is an international custom software development company delivering high-quality software solutions. Since 2000, we have been rendering top-notch development and QA services to our 200+ clients from all over the world. Our expertise covers all the stages of SDLC, which includes concept design, architecture design, code development, quality assurance, support and more.

Qulix delivers turn-key and custom software projects for banking, finance, insurance, multimedia, IoT and other areas. The list of the services that we offer includes backend and web apps development, mobile and cloud apps development, QA services, UI/UX design, and DevOps.

Find more of our best practices by visiting our blog or website.

For any further questions, please contact us. We'll be happy to share with you our insights on the topic to help your business grow.

tel: +44 151 528 8015 **email:** request@qulix.com

UK:
Oakwood, Dunstan Lane, Burton, Neston,
Cheshire, United Kingdom, CH64 8TQ

Poland:
Braniborska 40, 53–680
Wrocław, Poland

Uzbekistan:
130 Mustakillik Avenue, Mirzo-Ulugbek
District, Tashkent, Uzbekistan, 100077