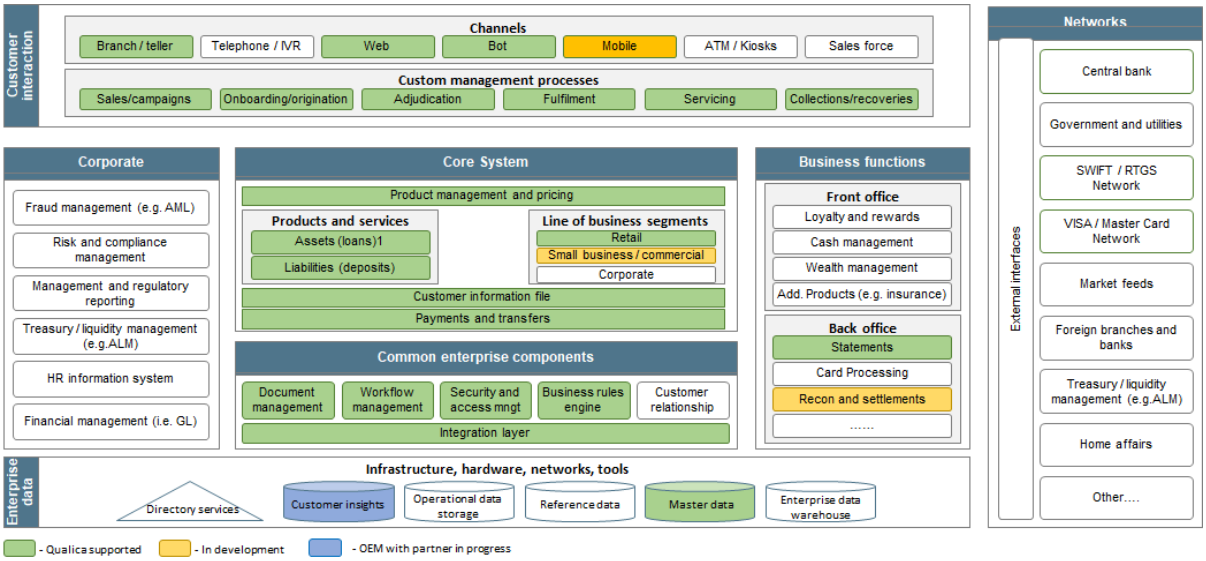
Application Architecture

Flexifin: Built for the rapidly changing world of financial services.

“In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies”
-- James Lewis and Martin Fowler

Radix has adopted the microservice architectural style for composing applications. The Radix suite of services comprises some 30+ services exposing over 400 endpoints. This service catalogue is constantly growing as we expand into various core-banking domains.

High Level Application Architecture



The core system encompasses the following services:

Account Number

This service allows FlexiFin to return an account number from number pools that have been configured against specific formats according to the specific needs of the “bank”. It can be in the format and numbering pattern that any historical system uses, so that documents such as statements etc. will have

the historical numbering system in place and will also support a “shadow” account in front of a core bank application bulk account.

Type : Restful API,

Assessment

Assessment as a service coordinates the affordability assessment of financial means whether it be via an internal assessment scorecard or integration into a third party bureau call. The function of the service is to determine, analyse and return the affordability of an applicant.

Communication: Restful API

Application Assembly

An important aspect of loan assembly is straight through processing, and the ability to pick up a process midstream when it was not previously completed. FlexiFin in this service maintains the state of the loan application assembly process, in a non-blocking way, through its life cycle. It allows the execution of elements of the process via either a worklist, or via straight through processing. Each loan application maintains its own state and as such does not rely on centralised workflow engines to maintain state. This makes is highly scalable.

Communication: Restful API

Bank Account Verification

The bank account verification service is a routing gateway to various third party providers that provide the functionality to validate and/or verify bank account details. This provides a flexible and configurable “facade” to allow for externalisation of this requirement in different geographies. In addition this service stores the information and provides other services with the result of the verification. E.g. anti money laundering and customer due diligence / know you customer services can utilise this information as required.

Communication : Restful API

Business Credit Bureau

The Business Credit Bureau service is a gateway service that integrates into third party credit data providers and provides the functionality to deal with the retrieval and posting of credit profile data for juristic persons.

Communication : Restful API

Collections

The debtor collections module is designed as a proactive mechanism for managing delinquent accounts, and recognises that collections strategies are numerous and most effective when applied conditionally. The module receives data from either the FlexiFin accounting modules or a third party accounting application, from where it manages the lifecycle of early stage collections.

Debtor collections reporting is aimed at managing key performance indicators and informing decision making.

Communication : Restful API

Credit Bureau

The Credit Bureau service is a gateway service that integrates into third party credit data providers and provides functionality to deal with the retrieval and posting of credit profile data for natural persons.

Decisioning

The Decisioning service is FlexiFin platform's stand-alone decisioning service to track and execute decisions.

Decisions are built in the proprietary language of the decisioning service via a user interface, or excel rules can be built and imported into the decisioning engine. Any rule result can then be made available to any of the FF services (or to any non FF services / third party applications).

Document Generator

The Document Generator service maintains document templates with "mail merge" functionality in order to produce personalised, system-generated documents.

The service feeds templates to other services in order to allow these services to gather available data via any resource endpoint within the platform, collapse it into a data object and return it to the flexifin-document-generator for merge and storage.

Effectively any data available in the platform can be converted into a HTML based template for processing into a PDF.

Disbursement

The Disbursement service deals with the processing of outgoing funds from the system.

The Disbursement service is separated from the Payment service that handles the processing of incoming funds, due to the need to scale outgoing money in a different manner. This also ensures that the larger batch processing nature of incoming fund processing is not a blocking factor when dealing with the disbursement of funds.

Fixed Term Lending Account Servicing

The FTLAS (Fixed Term Lending Account Service) is responsible for the generation of a fixed term repayment schedule based on the configuration of a lending product (from the product catalog service), the creation of a subsidiary debtor loan account, and the management of all incoming and outgoing monies to this loan, providing interest on an actual / actual daily basis. In addition all loan management functions such as settlement re-advances, write-off, restructure, refunds, re-allocations, and early settlement are handled by this Service. The Service is responsible for emitting events related to the loan that will be used by subsidiary microservices responsible for the querying of the debtor loan account and the financial ledger.

Identity Verification

The identity verification service is a routing gateway to various third party providers that provide the functionality to validate and/or verify identity details. This provides a flexible and configurable “facade” to allow for externalisation of this requirement in different geographies. In addition this service stores the information and provides other services with the result of the verification. E.g. anti money laundering and customer due diligence / know you customer services can utilise this information as required.

Communication: Restful API

Master Data

The Master Data service manages and maintains the party domain model with all associated meta data related to entities within the party model. The party domain model provides the ability to define and manage the multiple roles that a party can perform, as well as maintain the relationships related parties or roles. Due to the graph nature of the model, the system is able create any type of role definition and maintain any type of relationship between any roles.

Messaging

The messaging service is a gateway router to provide outbound and/or inbound SMS and email. It is used to provide status messages from automated processes within the system as well as disseminating documents such as statements.

One Time Pin

FlexiFin’s One Time Pin service provides SMS and/or email routing for unique secured one time pins to be sent from the platform.

Payment Gateway

FlexiFin’s Payment Gateway service provides unified routing to Third Party Payment Providers (TPPP). The service provides a “facade” to abstract third party functionality away from the platforms’ services, in order to provide a single interface to deal with payments.

Payment Hub

The Payment hub service provides the framework in which all payment instructions are managed. It allows for streaming capabilities to handle ISO 8583 and ISO 20022 and AXS protocols as well as the management of all rules surrounding ACH / PCH providers.

The service controls the processing of RTC (Real Time Clearing) and Batch for EFT and Card instructions to PCH / ACH Payment Gateway service.

Product Catalog

The Product Catalog service provides the ability to define product configurations based on predefined data templates for the various modules within the system. Each product template creates a standard template such as “fixed term lending”, from which independent configurations can be created. These

configurations are used to feed the product parameters into the Schedule Generator service, in order to generate product schedules.

Product Ledger

Product Ledger is an event sourced ledger to financially represent all transactional events within the FlexiFin platform. It brings together the domains of loans, disbursements and payments into a financial view in order to facilitate financial reporting within the platform itself. The product ledger for savings and transaction / current accounts is being refactored to be event sourced and will also use CQRS.

Properties

FlexiFin's Properties service provides for the centralised management of business rules (properties) and is based on the Spring Cloud Config technology to propagate business configuration throughout the platform. The Properties Service controls how to integrate configuration parameters into the code, what to expose to a UI and what is configurable through a UI vs viewed in a UI.
In Java, any service (in app.properties) that starts up, gets is properties from the properties service.

Quote

The Quote service is a light-weight service that tracks various quote types within the system. Its function is to store and maintain all metadata associated with the generation of quotes. The quote service supports the generation of printable quote documents.

Schedule Generator

FlexiFin's Schedule Generator service implements the integrated Matlab calculator which, based on a specific set of provided inputs, will generate a loan schedule.

Scheduling

FlexiFin's Scheduling service provides a centralised unified scheduler to control the execution of timed events across the platform.

Worklist

The Worklist service provides the platform with the ability to manage user based tasks in a worklist, adding in permission, escalation and tracking.

The worklist service is driven by a modern process management engine that supports the trio of Business Process Modelling Notation (BPMN), case model and management notation CMMN and decision management and model notation (DMN).

BPMN is very well suited for automated well defined and structured processes. Within Flexifin we use BPMN to manage transactions across microservices. CMMN on the other hand is extremely useful in managing *unstructured processes* ; processes that do do not always take place in a predictable way. Typical applications for the CMMN standard include:

- Dealing with exceptional cases, for example, customer complaints.
- Complex evaluations, for example, suspicion of fraud in claims settlement or dealing with complex financing arrangements.

Similar to BPMN and CMMN, DMN peaks when modeled decisions are executed by a compatible decision engine. This offers the following advantages:

Transparency: Everyone can understand how decisions are being made the knowledge is not locked in arcane source code.

Traceability: Every decision can be logged by the decision engine.

Flexibility: Decision logic can be changed rapidly without having to roll out new processes and lengthy training cycles.

All of the above services expose REST API's and have embedded within the service event handlers and event emitters for asynchronous communications. In addition each service implements a set of cross cutting concerns as a standard service pattern.

Cross cutting concerns

Each of the Services contained within FlexiFin are essentially capable of standing alone, yet they are aware of each other, and the Platform has to ensure that the interaction between the Services is managed. As a result, each Service needs to have a number of functions / capabilities that both allow for the ease of achieving certain requirements by the Service, as well as for the integrated, Service awareness within the Platform of the Services. These are referred to as "cross cutting concerns" as they cut across the Services within the Platform. The key concerns are listed below.

RuleBook

The RuleBook is a cross cutting concern within each microservice component of the system that provides the Service with the ability to code and manage business based logic and rules in a configurable manner across each discrete service in the platform. This gives the Platform incredible business flexibility as each service, and each instance of a Service can have different rules depending on the operating model of each business.

The Rule Engine library makes use of the RuleBook framework to execute and evaluate specified rules against the fact(s) provided.

RuleBook, allows a service to have a pattern where "pre" and "post" execution rules are maintained and enforced by the service, rather than hard-coding these in a method for the service. This enhances maintainability and adaptability. RuleBook is not a substitute for a decisioning rules engine, which is used for complex decisions. Such a rules engine is contained within the FlexiFin Platform.

RuleBook is a great tool for developers looking to get a handle on business logic and abstract it away from application logic. The driving force behind RuleBook is that it should be

intuitive; it should work how you work, and in many ways, that's what it does. RuleBook provides a simple and intuitive DSL (domain specific language) that is incredibly flexible. If you have a larger collection of Rules, they can be built as annotated POJOs (plain old Java objects), and RuleBook can transform the whole package into a RuleBook instantly.

Currently, RuleBook implements a single pattern for chaining Rules: Chain of Responsibility. The benefit of using this pattern, is that it is useful in applications more commonplace rather than amassing a rules library consisting of millions and millions of rules. Abstracting a command chain or a workflow is a perfect reason to use RuleBook. Processing dozens or even hundreds of Rules is also a great use for RuleBook. Basically, if you want a solution to the common problems often associated with layered business logic that you can implement in minutes instead of days, or even weeks, then RuleBook is a powerful and flexible solution.
Security Adapter and SSO
These days companies gather and manage vast amounts of user data. Loss of this data will expose companies to financial and legal liabilities or damage to their brand due to governments having introduced new regulations like the General Data Protection Regulation (GDPR). This has made security a major cross-cutting concern for application architecture.

In order to address the above security issues Flexifin makes use of Keycloak. Keycloak is a modern open source Identity and Access Management solution. It enables us to secure all sorts of frontend applications (apps) / services and offers the following features:
Single Sign-On
Identity Brokering and Social Login
User Federation
Client Adapters
Standard Protocols
Admin Console
User Account Management Console
Keycloak handles both authentication and authorisation within the architecture. Authentication is implemented with user credentials being returned in a signed JWT token. Authorisation is implemented at the resource level and the authorisation configuration is external to the code implementing the services. This means that Flexifin is able to implement any Role Based Authorization and Attribute Based Authorisation without having to implement specific roles at the service level as access to the resource being requested is evaluated at runtime against the Keycloak policy enforcement point,. This architecture provides for almost any combination of models. See keycloak Appendix for more on Keycloak.
Distributed tracing via the OpenTracing standard
https://opentracing.io/
In an application that consists of distributed microservices, distributed tracing is required to track the progress of a request through the application and to see where you might have excessive latency, a communication breakage or a misbehaving service. A distributed trace works by assigning a unique identifier, known as the trace id, to each request as it enters the system and then ensuring that the identifier is propagated to every service that deals with that request. Each service adds its own identifier, called the span, to the trace so you can track how long each part of the path took to complete. Once the entire request is completed, the

data containing the complete trace along with all the timestamp is sent to a central statistics service which allows querying and visualization of the trace data via a web interface. The current trace and span ids are also added to every log message, so it is possible to track the progress in the aggregated system logs for more detailed diagnosis of an issue.

Orchestration and Saga's

Each Service implements the Orchestration/Workflow tool Camunda to create a working process of and for a saga that executes an ordered flow of tasks. This gives each Service a workflow capability to adapt the specific business processes needed to support the target operating model desired, and of course which changes from time to time. The pattern and tool implemented in the Services allows the business to change its processes from time to time giving enormous flexibility to business.

A saga is a sequence of local transactions where each transaction updates data within a single service. The first transaction is initiated by an external request corresponding to the system operation, and then each subsequent step is triggered by the completion of the previous one. If a local transaction fails because it violates a business rule then the saga executes a series of compensating transactions that undo the changes that were made by the preceding local transactions.

Transactions are an essential part of applications. Without them, it would be impossible to maintain data consistency.

One of the most powerful types of transactions is called a Two-Phase Commit, which is, in summary, when the commit of a first transactions depends on the completion of a second. It is useful especially when you have to update multiples entities at the same time, like activating a loan, loading the initial balance on the loan and making the payment(s) to the borrower as a single action.

However, when you are working with Microservices, things get more complicated. Each service is a system apart with its own database, and you can no longer leverage the simplicity of local two-phase-commits to maintain the consistency of your whole system. Also as there are microservices that are "stand alone" and by design (for scalability , and maintainability etc) are not aware of the impact of their performance or function on other services, a way of managing and orchestrating the transactions within one Service that may rely on the functions of many other Services is needed, to allow for the entire system to operate in a predictable and efficient manner.

In order to achieve this consistency across the whole system, a consistent a pattern must be adopted to allow multiple actions to be performed on individual services and have successful actions be rolled back should a particular failure occur across the "business saga". In order to accomplish this a well defined pattern has been documented and adopted to manage distributed transactions

Traceability

Traceability is the ability to assign and have a single unique ID to a call which in turn may fire multiple sub calls each of which need to carry the initiating ID with it such that we are

able to trace any independent call back to its originating call. In addition this pattern / concern across all all the Services allows for the visibility of load, back-pressure and system health checks. These are absolutely critical checks in a microservices architecture which is utilising the power of cloud based “IaaS” utilising Docker and Kubernetes to provide for on demand elastic scaling of microservices.

In an application that consists of distributed microservices, distributed tracing is required to track the progress of a request through the application and to see where FlexiFin may have excessive latency, a communication breakage or a misbehaving service. A distributed trace works by assigning a unique identifier, known as the trace ID, to each request as it enters the system and then ensuring that the identifier is propagated to every service that deals with that request. Each service adds its own identifier, called the span, to the trace so you can track how long each part of the path took to complete. Once the entire request is completed, the data containing the complete trace along with all the timestamp is sent to a central statistics service which allows querying and visualization of the trace data via a web interface. The current trace and span IDs are also added to every log message, so it is possible to track the progress in the aggregated system logs for more detailed diagnosis of an issue.

OpenTracing is a new, open distributed tracing standard for applications and OSS packages. Developers with experience building microservices at scale understand the role and importance of distributed tracing: per-process logging and metric monitoring have their place, but neither can reconstruct the elaborate journeys that transactions take as they propagate across a distributed system. Distributed traces are these journeys.

OpenTracing is that “single, standard mechanism.” OpenTracing allows developers of application code, OSS packages, and OSS services to instrument their own code without binding to any particular tracing vendor. Every component of a distributed system can be instrumented in isolation, and the distributed application maintainer can choose (or switch, or multiplex) a downstream tracing technologies with a configuration change.
OpenTracing provides—is standardization of span management APIs, inter-process propagation APIs, and ideally active span management APIs

"Audit Logging (source to destination)" - includes traceability, and the who? inside of auditing, incl what to expose through to the UI and what stays in audit trails in system logs. UI viewable auditing must be for HUMAN ENGLISH consumption. Our system uses hibernate envers to maintain an audit of who is doing what with an audit log table within the database

All of the above is key in managing the integration of the microservices across FlexiFin to ensure the developers as well as Devops teams are able to understand, track and audit all inter-microservice events and transactions.
Unified Exception Handler
Should there be any points of failure in the code, the exception handler decides whether to store this error in logs, pass into an API, view the error through a UI, and what further actions need to be done with the error i.e. trigger an action whether in UI to DevOps or to Users in the system.

Fault-tolerant object-oriented software systems are inherently complex and have to cope with an increasing number of exceptional conditions in order to meet the system's dependability requirements. FlexiFin’s application software architecture integrates uniformly both concurrent and sequential exception handling.

The exception handling architecture is independent of programming language or exception handling mechanism, and its use can minimize the complexity caused by the handling of abnormal behavior.

The patterns allow a clear separation of concerns between the system's functionality and the exception handling facilities
User interface and Front end architecture.
As Flexifin is a collection of Microservices which expose API’s through an API gateway almost any front end technology can be used to access to back end resources, providing the security protocols are adhered to. As a standard Flexifin using the two of the most popular single page application frameworks in the public domain. These being React and AngularJS. Typically React has been used in conjunction with Single Page Application (SPA) that need to run on mobile as React makes it relatively easy to customise portions of the framework to optimise it for Mobile applications and of course React native targets Android and Apple operating systems at the OS level.

The AngularJS framework is used for to development a component model based on Google’s material design. Angular is ideal here as the framework is relatively opinionated and as such ensures consistency of approach from a front end development point of view. AngularJS also allows for the development of components which can be generalised and re used. To this end the architecture has been written to configure components as JSON data with a view to allowing business users to configure front ends without having to rely on development. The road map for this will encompass the W3C Web Component standard which provides for the creation of custom HTML tags which can be used within any SPA framework. This vision here to to provide a workbench which allows users to drag and drop components onto a canvas and to wire up these components to the services API either directly to the API or via the workflow engine. This approach is very similar to independent front end engines such as Avoka or Backbase.