FOLIO
Open Library Services Platform

Jakub Skoczen, Architect Index Data

Goals (summary)

purpose of the project:

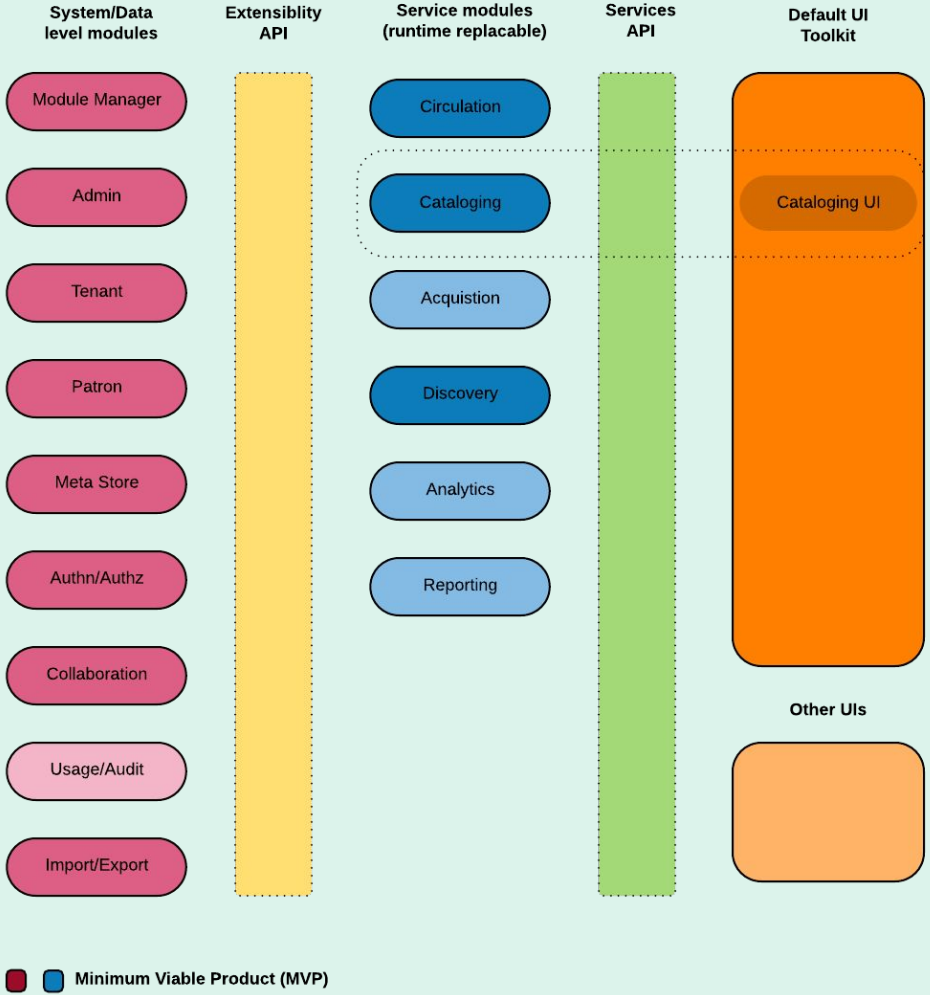
“stimulate innovation by making possible a vibrant community and marketplace in which developers and libraries can participate”

technical approach:

“support a decentralized, highly modular approach to the development of library services, in contrast to traditional ILS products which have been traditionally monolithic”

How to achieve these goals?

- platform that is extensible from the ground up — extensibility realized through pluggable architecture with well defined interfaces/APIs, rather than complex (and rigid) configurability
- microservices inspired — allow development teams to choose their own programming stacks and interface with the core through standard means (HTTP/REST, JSON/XML, etc)
- cloud ready — scalable and ready for deployment on cloud infrastructure (AWS) but not bound to a particular vendor, multi-tenancy as a core feature of the architecture, heavily instrumented


System/Data level modules
Extensiblity API
Service modules (runtime replacable)
Services API
Default UI Toolkit
Module Manager
Admin
Tenant
Patron
Meta Store
Authn/Authz
Collaboration
Usage/Audit
Import/Export
Circulation
Cataloging
Acquistion
Discovery
Analytics
Reporting
Cataloging UI
Other UIs
Minimum Viable Product (MVP)

OKAPI – the backend

- middleware, though not in the traditional sense, “switchboard”

- Inspired by the API Gateway pattern (microservices), with ideas from the Message Queue design pattern (broadcasting)

- core responsibilities such as deployment, monitoring, service discovery, caching, request shaping and management (possibly more, eg static response handling)


OKAPIS
A OKAPI IS A MAMMAL THAT IS A HORSE, ZEBRA,AND GIRAFFE

OKAPI – implementation details

- built with extremely fast non-blocking and polyglot networking toolkit (vert.x)
- relies on HTTP (and its semantics) for the services' transport protocol
- promotes RESTful style of application protocols and JSON as the message (data) format
- allows to build request/response processing pipelines — by combining multiple services under a single routing entry and forwarding headers/messages according to service's metadata
- Comes with a powerful Event Bus that can be exposed with various protocols (e.g STOMP, AMQP)

OKAPI – interfaces/APIs

- OKAPI's own API — RESTful API to register and manage services (aka "modules") and tenants (tenant ↔ module associations), documented and versioned

- pluggable interfaces — versioned, both operations (RAML) and data schemas (JSON Schema), implementations of thereof provided by modules, replaceability

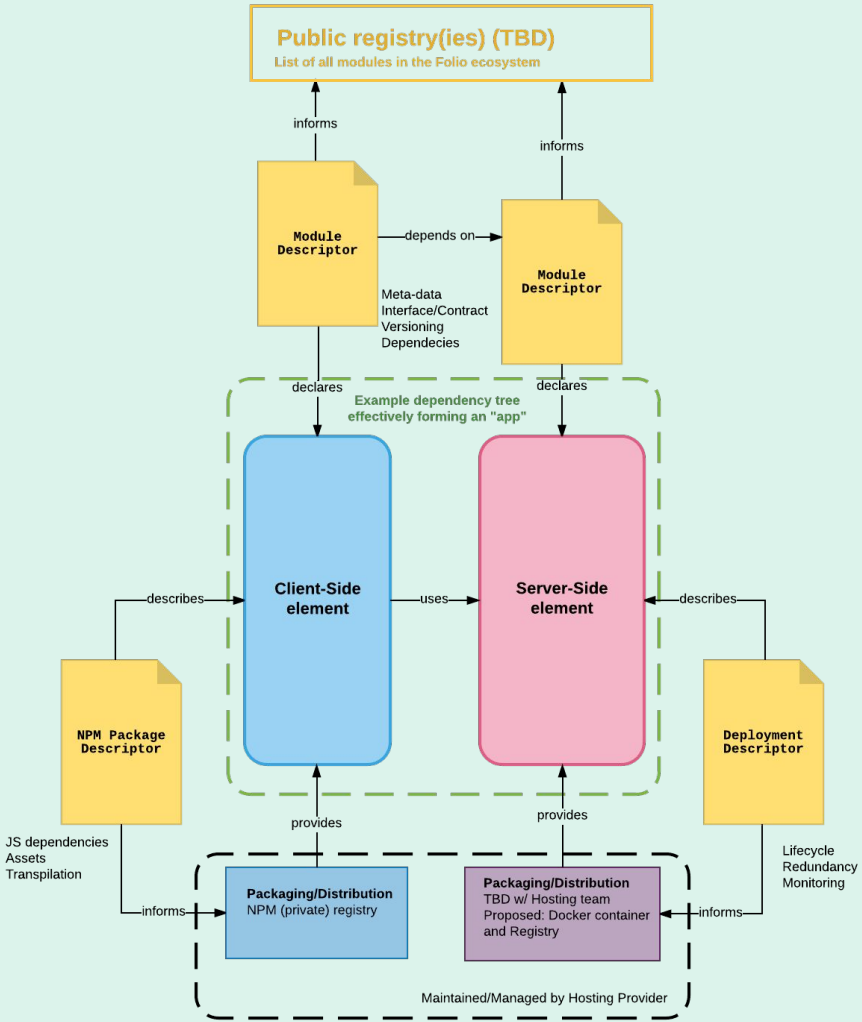
- "dynamic binding" — dependencies on interfaces, not implementations

Modules (on the server)

- self-contained HTTP services — programming language agnostic, freedom of choice wrt programming stack
- implement a set of hooks for registration/maintenance/monitoring, no constraints on the provided APIs but REST+JSON preferred
- stateless — easily load-balanced, must support auto-scaling
- persistence — externally provided flexible APIs for working with core data models and configuration

Modules (on the client, UI)

- SPAs with rich client/browser technologies — re-generated for tenant after module selection (for speed)
- UI toolkit — application of React and Redux, “leaky” abstraction
- Decoupled from but with clearly expressed dependencies on the back-end services


Public registry(ies) (TBD)
List of all modules in the Folio ecosystem
informs
informs
Module Descriptor
depends on
Module Descriptor
Meta-data
Interface/Contract
Versioning
Dependecies
declares
declares
Example dependency tree effectively forming an "app"
describes
Client-Side element
uses
Server-Side element
describes
NPM Package Descriptor
Deployment Descriptor
provides
provides
JS dependencies
Assets
Transpilation
informs
Packaging/Distribution
NPM (private) registry
Packaging/Distribution
TBD w/ Hosting team
Proposed: Docker container and Registry
informs
Lifecycle
Redundancy
Monitoring
Maintained/Managed by Hosting Provider

Building blocks for the “app store” metaphor

- modules may be grouped (dependencies); can be enabled dynamically, per tenant
- applications providing all patron and staff features of a library system: Cataloging, Circulation, Discovery, Acquisitions, etc
- tools with varying complexities of provided functionality and workflows, to fit libraries of different shapes and sizes