Designing and implementing a modern portal architecture

and the lessons I learnt along the way...
Introduction

Who am I?

How did I end up here?

What am I talking about today?
Where did Manage Centre come from?
User experience
User experience
History of development

- Vendor selection process
- External UX design firm
- Outsourced dev team with internal business stakeholders
- Start-up sized team
Lesson one: Having the right Product Manager is critical
Historic architecture
DEVOPS TALKS CONFERENCE 2019

Platform
- Red Hat Linux
- OS-level Virtualisation
- Web Server / Reverse Proxy / Load Balance

Runtime Environment
- Java Virtual Machine
- Polyglot event-driven Application Framework
- Security Framework

Language
- JavaScript
- In-Memory Data Grid
- Content Delivery Network and File storage
Languages
- HTML5 - Structure
- CSS3 - Style
- JavaScript - Behaviour

Frameworks
- AngularJS by Google
  Single Page Application Framework
- jQuery
  DOM Manipulation
- Bootstrap
  Styling & Design

Special Purpose
- HighCharts
  Charting
- KeyLines
  Flowcharting
- Sass
  Stylesheet processing
Problems we have
Lesson two: Web Components are really cool and you should use them
So what do we want in a portal architecture?

- Follows micro-service and micro-frontend patterns
- Something that is easy to contribute to *if you meet the standards*
- Ability to surface data from new data sources quickly
- Interfaces that have a consistent user experience
Well that’s nice but how do we get there?
Conceptual models

Charting model

Action model

Entity model
Entity model
Charting model

External consumer

Public API

Chart executor

Integration layer

SQL extractor
NoSQL extractor
MOLAP extractor
...

SQL DB
NoSQL DB
MOLAP
...

Private API

Public API

Client API

Mobile

Web UI
Action model

External consumer

Public API

Private API

Integration

Internal systems

Web UI

Mobile

Client API

Public API

Action service

3rd party APIs

MSP APIs

EMR

Action registry

Workload services

Automation platform

Action registry contains details about an action, logic to execute, multi-step workflow etc. Might just be an automation platform and the "action service" is TAC.
Making it real
Patterns and tech we want to use more of

- Micro-front ends
- Progressive Web Apps
- GraphQL
Micro-frontends
Micro-frontends

Benefits
• A feature team can control the entire flow from data through to user interface
• Teams can in theory right in whatever web framework they like
• Smaller pieces of UI functionality that can be re-used between different portals
• Skating to where the puck will be, Web Components

Challenges
• Have to get the integration model into the parent app right
• App bundle size can get out of hand if not tightly governed
• Styling can get tricky with lots of different components
Basecoat
Lesson three: Having a living style guide makes the “look and feel” part of UX fantastically easy to share across teams.
Micro-frontends

• Why do we think it is cool?
  • Extends the same ideas and thinking about microservices to the front-end
  • Delivers the same benefits (and challenges) as microservices

• How does it fit the concepts?
  • With the right standards lets us decentralize development effort to the teams developing the services
  • Lets us share components between teams and other portals in the business more easily
GraphQL

• Why do we think it is cool?
  • Fantastically powerful query language, only get the data you asked for
  • Makes it easy to add new data sources or change them without impacting the front end
  • Makes it easy to “mash up” data from multiple sources completely invisibly to the system asking for it

• How does it fit the concepts?
  • Lets you define a single way of representing entities which can be shared across portals
  • Provides a single consistent interface for accessing data from any data source
Progressive Web Apps

- Progressive
- Responsive
- Fast after initial load
- Connectivity independent
- App-like
- Fresh
- Safe
- Discoverable

- Re-engageable
- Installable
- Linkable
Progressive Web Apps

- **Why do we think it is cool?**
  - Enables app like behavior without needing to have native applications
  - Shared code base across mobile platforms (don’t need Android and iOS teams)

- **How does it fit the concepts?**
  - Makes the content available on any platform
  - Re-uses the common services to deliver a consistent experience
  - Can be refreshed with new technology since it sits on top of the business logic layer
The future of portals at NTT
Thank you!