REQUEST FOR COMMENTS

MyAccount NG web application

Problem & Context

And we progress with project Highlander Babel is slowly being phased out in preference of CommerceTools and as such applications and systems that depend upon Babel either need to be deprecated or replaced.

Currently user authentication and management is still handled by a number of older babel web pages (which in turn query Babel API) that have yet to be migrated over to the Rollercoaster platform; this RFC will lay out the scope of the migration alongside a list of technology choices considered for the new platform.

Considered, but out of Scope

The migration of user management over from Babel to Commercercetools has been broken down into Four distinct phases.

1. **Replacement of user authentication interfaces (using existing babel session flow)**
2. Replace account management interfaces.
    a. Profile preferences
    b. Credits
    c. Albums
    d. Creations
3. Split user & credentials migration into Commercetools.
    a. Migrate base user entity and credentials
4. Entirely new set of user and authentication services with JWT/OIDC.
    a. Cognito authentication handlers
    b. Middleware authentication layer
    c. Localisation of orchestration endpoints
5. Fully migrate all user data as a separate project.

This RFC will only focus on the **first phase** of the migration, the replacement of the user authentication UI.

Project constraints

- Design resources are limited and given our rather tight deadlines we cannot wait on full UX/UI design.
- Originally a new team was allocated to pick up the MyAccount project, now all work will be done by core team members.

Technical constraints

- Must be a whitelabel solution and support multiple brands
- Babel is very slow, as such UX and technology choices need to minimize the impact of this on our customers.

Technical requirements

- All API requests will go via orchestration-v2.
- All Artifacts will be deployed to AWS.
- The application will use the React framework.
- The application will be written in Typescript.
- E2E tests will use Cypress.
- Unit tests will be written with Jest.
- The application must support Experimentation via external configuration.
- The build pipeline will use Github actions.

Security constraints

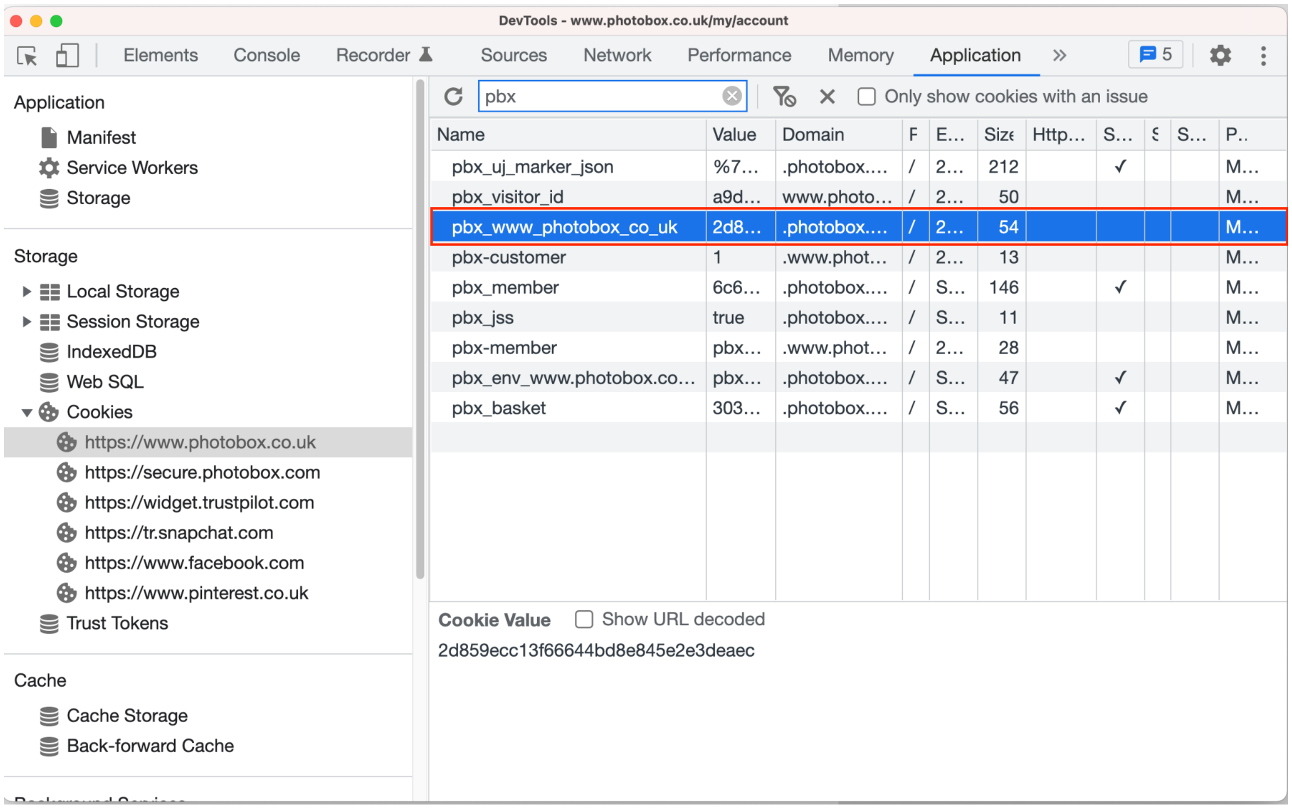
Currently whenever a new babel session cookie is created the policy settings applied are known to be insecure.

OWASP recommends that cookies which contain sensitive information are flagged as **HttpOnly,** however this is not feasible within the photobox estate due to how a number of the older applications retrieve and submit session tokens.

Babel web extracts session tokens using nested Javascript which would no longer function if the session cookie was flagged as **HttpOnly**.

mStudio also relies on the session cookie being accessible via Javascript, as such until both Babel web and mStudio have been decommissioned we cannot flag the session cookies as **HttpOnly**.

It should be noted that we are aware of the security implications of this decision, but we cannot fix these deficiencies until Babel Web & mStudio have been decommissioned (Phase 4 of the my account migration).



Name	Value	Domain	F	E…	Size	Http…	S…	ξ	S…	P..
pbx_uj_marker_json	%7…	.photobox.…	/	2…	212		✓			M…
pbx_visitor_id	a9d…	www.photo…	/	2…	50					M…
pbx_www_photobox_co_uk	2d8…	.photobox.…	/	2…	54					M…
pbx-customer	1	.www.phot…	/	2…	13					M…
pbx_member	6c6…	.photobox.…	/	S…	146		✓			M…
pbx_jss	true	.photobox.…	/	S…	11					M…
pbx-member	pbx…	.www.phot…	/	2…	28					M…
pbx_env_www.photobox.co…	pbx…	.photobox.…	/	S…	47		✓			M…
pbx_basket	303…	.photobox.…	/	S…	56		✓			M…

Cookie Value ☐ Show URL decoded

2d859ecc13f66644bd8e845e2e3deaec

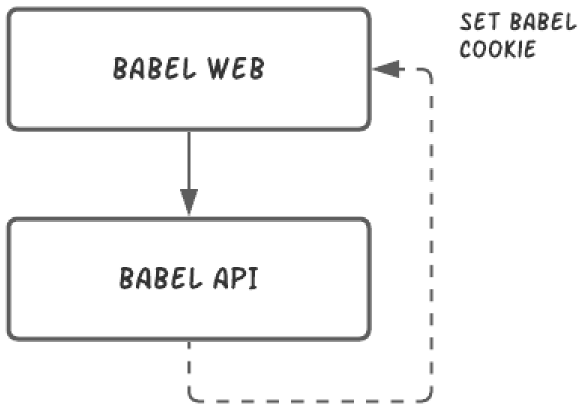
Longer term we wish to address this constraint.

General assumptions

- User accounts will be scoped per brand.
- Longer term we wish to add **3rd party login support** (Google, facebook and so on)
- The apps will use native (Babel) authentication for phase one, this may change when we integrate social login.

High level architecture

Current architecture



The current architecture is pretty simple, whenever you visit a Photobox site a new cookie is created and it's name is scoped to the locale domain.

Most applications contain logic to generate this cookie

Proposal one

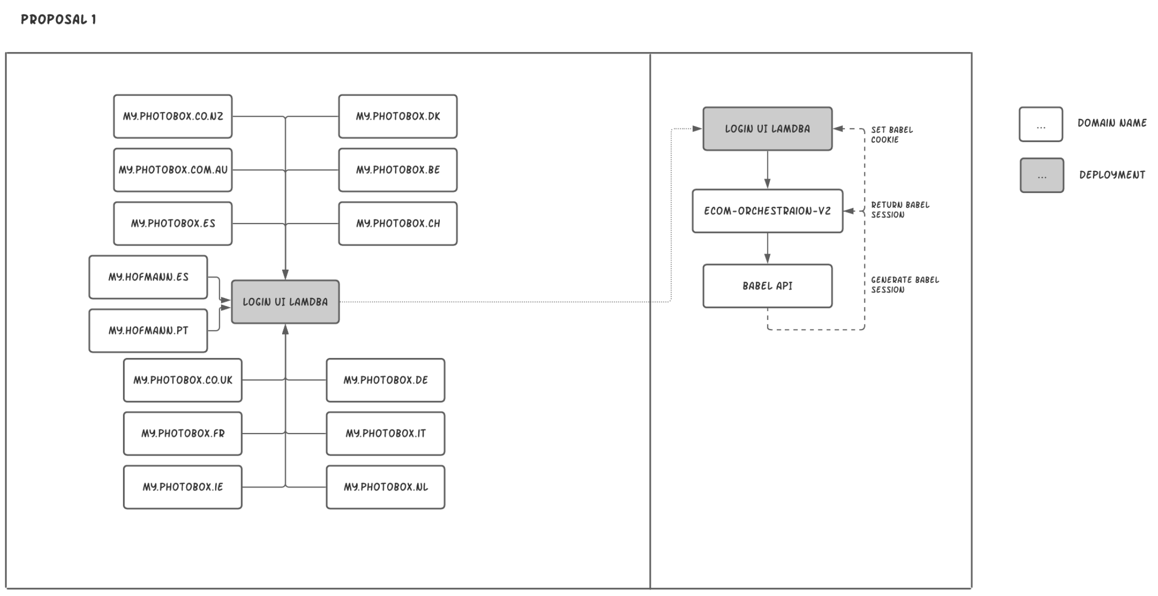
This is the simplest of the two proposal's when only considering **Phase one** of the "My Account" migration, but the overhead required to adapt it for **Phase four** would be notable.

TLDR; a new authentication web application will be created, deployed and made available via a subdomain for each of our supported brands/locales. Etc the authentication web application for ***photobox.co.uk*** would be made available ***my.photobox.com*** (actual domain TBD).

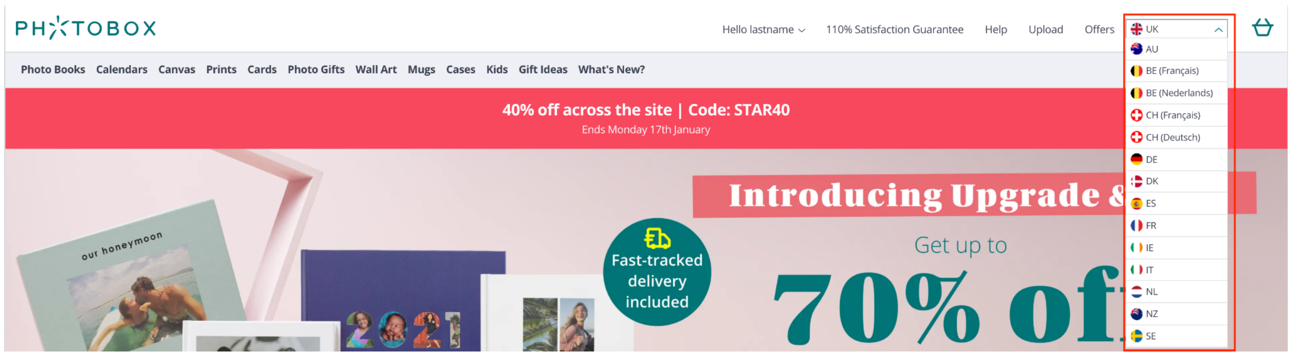
And within this subdomain the authentication web application would be made available via scoped URL, the following example uses photobox.co.uk.

- The **login UI** would be ***my.photobox.com/identity/login***
- The **register UI** would be ***my.photobox.com/identity/register***
- The **password UI** URL would be ***my.photobox.com/identity/password-reset***

Note there will only be one deployment of the authentication web application but it will be served (masked) across multiple domain names.

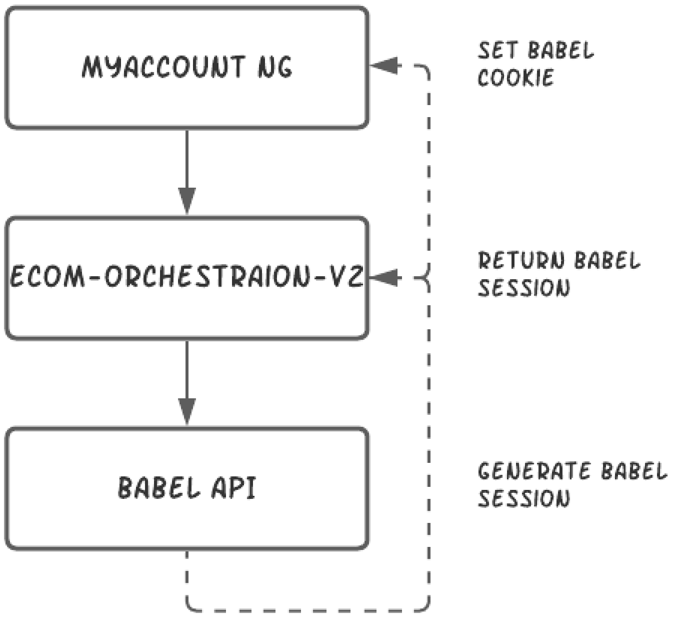


Given that cookies are always scoped to the current locale of a user that is logged into (say photobox.co.uk) a user currently needs to login again if they change locales (say navigate to photobox.fr). It should be noted that this is an edge case but we do provide quick locale switching via the web headers. Proposal 2 will allow for multi locale logins.



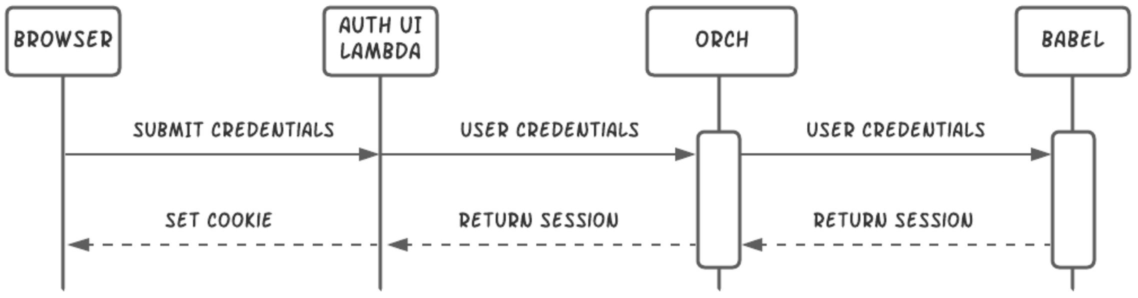
This new application will be very minimal and only allow for user login, registration and password resets. The actual user management application would be built as a separate application in **phase** two of the migration**.**

Please note we have made a number of assumptions based on competitor analysis and general security trends/guidance.



The authentication flow will remain nearly identical to the current babel flow.

The application will call orchestration-v2 (which in turn calls babel) to generate a babel session and then simply set the same session cookie.



At a high level we would only need to build a fairly simple react app that can proxy calls to orchestration-v2 and set cookies.

There are also a number of additional considerations.

We eventually plan to move away from insecure cookies and implement **HttpOnly** session cookies, This in turn means that the authentication application cannot be purely Client side. HttpOnly cookies can only be set and read server side and are completely hidden from web clients.

This single consideration dictates that the React application needs to be server side rendered (**SSR**) even if we do not plan to implement **HttpOnly** cookies during phase one.

There is an argument to be made that session tokens should be stored in **localstorage** and not cookies, this argument is also valid given that a cookie's value cannot exceed 4096 bytes.

However the general consensus is that session tokens should be stored as secure **HttpOnly** cookies. This is the **OWASP** recommendation and it is a pattern followed by most web applications.

Despite most large technology platforms supporting **localstorage** based session storage (such as AWS cognito/amplify) they do not use it within their own core products.

It should be noted that we plan for this application to become the Photobox authentication gateway, unlike other applications which are only used in isolation and security (whilst still a concern) is not as important given the scoped nature of the session tokens they store.

HttpOnly Attribute¶

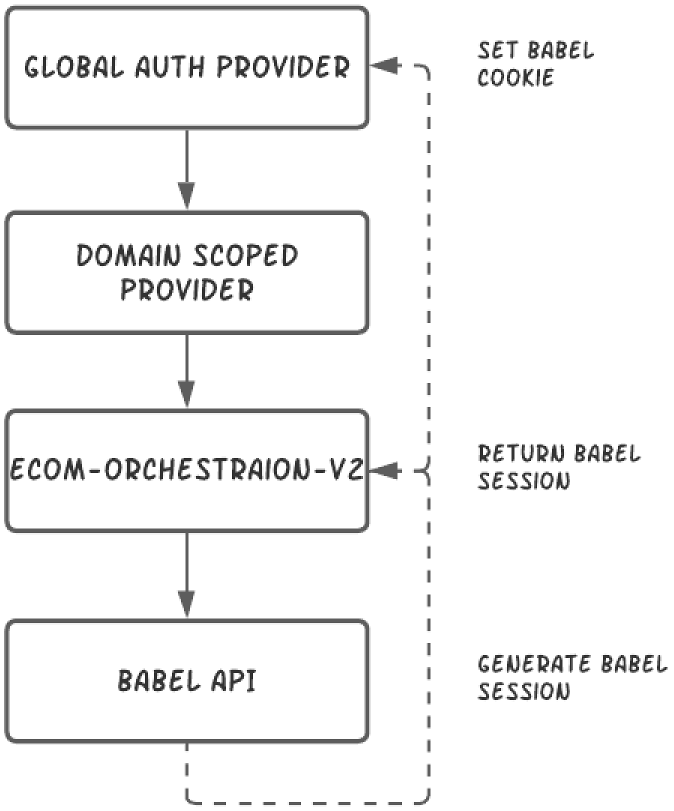
The HttpOnly cookie attribute instructs web browsers not to allow scripts (e.g. JavaScript or VBscript) an ability to access the cookies via the DOM document.cookie object. This session ID protection is mandatory to prevent session ID stealing through XSS attacks. However, if an XSS attack is combined with a CSRF attack, the requests sent to the web application will include the session cookie, as the browser always includes the cookies when sending requests. The HttpOnly cookie only protects the confidentiality of the cookie; the attacker cannot use it offline, outside of the context of an XSS attack.

But it should be noted that we do not need to (nor can we) implement HttpOnly cookies currently as it would break a number of our legacy applications.

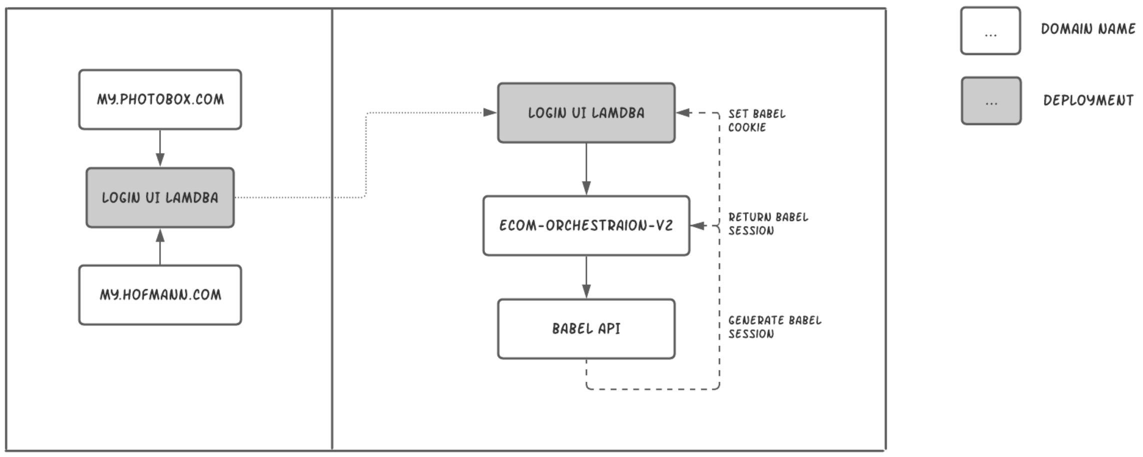
We would also need to localize orchestration for every brand/locale domain that we currently operate.

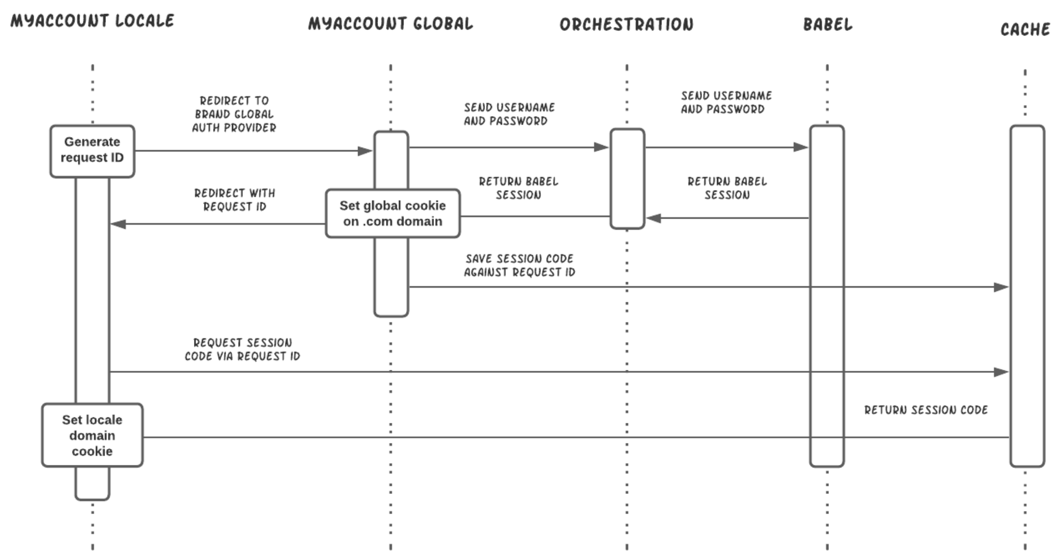
This model aligns with ASOS if you would like to navigate an already existing solution.

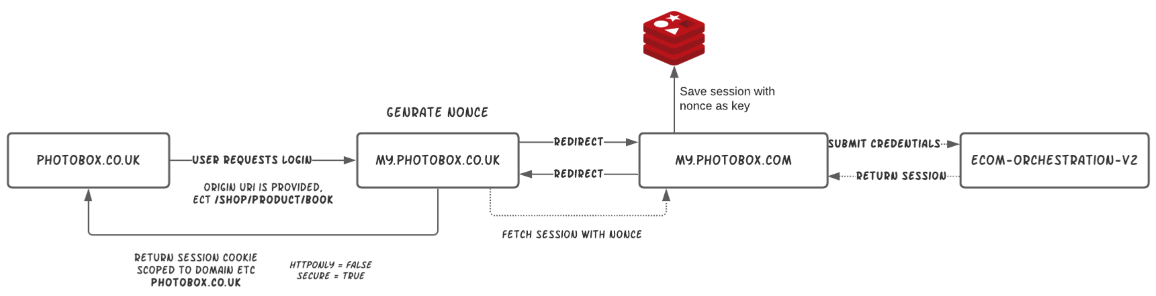
Proposal two


GLOBAL AUTH PROVIDER
DOMAIN SCOPED PROVIDER
ECOM-ORCHESTRAION-V2
BABEL API
SET BABEL COOKIE
RETURN BABEL SESSION
GENERATE BABEL SESSION

PROPOSAL 2


MY.PHOTOBOX.COM
LOGIN UI LAMDBA
MY.HOFMANN.COM
LOGIN UI LAMDBA
ECOM-ORCHESTRAION-V2
BABEL API
SET BABEL COOKIE
RETURN BABEL SESSION
GENERATE BABEL SESSION
...
DOMAIN NAME
...
DEPLOYMENT


MYACCOUNT LOCALE
MYACCOUNT GLOBAL
ORCHESTRATION
BABEL
CACHE
Generate request ID
REDIRECT TO BRAND GLOBAL AUTH PROVIDER
SEND USERNAME AND PASSWORD
SEND USERNAME AND PASSWORD
RETURN BABEL SESSION
RETURN BABEL SESSION
REDIRECT WITH REQUEST ID
Set global cookie on .com domain
SAVE SESSION CODE AGAINST REQUEST ID
REQUEST SESSION CODE VIA REQUEST ID
RETURN SESSION CODE
Set locale domain cookie



Technical architecture

React rendering pipeline

A separate RFC has been created with additional information on React rendering pipelines which can be found here.

Recommendation:

Server side rendered application (SSR)

To framework or not to framework

A separate RFC has been created with additional information on React frameworks which can be found here.

Recommendation:

Custom application that can support SSR for user authentication and CSR for account management.

React state management

A separate RFC has been created with additional information on React state management which can be found here.

Recommendation:

React Context in conjunction with redux and redux sagas.

Avoid connected containers.

Styling methodology

A separate RFC has been created with additional information regarding CSS styling methodologies which can be found here.

Recommendation:

Tailwind alongside emotion

Application bundler

Recommendation:

Webpack 5.0

Deployment framework

Recommendation:

Serverless 2.0 >

CI/CD

Recommendation:

Github actions

Technology stack summary.

- Server side rendered SSR.
- Custom React app.
- React Context state along with redux and redux-sagas.
- Tailwind CSS with emotion ui.
- Webpack 5.0
- Serverless framework

Domain configuration

Both proposals will require that an authentication application is deployed to it's own subdomain for each of our locales.

- my.photobox.co.uk
- my.photobox.fr
- my.photobox.ie
- my.photobox.de
- my.photobox.it
- As so on

This will also have an impact on our deployment pipelines and their domains, currently we only utilize the photobox.com domain and each of the locales are assigned subdomains from the TLD.

If **proposal one** is chosen then additional work will need to be done in regards to service localisation.

Given our longer term desire to flag all authentication cookies as **HttpOnly** we need to ensure that all resources are available from the same domain. For example a cookie scoped to ***.photobox.co.uk** and marked as HttpOnly will only be submitted for requests either within the TLD (top level domain) or subdomains (such as my.photobox.com).

As such **ecom-orchestration-v2** would need to have additional domain names per brand/locale.

https://graph.photobox.co.uk or https://photobox.co.uk/graphl.

If **proposal two** was chosen then the HttpOnly cookie issue goes away, given that authentication is always pushed to a service deployed within photobox.com for photobox and hofmann.es for hofmann that domain will always have the session cookie available.

Deployment

Given the suggestion that the authentication application should utilize SSR and the user management app should use CSR we need to ensure that our deployment process can accommodate both approaches.

We can follow our standard deployment process with the serverless framework via Github actions.

Each feature branch will be given it's own feature build that can be used for online testing and regression testing.

The application will be deployed to new locale specific sub-domains.

We have examples of both react CSR and SSR serverless configurations.

https://github.com/photobox/ecom-account-web/blob/main/custom-ssr-sls-browser/serverless-full.yml

Experimentation

We currently have a fully functional AB test API that is used to dice roll experiment variants. We simply need to hook up the existing experiment service into the web-application and then we can make use of existing AB testing libraries that are already available with the react ecosystem.

One of the more common react AB testing libraries is `@marvelapp/react-ab-test` which has native react hook integration and you can toggle on a debug component allowing for experiment component testing.

Dependencies & Integration

There are no technical dependencies for phase one, however design resources will be required.

Infrastructure

- AWS Lambda (for application).
- AWS S3 (for application assets).
- Maybe AWS Elasticache (to cache nonce keys).

Scale & Performance

Given that all authentication calls will still query Babel all of our current scalability constraints will still apply.

However the applica itself will be deployed as a Lambda function so the application in itself has almost limitless scalability.

Reliability

The Application will only be as reliable as Babel.

Redundancy

If Babel is down the application will also be down.

Monitoring & Instrumentation

Dynatrace and Sentry will be implemented from the start giving us visibility into the application.

Failure Scenarios

If the Babel API goes down all authentication requests will fail.

If there is an interruption to AWS then the application will not be accessible.

Security

We know that there will be a number of security issues regarding cookies, however these are accepted risks given the flaws are already present within babel and we have a plan to remedy them.

Privacy

Operational Implications

The application will be owned by core.

Future Directions

Please see

Rollout

The application will be rolled out and tested in isolation, once we have confidence in it’s reliability all login, registration and password resets to babel will be redirected to the new application.

Competitor Analysis

https://docs.google.com/document/d/1-KAJD-y58vbLlTaY8lzCBYzfMod_W8VEvLf9tFx_kCg/edit

Current state of MyAccount

https://docs.google.com/document/d/10y-qbTq6LAnkeyV3xsdMG8TUfmXt6yKMtOo-7OEylo0/edit

Risks & Open Questions