ROUTE 06

Platform for Platformer

to B プロダクトで Vite + React Router を採用して半年後の感想

Hirotaka Miyagi / @MH4GF

2024/02/28 TechBrew in 東京 〜フロントエンド技術選定、その後どうなった？〜

自己紹介

ROUTE 06

- Hirotaka Miyagi / @MH4GF
- GraphQL, GitHub Actions, 静的解析が好き
- ROUTE06, inc.
    - フルリモートワークの会社で北海道から沖縄までメンバーが在籍しています！🗾

今日話すこと

ROUTE 06

- 2023 年に、所属チームのプロダクトで **Vite + React Router による SPA** を採用しました
- Next.js や Remix などのフレームワークを利用せず、自前で実装した経緯・半年後の感想を共有します

今日話すこと

ROUTE 06

過去に公開した技術選定記事のうち、Vite の採用にフォーカスした詳細版です

2023-08-08

Plainのフロントエンドにおける技術選定(2023年8月版)

ROUTE06 でソフトウェアエンジニアをしている @MH4GF です。
ROUTE06 ではエンタープライズ向けビジネスプラットフォーム「Plain」を開発しています。この記事では 2023 年 8 月に Plain クラウド EDI の Web フロントエンドで採用している技術について、その選定理由をまとめました。
現代の Web フロントエンド技術は領域ごとに選択肢が多く、プロダクトに最適な技術選定をする上で検討事項が多いと感じます。この記事がフロントエンド技術選定において参考になれば幸いです

Vite を採用した背景 の前に...

ROUTE 06

今日伝えたいこと

向き合うプロダクトの特徴やフェーズ、取るべきリスク・回避すべきリスクを整理し、プロダクトに最適な技術選定を行う

事業には主に「技術リスク」と「市場リスク」がある

技術リスク

主に「作れるかどうか分からない」リスク。

- 科学的にできるか
- 品質を担保できるか
- 大量生産できるか

市場リスク

主に「売れるかどうか分からない」リスク。

- ニーズや課題があるか
- 市場は大きいか（急成長するか）
- 規制がどうなるか

16

ref: 「作ってから売る」と「売ってから作る」と「売れるようにしてから作る」　～技術の社会実装のための『開発』～ - Speaker Deck

プロダクトの概要

ROUTE 06

- **エンタープライズ向け SaaS**・マルチテナントアーキテクチャ
- 現在開発中のプロダクト「Plain EDI」は商取引における**クラウド EDI** にフォーカス
- 技術観点では**入力内容の多い複雑なフォームとデータ量の多いテーブル表示**が主な機能
- Web アプリケーションのみの提供で、主にデスクトップでの利用を想定

このプロダクトのフロントエンド領域を取り巻くリスク

ROUTE 06

- 市場リスク ... **ドメインの複雑さをどう低減し、売れる機能を提供するか？**
- 市場リスク + 技術リスク ... SaaS とフルスクラッチの中間のテーラーメイドの領域を狙い、お客様ごとのカスタマイズ性を強みとしたい → **テナント固有実装をどう安全に、効率よく、柔軟に提供するか？**
- 技術リスク ... **現代のフロントエンド領域の選択肢の多さ・移り変わりの速さ・発展途上が故の不安定さにどう対処するか？**

技術選定の背景

ROUTE 06

技術選定の観点はいくつかあるものの、我々のプロダクトとして重視したかったこと

- **UX やカスタマイズ実装のようなプロダクト固有の課題**に対して時間を投資できるようにしたい
- 初期フェーズのため、**開発中の技術的に詰まる時間**はできるだけ減らしたい
    - (もちろん開発者としては、詰まった時など隙を見て OSS への貢献もしたい)

Vite 採用の背景

ROUTE 06

以下の 2 つに分解できます

- **not SSR という選択**
- フレームワークを使わないという選択

not SSR という選択

ROUTE 06

- **レンダリングでサポートすべき環境が複数あることによる複雑さ**
  - 「このコードってブラウザと Node.js のどっちで実行されてるんだっけ？」
  - 「Storybook では動くけどテストの JSDOM ではなぜか動かない」
- GraphQL のクエリキャッシュを Node.js とブラウザで共有することが難しい

→ 開発体験の観点では、Node.js ランタイム上でレンダリングすることで得られる恩恵があまりなく、開発中問題にぶつかる時間の方が大きい？

not SSR という選択

ROUTE 06

プロダクトのユーザー体験の観点としてはどうか？

- データの入力や保存、インタラクションが多く、**ブラウザで動作する JS が主となる**
- 業務用 PC での利用が多くネットワーク環境も安定している想定のため、 SPA のデメリットである **SEO や初回データフェッチ量の制限・バンドルサイズの増加に強い影響を受けない**
- 認証前のページもログインページしかない

→ **SPA の方が相性が良いのではないか？**

フレームワークを使わない選択

ROUTE 06

- React としてはフレームワークを使うべき
  - React プロジェクトを始める – React
  - この主張はその通りだと思います

フレームワークを使わない選択

ROUTE 06

当時(2023 年 5 月頃)は React で業務アプリケーションとしての SPA を実現する主要なフレームワークは **Next.js Static Export のみだった**

- App Router の場合、**ダイナミックルーティングがビルド時に決定したパスしか使えないため** `:id` のようなパスが使えないのが最も大きなネック
- Next.js の方向性としても、キャッシュやパフォーマンス最適化など **to B プロダクトには too much な機能が多かった**

フレームワークを使わない選択

ROUTE 06

フレームワークを使わないことによる自前実装のトレードオフは何があるか？

- データフェッチ方法 ... GraphQL を使うため、どのフレームワークを使うとしても `useQuery()` を使うことに変わらない
- ルーティングライブラリの選定 ... 様々なライブラリが名乗りを挙げているが、**React Router** で要件を満たせそう
- チャンク分割の実装 ... Next.js が行ってくれていたような細かいチャンク分割を自分で取り組む必要がある
    - 初回データフェッチ量の制限・バンドルサイズの増加の影響を強く受けないため、問題が出てから対応する形で良い

フレームワークを使わない選択

ROUTE 06

> **いずれ離れることを想定して依存する**
>
> ~~ 中略 ~~
>
> 私は技術選定の際はフロントエンドに限らず、採用するものを
>
> - Tier 1 : これを捨てる時はコードを一から書き直す覚悟を持つ「心中する」相手
> - Tier 2 : 差し替えには大きな労力を必要とする「強く依存する」相手
> - Tier 3 : いずれ差し替えることを想定した「依存を軽くする」相手
>
> ぐらいにカテゴライズしています。

ref: フロントエンドの移り変わりは早すぎるのか

フレームワークを使わない選択

ROUTE 06

- ただの React を素朴に使うだけであれば、**将来的な別フレームワークへの移植容易性は高いと判断**
  - データフェッチは GraphQL で行っており、フレームワークの機能に依存していなかった
  - 例えば App Router に移植したとしても Client Component として再利用できる
- React は **「心中する」**相手だが、 Vite は**「依存を軽くする」**相手

Vite + React Router を採用

ROUTE 06

- **not SSR という選択** ... 開発体験・ユーザー体験の双方から恩恵が受けづらく、開発中問題にぶつかる時間の方が大きいため、考えることを減らしたかった
- **フレームワークを使わない選択** ... プロダクトに最適なフレームワークがない・自前実装のトレードオフを整理したところ問題ない・将来的な移行容易性も高いと判断

→ **Vite + React Router を採用しました**

半年経った感想

半年経った感想 - Vite

ROUTE 06

- 👍 「ブラウザで動作すれば良い」というシンプルさはかなり享受でき、機能開発や他の技術調査に注力できた
  - Storybook の Play Function でインテグレーションテストを書けば、JSDOM 環境も不要
- 👍 Vite の HMR はかなり快適
- 👍 「Vite だからできない」ことはほとんどない
  - Vite プラグインの充実性
  - pnpm workspace での monorepo 環境のビルドも問題ない

チームメンバー「開発中 Vite が理由で何かに詰まった記憶がない」

半年経った感想 - Vite

ROUTE 06

- 👍 ビルド成果物が静的アセットだけになることによるシンプルさ
  - S3 + CloudFront 構成で済むためでコストもかなり安く、PR に紐づくプレビュー環境の構築容易性も高い
  - ライブラリや Node.js のアップデートの影響も、ビルド結果の Diff を見て測りやすい

半年経った感想 - React Router

ROUTE 06

- 👍 複雑な要件(パンくずリストの動的生成・ページトラッキングの実装など)も問題なく実装できた
  - 事例や情報も多いが、v6 未満の情報も多いので注意が必要 😐
- 👍 Sentry が手厚くサポートしており、パフォーマンスモニタリングがしやすいのは選定時には気づいていなかった
- 😐 ページ数が増えた場合、ファイルベースルーティングはあった方が可読性は高い
  - React Router でも実現する方法はあるので今ならそちらも検討するかもしれない

余談 - ルーティングライブラリは発展途上

ROUTE 06

- React Router の型安全性の低さを取り上げ、代替するライブラリが多数名乗りを挙げている状況
  - Next.js であっても、App Router でルーティングが要件を満たせずに Pages Router や Remix に切り替える事例もよく見かける(観測範囲)
- フレームワークの乗り換えを検討する場合、**ルーティングライブラリとの結合がボトルネックになる**
  - 先ほどのカテゴライズでの「**強く依存する**」相手となってしまっている
- トップレベルのコンポーネント以外は無秩序にルーティングパッケージを直接 import はさせず、基本的に props で渡すようにすればよかったかも
  - 教訓: React コンポーネントは基本的にフレームワークに依存させずに「純粋」に保つ方が移行容易性が高い

半年経った感想 - チャンク分割

ROUTE 06

😐 チャンク分割についてはやはり考える時間が増える

- React Router の `lazy route` と Vite の `splitVendorChunksPlugin()` だけでは vendor.js が巨大になり、ページごとのチャンク分割ができていない
  - これは我々の実装の問題かもしれない(pnpm workspace で分割したパッケージは node_modules に含まれるため vendor 扱いとなってしまう)
- チャンク分割手段の選択肢は他には `manualChunks` しかない

→ 認知負荷が増えるためまだ `manualChunks` は使わず、課題が出てきた時に対処できるよう Sentry で FCP の監視をしている

2024 年技術選定するとしたら

ROUTE 06

- 同じようなコンテキストを持つ業務アプリケーションの場合、Remix SPA モードをまず検討するのも良いかも
  - 2024 年 2 月に Stable となった
  - ルーティング・チャンク分割の仕組みを任せられ、要件が変わった際に SSR に切り替えることもできる
  - loader / actions を使うのは強く依存することとなるため要検討

まとめ

ROUTE 06

- プロダクトの特徴やフェーズ、取るべきリスク・回避すべきリスクを整理し、最適な技術を選定する
- Vite の選定 == not SSR という選択 && フレームワークを使わないという選択
- React コンポーネントは基本的にフレームワークに依存させずに「純粋」に保つ方が移行容易性が高い
- 今は Remix SPA モードから検討するのが良いかも