Private Outputs

Webゲーム開発

- AirPoker: 某漫画内ゲーム。Next.js + WebRTC + MongoDB
- 王への請願: 有名ボドゲ。Next.js + WebRTC
- Slack CodeGolf: Node.js + websocket on heroku

OSS貢献

- Node.js本体: timers、hidden class最適やらdocsやら
- AI音声合成TTS Style-Bert-VITS2の初期リリース・設計レビューetc...
- AI歌唱合成NeuCoSVC, kotoba-whisper HF pipeline
- その他...

副業

- Kakekomu (ST Booking): CTO(後輩)お誘いで1ヶ月手伝い、質問回答通知機能・jwt暗号化・テスト拡充など
- Meetsmore: CTO(同期)お誘いで1ヶ月手伝い。MongoDBのアプリデータをBigqueryへ(table/partition設計・Digdagバッチ・コンテナ化)
- Smoothy: Googleの営業の方と組んで。飲食店などの予約サービス(ReactNative,サ終)
- Kotoba Technologies: Transformer Modelプロダクト開発のお手伝い中

その他

- 登壇歴: https://github.com/darai0512/talks/blob/master/README.md
- 販売・配布
  - twitch配信コメントのOBS画面反映アニメーション（js+websocket+css）
  - VR用小物 by Blender/RVCモデル/VITSモデル/画像lora: booth販売
- ハッカソン
  - 国内: 目覚ましゲーム(iOS)で特許出願
  - 台湾ハッカソン: 2度参加。英語プレゼン。現地エンジニアとは今も交流あります
- leetcode: See gist
- 資格: 認定スクラムデベロッパー(csd)

職歴: オンライン決済システム（API・SDK）

概要

オンラインでのクレジットカード決済機能（PCI-DSS準拠のAPI・SDK）を提供するBtoBサービス企業で働いた。
（在職中に上場を果たした）

会社・チームの規模

従業員数250名以上のEC事業社の子会社として、`PAY_Dev Section` に所属。

子会社は30名程度でエンジニアは5名(+PCI-DSSセキュリティ専門1名、中途即戦力のみ)程度で推移

期間

2018/12 - 2024/12 (2018/12 - 2019/12 は副業の業務委託契約で本業と掛け持ち)

進め方

主サービスである決済サービス基盤(BE: python gunicorn+pyramid, DB: Aurora Postgresql, FE: vue.js+jinja)とバックオフィス(スタックは同上)の保守運用については、スクラム形式でnotion（以前はasana）でのタスク管理で進めた（コロナ以降はフルリモート）。
エンジニア・ビジネス・CS・マネージャー全員でプロダクトバックログの策定・リファインメントを行い、優先度・ポイント付されたタスクを順に消化していく。
さらに別でPJ毎に集まり、同様の管理で進める。

アドホック作業（redashやBigqueryでのKPI算出・顧客企業からの技術問い合わせ回答）は当番制で担当した。

成果（6点ピックアップ）

Platformサービスリニューアル

- 工数：エンジニア3人・PM2人で4ヶ月ほど
- 役割：プレイヤーとして要件定義からDB設計・BE設計・FE全般の開発
- 課題：PlatformというBtoBtoB機能（弊社 to Platformer to Tenant）をリリースしていたが課題があった
    i. アカウント体系を新しく用意したため、主サービスのコードやデータを再利用できない（cf, モノレポ管理で、主サービスではAPI・そのGUIである管理画面・バックオフィスのそれぞれでアカウントベースのserviceロジックを共通化）

    ii. BtoBtoC（弊社 to Platformer to 個人事業主）で使いたいというニーズに対し、座組やデータ構造が適していない
- 解決：`データ構造を1から再設計し、主サービスのアカウント体系に組み込む`ことで、コードやデータを共通化し、主サービス側の新機能をPlatformでも使えるよう改修しマイグレーションした
    - Platform（BtoBtoB）と主サービス(BtoB)とを比べると前者はアカウント構造がネストしている。メリデメを洗い出し、再帰ではなく、rootのアカウントを既存テーブルに一本化・ネスト分は別テーブルに分け、各リソースとnullableの外部キー制約で紐付けた
    - 分かれていた管理画面も一本化。ログイン後画面やAPI構造に差異は生じた（UIコンポーネントやserviceでのif分岐等が増えた）が、DIやネスト構造のフラッテンなどでカバーし、共通化のメリットが上回った

チャージバック（不正利用時に売上没収する仕組み）自動化

- 工数：エンジニア2人・PM1人で6ヶ月ほど
- 役割：開発リーダーとして企画・要件定義から状態遷移図作成・DB設計・BE設計と開発
- 課題：チャージバック作業は上流（カード会社など）・加盟店と幾度とメールやり取りし、請求書送付して金額回収する、煩雑なフローの手作業だった。担当者も属人化していた
- 解決：
    - `各上流とのやりとりフォーマット策定から参画`しできるだけフォーマットやフローを共通化した。残りの差分は汎用的に読み込めるバッチ処理を構築した
    - 複雑な状態遷移をステートマシンに落とし込んだ
    - 管理画面上に反証入力フォーム（ExcelフォーマットのUI化）を用意するにあたり、Excel入力のほうが楽、とならないようデザイン工夫や自動入力機能・バッチ取り込みを拡充
    - 売上金から相殺するロジックを用意し、取りっぱぐれや自動化の難しい請求書送付を減らした
- 開発においては、業務委託エンジニア2名(3ヶ月おきに別人1名ずつ)に対して自分がリードした。ドメイン知識の差を埋めるべく、ビデオ通話多めの気さくなやりとり・github上で認識違いが減るよう専門用語ではなく具体的なコード変更で示すなど工夫した

マルチアカウント化+PCI-DSS+Vue.js 2系EOL対応+デザイン刷新

- 工数：エンジニア自分のみ・デザイナー1人・PM1人で5ヶ月ほど
- 役割：プレイヤーとして企画・要件定義からDB・BE・FEの設計と開発
- 課題：以下の課題全てに対応
    i. 1企業1アカウントの割り当てから、IAMのような1アカウント内でロールで権限管理された複数ユーザーがそれぞれログインできるよう管理画面を改修する
    ii. PCI-DSSの要請から管理画面に操作ログを用意する
    iii. Vue.js 2系が`EOLを迎えるため3系移行`する
    iv. 管理画面デザインを刷新する。ただし`デザイナーとはfigmaでやりとりできるが、vue.jsは扱えずローカル環境構築も難しい`。
- 解決：
    - 平易なERD、pyramidインターフェースの拡張で設定的に権限管理できるようなロールテーブルのロジック
    - 結果、100以上のエンドポイントのコードをいじることなく、デコレータでの設定追加のみで権限管理と操作記録を実現
    - 機能追加・デザイン変更がPMやデザイナーと共有できるよう、常にstaging deployが自動で走るようにし、画面ベースでコミュニケーションした。これはPMを安心させ、素早いバグ報

告・仕様の見直しに役立った
    - Vue.js 2->3移行は最小限で達成させるため、2系資産のOptions APIはそのまま採用
    - ネックはライブラリ対応：VeeValidateというライブラリの2系を数百箇所以上利用しており、Vue3系対応のVeeValidateは4系からで2->3->4の間には多数の破壊的変更があった
    - `VeeValidate2系を利用したまま、ライブラリ内部を改変することなく、Vueの初期化方法だけを3系対応するワークアラウンドがないかコードから探した`
    - 結果、非公開メソッドやプロパティを拡張+mixinを利用することで、既存コードとVeeValidate2系のままVue3系を適用できた
    - その際、工夫をしないとinput/changeイベントなどとnexttickのタイミングがズレてリアルタイムバリデーションが動作しない。イベントループへの知見があったので、適切なnexttickがなされるよう初期化の工夫で対応できた
    - この移行方法は根本的ではない（Vueの脆弱性対応に追従できてもVeeValidateの脆弱性が見つかったら片手落ちである）が、VeeValidateのバージョンが固定されること（そのバージョンにその時点で脆弱性報告はない）、今回は1からQAを行えること、PCI-DSS監査期だったためペネトレーションテストもできること、出口戦略ガイドライン策定（1package内で2系と4系が共存できることを確認し、新規にはVeeValidate4系などを使う）でカバーした
- 残存課題：デザイナーとのやりとりの難しさ
    - 環境構築・Vue.js学習などのコストが壁で、HTML・CSS・JSを別リポジトリで管理し自分が分解してVue.jsへ取り込んだが、デザイン差し戻しで変更されるとその差分をどこに反映するべきか見失いやすかった

高度技術課題への対応（ex, 独自拡張したcryptography/pyopenssl脱却）

- 課題：当時のpyopensslではApplePayの証明書検証（X509Extensionのoid取得・ECDH derivationなど）に対応しておらず、前任者がApplePay決済対応のためにforkして利用していた。その後OSアップデート・PCI-DSS準拠のために、古いversionでforkされたcryptography/pyopensslからの脱却が必要だが、前任者が去って周辺知識を持つ人間がいなかった
- 解決：
    - まずforkして足されたコードを読み解き、次にそれぞれの最新版のコードを読んで比較した
    - ex, X509Extension対応は最新版の特定libsslバージョン併存下のif-defマクロ付きで組み込まれており、条件を満たせばforkで追加した一部を捨てられる、など
    - 細かくテストを書き、レビューワーに噛み砕いて証明書検証・暗号ロジックなどをシェア。forkを捨て本家に追従できるようになった
- 他、urllib3のTLSハンドシェイクタイムアウト時の例外が広範囲を同じ例外クラスにまとめている問題など...etc

コンテナ化とインフラ刷新

- 工数：エンジニア自分のみで技術調査・基礎構築を半年、その後エンジニア3人・セキュリティ専属1人・PM1人で仕上げ3ヶ月
- 課題：PCI-DSS準拠の作業（常にOSアップデート等）を減らすため、自前ec2インスタンスから脱却し以下を実施
    - Fargateによるコンテナ化
    - terraformによるインフラコード管理
    - 自前nginxをやめcloudfront->ALB->ECSという構成へ

    - アウトバウンド厳格化（全ての外部通信はproxyを通す）
- 解決：
    - workspace（マルチproviderとmodule分割）を使ってstaging/production同一構成となるようterraformのディレクトリを構成
    - nginx代替としてパスリライトやヘッダー付与およびLuaで実装していたrate limitなどをcf functions+terraformへ
    - anisibleデプロイからjsonnetで記述したecspressoによるデプロイへ
    - コンテナ化に伴うアプリケーションの改修：fs脱却/logging設定/環境変数ベース（tfstate/secret managerから設定値を読む）
    - requests/urllib3/各種SDK(sentry,bigquery,onelogin...)にそれぞれプロキシ設定。多くはHTTP_PROXY環境変数で対応できるが、内部通信でプロキシ不要のboto3も影響を受けるので工夫

※初めの数ヶ月はEKS(kubernetes)想定でhelm+devspaceで構築。しかしAWSサポートチームからのk8sの長期運用の難易度の高さの警告・EKS on Fargateでは実現しにくい機能などが見つかり、ECSへ方針転換した

SDK保守運用と新規開発

JSカード情報入力ライブラリ

最新のPCI-DSS要件を満たすiframe型(加盟店ドメインを経由しない)電子商取引jsライブラリを1人1.5ヶ月で新規開発

- 世界規模のライブラリをリバースエンジニアリングで参考に
- material designと合わせたサンプルなどドキュメント拡充させ、要件を満たさない以前のJSからの移行をサポート。完全移行に成功

動作ブラウザを制限できる管理画面とは異なり、エンドユーザーの様々な環境下での動作保証に苦労した

- playwrightでchrome/safari/ffのテストを自動化
- iOSやAndroid、windowsはBrowserStack
- それでもLineアプリ内ブラウザのみでwindow.openが動作しない・3DSecure対応のUIで国コード絵文字がwindowsのselect.option内で適切に表記されない、などエッジケース多数、地道に改修
- ビルドした成果物がスコープに収まっている（変数がグローバルになっていない）ことの検出は抽象構文木でテスト

mobileカード情報入力ライブラリ

Swift / Kotlin / react-native + Expo / flutter を用意。モバイル専属エンジニア1名のサポート・レビューという形で貢献。

Server SDK

Go / Node.js typescript / PHP / Perl / Java / Ruby / Python は全言語を扱える唯一の存在として保守運用。

- openapiからの自動生成ではないため、リッチな機能を意識した
  - レートリミット時のオートリトライ・actionsによる自動publish・循環参照を回避したオブジェクト構造の体現...etc
- 特にGoに力を入れた
  - ゼロ値問題への対策
  - バージョンサポート制約の都合でgenericsを使えず、リスト構造の中に各リソース構造体が入るケースの表現に難儀した
- 自作の非公式rust SDKやOpenAPI v3.1を公開

職歴: メディア売買Webサービスローンチ

概要

二次創作物（画像・動画・小説・vrmなど形式任意）を売買できるサービスを、構想から2年総勢10名で開発したがリリースに至っておらず、
力を貸してくれと声がかかり、本業と並行して副業業務委託としてサービスローンチさせた。

チームの規模

(ジョイン時は)創業者と開発者として大学生2人だけだった。

期間

2019/10 - 2019/12 (副業)

役割と果たしたこと

開発リーダーとして、企画・デザイン整理と要件定義をしつつタスク化、エンジニア他2人をリードして1ヶ月半でリリースさせた。

詳細

まずpush内容がslack通知される・stg環境に反映されるCI/CDを構築した。
Trelloに残っていたタスクチケットをみて、全員が副業なのでスクラム的なやり方は適切に思えるが、開発人数に対するタスク粒度の不適切さが見受けられた。
アジャイルは開発速度を早くする手法ではないし、リリースを急ぎたい中でチケット消費の仕事した感は邪魔になる。
絶対に必要な画面や機能を、明文化されてない要件などは開発者任せにしてとにかく開発し、pushの通知で創業者がstg確認し、QAも兼ねて仕様確認をしてもらうルールにした。
(PR bodyがslack通知されるので自身で決めた仕様は明記する。終盤はPRさえ時間が惜しかったのでcommit messageに変更)

BE(gRPCのマイクロサービス。go言語でのメディア処理・ストレージ・ORM API・dispatcher・その他)は優秀な学生が開発しており、まずはgRPCを書き上げを手伝って全てのサービスが動く状態とし、レビューとバグ修正以外は彼に任せた。
FE(GraphQL, Nuxt.js)が経験の少ない学生だったので、画面と機能とで分担しつつ、少しずつVueやGraphQLを手解きした。

とにかく作った：マルチメディアアップロードフォーム・paypal決済と口座登録フォーム・検索機能・ログイン...etc

当時開放されていたYahoo!Japanのロッジに集まっていたので本業後に合流し、閉館後はファミレスで開発する日々だった。1ヶ月後の深夜3時にジョイフル赤坂店でリリース、乾杯の喜びは今も覚えている。

職歴: 部署横断言語（Node.js）サポート活動

所属と期間

2018/08 - 2019/12
大手IT企業の正社員として、の退職まで兼務

概要

社内の利用推奨言語として当時Node.js(10系時代)が採用され、Node.jsコミッターの方が入社したことから、部署横断でNode.js開発をサポートするチームが発足し、兼務として活動した。
利用者向け講習会・各部署ごとの個別課題の相談会・社外広報・チーム自身の能力アップを図ったコード輪読会などの活動をした。

活動実績

利用者向け講習会・相談会

expressでAPIサーバーを作ってみる、などではなく、Node.jsの核である基礎技術としてModule/Buffer/Events/Streamの4点に絞り、それを実践的に学んでもらうことで問題発生時に自身で解決できる人材となれるようなプログラム（3時間x4回、定員20名前後）を組んだ。オリジナルの資料と問題集は講習後アンケートで好評を得て、複数回プログラムを実施できた。
相談会では、シングルスレッド故のパフォーマンス・スケール方法（設計次第で解決できることを丁寧に説明)、PHPから乗り換えたらパケットロストがやメモリリークが発生（tcpdumpなどで調査しnode.js自体が原因ではないことを説明)、などがあった。

外部広報

- https://nodefest.jp/2018/schedule.html#conference-2-3 <- 自分が登壇
- https://techblog.yahoo.co.jp/advent-calendar-2018/yahoo-frontend/
- https://en-ambi.com/itcontents/entry/2019/08/08/103000/

コード輪読会

上の資料を作るため、Node.jsのコードを読み合わせた。
特にEventループ理解のためにlibuv/v8などをコミッターの方の解説の下で理解できたのは最大の財産となった。
その後も第二回講習会で通信・暗号周りを取り上げるため、TLS通信・HTTP2・暗号化などの基礎技術をNode.jsの実装ベースで学んだ。この知識は広く業務で活かすことができ、エンジニアとして最も成長できた期間だった。

https://techblog.yahoo.co.jp/entry/2020072830014370/

職歴: DMP（データマネジメントプラットフォーム）設計・開発

所属と期間

大手広告配信会社（自社メディアもあり）の正社員として、2016年〜2019年の退職まで担当

課題

ターゲティング広告販売において、競合他社のデータ利活用が盛んになる中、自社営業や代理店が広告主への提案力を高められるBIツールを作りたい。

大量のデータ（メディアへのアクセスログ1日分でHDFS圧縮して1TB以上、20億PV）があり、それをHadoop管理できているが、「競合他社のユーザーに自社製品を認知させたい。どれくらいリーチしてどういう人たち（デモグラ・普段の行動履歴など）にあたるのか？」と訊かれてその場でMapReduceを繰り返すのは時間がかかりすぎる。
また、広告主側のビッグデータと掛け合わせた分析もニーズとしてあった。

解決

サードパーティデータも連携可能で、数秒で結果を返しリッチに可視化するツールを、Hive+Solr、React+d3で開発した。

ex,
広告主の「自社ユーザーリストA」と、自社データの「競合他社に興味関心がある（検索・BT）ユーザー群B」をB and not Aと掛け合わせたユーザーリストに配信したい。
配信前にリーチ数やデモグラ情報が即座にビジュアライズされ、リーチ数を増やしたければデータの期間を伸ばしたり他のセグメントを足す。逆に東京都の女性だけに絞ることもできる。

開発における役割

エンジニア20人のPJにおいて、6つのコンポーネント（マイクロサービス）を開発。
要件定義とシステム設計（最初は一部のコンポーネントの外部設計がメインだったが、途中から全体のグランドデザインを担当）と、2つのコンポーネント（ETLを行うサービス、可視化を行うUIサーバー）の開発・テスト・保守運用を担当した。ETLコンポーネントではPMをした。

成果

設計・運用面

- 全体として単一障害点をうまない疎結合なコンポーネント図になるようとことん協議。初めてのマイクロサービス・APIオーケストレーションだったが完成できた

- 全体をみれるエンジニアとして各コンポーネントの設計上の課題を察知・調整
  - 先行して全体で使う技術を勉強しておき、全体設計の段階から議論をリードすることで、コンポーネント毎の開発に移ってからも信頼感を得ながらて進捗管理・ヘルプ参加できた。納期が押した際には、全体最適を意識して機能の取捨選択の議論に参加できた。
- バックエンドでは、長期間変わらない属性データ（居住地・性別など）・短期間で変わりうる属性データ（デバイス・興味関心など）・一回一回の行動データ（検索・サイト閲覧など）と集計サイクル都合で分割しつつ、UIではこれらの違いを意識せずに一律に選択できるよう、デザインや異なるデータ種別の抽象化に注力した
- サードパーティデータのデータ構造は容易に変わりうる。Hiveテーブルのカラム型が修正されると、UI都合でデータの選び方などに影響が出てしまう。逆にUIにおけるセグメント限度数や集計期間制限は、HiveやSolrのキャパシティ・パフォーマンスの知見がないと設定できない。マイクロサービスとして担当者を分けていたので、入り口のデータ受け取り担当と出口のUI担当が意思疎通が必要だが、システム全体の理解する1人として、調整のキャッチアップや橋渡しができた

開発面

PHPからNode.jsへ

PJ内のエンジニアがphp経験のみだったが、下記利点からNode.jsでの開発を促した。

- セッション数に制限がない
  - リーチ数計算はDWH(solr)へのAPIで取得する。基本的に数秒で返るが、期間が長いなど、重い条件式ではキャッシュが効かないと数十秒かかる。php + apacheの構成だとリクエストの間にapacheがpsを占有するため、UIへの接続数がスケールしなくなる(MAX_CLIENT< 32767)
- UIとのシームレスさ
  - UIとFEとでバリデーションコードを共有
  - UIのreact/reduxはデザイナーが担当するが、運用はエンジニアが行う。以前から、JSが得意なエンジニアが少なく、UI運用が課題だった。Node.jsでJSに慣れることは100%ではないにしろ心理的障害を取り除けたと思う
- 自分が全社Node.jsサポートに兼務していたため、ライブラリや言語自身の問題対応に迅速に対応できた
- UI・Hive・Solrが会話する方法として、PEG.js（構文解析器）で独自の式（先述の例でいうB and not Aのような式）を作成した
  - JSなのでパースと式文字列生成をFE・BEで使いまわせる
  - 式自体は独自化しすぎないようSolrクエリに寄せた

サービス間はAPI仕様書ベースの会話に

API Blueprintを使ったAPI仕様書の作成方法を共有し、PJ全体に広めた。
markdownをaglio/drakovで仕様書・mockサーバーも生成される。

openapi（当時swagger）と悩んだが、非エンジニアのPMも見る・触ることもあり、markdown形式で記述ができることを選んだ。
2024年の今振り返れば、API Blueprintは廃れており、swagger(openapi)仕様で作った方が長期的に活用できたと思う。
しかし当時必要だったことは、FEとBEとの会話をAPI仕様書ベースにすること・誰でも仕様をかける、アクセスできること・ドキュメントとモックサーバーが同期していることであり、API Blueprintで十分

な効果を得られたと思っている。
APIの外部公開予定はないし、当時のswaggerの自動生成SDKのnode.jsコードもあまり使い物にはならなかった。

苦戦したこと

- データにおける日付や期間の扱いが難しく、コンポーネント間で連携するのに難儀した
  - 前提として、partitionは一日単位の日付（基本的にlog_date）で切る
  - ETLのHiveによる前処理において、ログの中身の日付（イベントを起こした日）とデータ整形が完了した日付は必ず一日以上ずれる。特に複雑な条件を施すと生成に2日以上かかる。また、サードパーティデータは複数の日付のログが混ざっていることが多い。
  - 日付の指し示す意味が異なると正しい結果は得られない。UIで表示される日付の意味がデータ毎に変わりうるので、工夫が必要だった
  - こうしたメタ情報を一元管理するコンポーネントが必要と気づき、追加開発した
- データを機械学習に使う場合、季節変動を考慮するために最低一年間のデータが欲しい。しかしDWH（solrのオンメモリで利用）には長期間は保持しきれず、試行錯誤が必要だった
- HiveはSQLではない
  - クエリの構文解析においてSQL的な直感とは挙動が異なる点があり、事前にコードを読むべきだった。クエリがどうMapReduceに変換されるか想像できないとパフォーマンスへの影響が大きい
  - またSQLインジェクションはライブラリで防げるが、独自の計算式からHQLに変換する際には自身でインジェクションに気をつける必要がある。実際リリース後、UIから渡された%の文字列がエスケープできておらず、条件式が緩くなり処理遅延につながった（何十時間経ってもMapReduceが終わらない）

職歴: 機械学習用メタデータ生成システム刷新

所属と期間

大手広告配信会社（自社メディアもあり）の正社員の入社初仕事として、2013年末〜2015年。保守運用は退職まで担当

背景

自分の所属するチーム5人で、年10億以上売り上げる行動ターゲティング（以下BT）広告のユーザーリストを作っていた。
Gradient boostingベースの機械学習ロジックで生成するに際して、学習に使うデータにメタデータを付与して精度を上げていた。
そのメタデータは他部署で人力で行っていたが、レガシーで運用も非効率だったため、ツールや運用方法を刷新した。

役割と開発物

PMと開発1人（=自分）のPJで、以下の開発と運用を担当した。

A. メタデータ作成（行動データのタグ付け）のUI/FE/BE（ex, 「ハワイ 旅行」という検索ワードに海外旅行タグを、ニンテンドーDSの広告クリエイティブにゲームタグを付与）
B. MapReduceで行動ログとAの辞書をlookupさせ、ルールベースのモデリングを行って興味関心別のユーザーリストを作成するシステム

やったこと・成果

Aについて

- Oracle 9iから12cへのマイグレーション
  - 刷新前システムは、OracleのDatabase Link(Materialized View)経由で取得し、整形・加工はPL/SQLで行っており、処理が極めて遅かった
  - PL/SQLの処理を、誰でも読みやすいANSI SQLに置き換えたり、言語側に移して単体テスト可能な状態にした
  - 同時にDB設計も見直して速度改善に努めた
  - 大改変だったのでPMと密に連携して、コードを熟知した上で再設計と必要データのマイグレーションを慎重に行なった
- isilonからのデータマイグレーション
  - Bへの辞書データの配布はisilon経由で行われていたが、マウントは実機でしかできない制約があり、スケールできない・実機破損時のリスクなどから、s3互換の内製ストレージへ切り替えた

- 単体テストのないperlバッチを、上の新DB設計に合わせて処理を改変し、単体テスト・スモークテストを用意
- さらに新しい機能面はpythonのDjangoを使って書き直した
  - pythonを選んだのは、入ってくる新卒がpythonを使えるケースが多かったため
  - Djangoを選んだのは、django-noseがテスト毎に自動で（テスト用）テーブル作成をし、SQLを含めたテストを行って終了時には破棄までしてくれたから
- UI改善では、UIがドキュメントのないPHPの独自フレームワークだったため、刷新前にcasperjsでE2Eテストを作成した
  - しかし工数がかかりすぎた。当初はnightmarejsの2系(electron)が直感的なAPIで動作も速く、Mac上でテストコード開発し独自CIにのせて運用していた。そのタイミングで全社CIが用意・強制され、選べる環境がcentosの6 or 7のみで、当時のelectronが構築できず、e2eコードを捨てることになった
  - casperjs(phantomjsで動く)で書き直したが、程なくphantomjsのメンテ終了が発表された。当時はPuppeteerもPlaywrightもなく、Seleniumなどを選ぶのが次善だったと思う
- メタデータ付与を行う地方のチームとは何度も出張やビデオ通話でコミュニケーションを取り、UI改善に活かした。またデータ分析を経て、業務改善施策を行った
  - ヒアリングの中で、歴史的経緯で入稿順にメタ付与作業をしていることを知った。そのため新しい広告でメタ情報が付与されていないものが大量にあった。
  - BTの学習への影響度は広告クリエイティブや動画閲覧などが最も重みがある。そこで開発の傍ら、データ分析を行ってみた。クリック数上位の広告を取り込んだ場合の最終的な効果（リーチ数の増加予測）が高いを出してみたところ効果が高いことがわかったので、クリックの多い順に変更する機能をつけ、作業してもらう提案を採用してもらった

Bについて

- 学習データを増やすため、新たな広告データを加えるためのデータ提供元との調整・MapReduceでの新しいログパースとlookup・機械学習のルールベース追加・数値検証を行った
  - 高速化・リソース削減のために従来のHadoop処理からHive(OnTez)へ移行
  - oozie coordinatorのrerun機能が運用時に重宝するため、Hiveコマンドもoozieで起動させる実装
  - HDFSへのアップロード処理・HDFSからs3への転送処理をREST APIであるHttpFSコマンドで実装

以上