職務経歴書

基本情報

key	value
名前	岡本 侑樹
住所	東京都
年齢	25歳（1999.1.8生まれ）
ポジション	フロントエンドエンジニア、バックエンドエンジニア、SREを幅広く
エンジニア歴	2年半（3年目）
Zenn	https://zenn.dev/felipe
Twitter	https://twitter.com/okamoto_uuid
Speaker Deck	https://speakerdeck.com/yuki_okamoto476
TOEIC	860点
出身大学・学部	同志社大学 法学部
その他	University of the PeopleでComputer Science専攻中（2023.11〜）

実務で経験のある言語・フレームワーク・ツール

フロントエンド

React | Next.js | TypeScript | Nuxt.js | JavaScript | Jest | React Testing Library | Playwright | Storybook | GraphQL | Apollo Client

バックエンド

NestJS | TypeScript | PostgreSQL | Jest | GraphQL | Apollo Server | GitHub Actions

インフラ

GCP | Terraform | Datadog

経験の概要

フロントエンド

React、Vue（2系）ともに経験あり。要件定義、基本設計、詳細設計、開発、テスト、コードレビューの経験あり。

バックエンド

NestJSで開発経験あり。要件定義、DB設計、API設計、基本設計、詳細設計、開発、テスト、コードレビューの経験あり。

インフラ

GCPの様々なマネージドサービス（Compute Engine, Cloud Build, Cloud Functions, Cloud Scheduler, Cloud Storage, Cloud Run, Cloud Pub/Sub Cloud SQL, Cloud Logging, Elastic Cloud, IAM, Load Balancing, Secret Manager, Workflows） 使用経験あり。

業務経歴

レバレジーズ株式会社（2022.12〜現在）

プロジェクト概要

HR系のSassの開発（マイクロサービス）

役割

（2022.12〜2023.11）サービス共通の基盤プロダクトチームの開発メンバー
（2023.11〜2024.2）上記プロダクトチームのリーダー
（2024.2〜）新規プロダクトの開発メンバー

担当業務（経験した業務）

フロントエンド、バックエンド、インフラを幅広く担当。
歴としてはフロントエンドが長いが、最近はバックエンド、インフラ中心。
開発だけでなく、チームのマネジメント（最大で4人のメンバーをマネジメント）、企画、要件定義、プロジェクトの管理なども経験あり。
サービス共通の基盤チームのリーダー時には共通基盤という性質上、他プロダクトの影響範囲調査、他チームとのコミュニケーション、タスク依頼、スケジュールの調整も行っていた。

使用技術

React | Next.js | TypeScript | NestJS | GraphQL | Terraform | GCP

発揮したバリュー1

IP制限機能を新規開発するにあたって以下を主導で行なった。

- 競合プロダクトの調査から入り要件定義。プロダクトマネージャーとディスカッションして仕様を策定。
- 画面設計を行い、デザイナーとディスカッションしながら画面作成。
- 仕様から必要なDB設計、DB作成。
- フロントエンド、バックエンド、インフラでそれぞれ必要なタスクを分解しメンバーにアサイン。
- アサインした開発のコードレビューや進捗管理。
- ビジネス側や他チームとのスケジュールやデプロイの調整。

発揮したバリュー2

【課題・状況】

フロントエンドでCSSのデグレが起きたり、レビュワー・レビュイーともにデグレの人力確認で工数がかかっていた。

【行動】

仕組みで解決するためにビジュアルリグレッションテストを導入。
具体的には、Playwrightで全画面のスクリーンショットを撮影、スクリーンショットをGCS（Cloud Storage）に保存、最新のブランチのスクリーンショットと比較し差分があれば検知してくれる仕組みをCIで構築。

【結果】

デグレがほぼ0に、デグレ確認の工数も0になった。

発揮したバリュー3

マイクロサービス配下の新規プロダクト開発において以下を主導で行なった。

- BFFの構築
- GCP・Terraformを用いて開発環境とテスト環境の構築
- CI/CDパイプラインの整備
- エラーログのslackへの通知
- Compute Engineの高可用性対応

発揮したバリュー4

【課題・状況】

①長年アップデートされていないことで、脆弱性の指摘やサポートがもうすぐ切れるライブラリが複数存在していた。
②アップデートしようにも他のライブラリが依存、そのライブラリにも別のライブラリが依存しているなど、依存関係が複雑に絡みあっていて（ex. peer-dependenciesのエラーが10個ほど出る）容易にアップデートできない。
③アップデートすると破壊的変更が起きる（ex. TypeORM 0.2→0.3、Apollo Server v2→v4）。 上記の理由からバックエンドのあるマイクロサービスで技術的負債となっていた。

【行動】

①を受けてライブラリのバージョンアップに着手。
②に対しては、ライブラリをアップデートしたときに吐き出されるエラーから、どのライブラリがどのライブラリに依存しているか、どのバージョンなら解決できるかを紐解いて整理。
アップデートする際に対応していないライブラリが出てきた場合は、そのライブラリの代わりに別のライブラリをインストール。
これを繰り返して全ての依存関係のエラーを解消。
また、更新が止まっていそうなライブラリが存在した場合は、将来的なリスクも鑑みライブラリを使わない独自実装に切り替えた。
③については、ライブラリのGitHubや公式ドキュメントのマイグレーションガイドから変更箇所を読み解き、既存のコードに合う形にコードを修正。 また、アップデートにより変更があった箇所や修正していく中で詰まった箇所は、解決法とともにドキュメントにまとめ他チームに展開することでナレッジの共有をおこなった。

【結果】

1ヶ月ほどかけてライブラリをバージョンアップし、脆弱性を大きく減らすことができたともに技術的負債の解消に貢献した。

発揮したバリュー5

【課題・状況】

2000件のデータを取得するAPIの実行時間が25秒ほどかかっておりUXが低下している箇所があった

【行動】

フロントエンドとバックエンドの両方から改善を試みた。 フロントエンドでは、APIを実行している関数をReactのhooksを用いて最適化、使用していない値をそもそもAPIで取得しないように修正することでオーバーフェッチングを解消。 バックエンドでは、APIで取得している値の中でいくつかn+1問題が起きていたので、dataloaderを用いてSQLのチューニングを実施。

【結果】

フロントエンドの改善で約5秒、バックエンドの改善で約10秒ほど、計15秒ほど実行時間を短くすることができた。

発揮したバリュー6

【課題・状況】

セキュリティの観点から、事業部内の全チームでファイルアップロードの実装方法の変更と新規のマイクロサービスへの移行が必要になった。

【行動】

自チームの移行・実装を担当。 具体的には

- FE（フロントエンド）、BE（バックエンド）、インフラ含め影響範囲を洗い出し
- FE、BE、インフラをどの順番でどのような実装に修正すればダウンタイムなしで移行できるか整理しタスクを細分化
- 整理した事項に基づき、以下を全て一人称で担当
    - インフラ：移行にあたって必要なTerraformの修正やGCPの設定を追加
    - BE：新規のマイクロサービスへの移行、APIの修正・新規作成
    - FE：修正したAPIの繋ぎ込み部分を修正

発揮したバリュー7

【課題・状況】

フロントエンドのリポジトリにおいて以下の状況が発生していた。
①脆弱性の指摘が108個あり長年放置されていた。
②create-react-appを用いておりアプリケーションを起動するのに45秒ほどかかっていた。

【行動】

以下の打ち手を講じた
①クリティカルな脆弱性は指摘されていなかったがいつかは解消しなければならないということで、脆弱性が指摘されているライブラリをアップデート。更新が止まっていそうなライブラリについては別のライブラリに移行したり、ライブラリを使わない独自実装に切り替えた。
②不便というほどではないがもっと高速に起動できれば開発効率が間違いなく上がると考え、ビルドツールをcreate-react-appからviteに移行。

【結果】

①108個指摘されていた脆弱性を0にできた。
②アプリケーションの起動が45秒から10秒ほどに短縮でき自分だけでなくチームメンバーの開発効率も上がった。

発揮したバリュー9

【課題・状況】

- プルリクエストのレビューで、無駄な処理の指摘や書き方の思想の違いの議論などで時間を使っている時がある
- コードの可読性や保守性を高めるためにコーディングルールを作ろうにも結局参照されない可能性が高い（実際に見られていない）
- 品質向上というチーム目標のためにバグを生むようなコードを排除してバグを減らしたい

【行動】

人力ではなく仕組みで解決するためにeslintのルールを強化。
具体的には、数あるライブラリの中から、バグを生みそうなコードや可読性・保守性の低いコードを検知してくれることに適したeslintのライブラリを選定し導入。
メンバーからも意見を伺い、その中から今のリポジトリに適したルールを抽出し、30ほどのルールをeslintに追加。
これにより、ルールに反したコードを書くと開発時にエラーが出るようにし半強制的にルールに従わせるようにすることで、レビュー工数の削減とコードの可読性・保守性・品質向上を図った。

株式会社メンバーズ（2021.4〜2022.11）

プロジェクト概要

シニア向けサービスの開発

担当業務

シニア向けのWebサイト、Webviewページ制作

使用技術

Nuxt.js

発揮したバリュー1

【課題・状況】

APIから取得した情報を表示する機能を実装する際に、以下の課題があった。

- リリースに間に合わせるために、実装方針をある程度固めて実装していたが、途中でAPIが改修され、仕様が大きく変更された。
- 例外パターンの仕様が存在しない。そもそも考慮されていない。

【行動】

このような課題がある中で、まずはリリースに間に合わせることを優先するために、最低限必要な例外パターン時の仕様についてディレクターとデザイナーに提案。旧仕様のロジックを流用しながら、最低限必要な機能を実装してなんとかリリースに間に合わせることができた。
そして、リリース後に以下の改善をおこなった。

- エラーパターンを含め全ての例外パターンを実装することで、エラー時のユーザーのとるべき【行動】をわかりやすくした。
- 一度APIの通信に成功しデータを取得できた場合は、そのデータをキャッシュとして保持し、2回目以降そのAPIにアクセスしないように実装を変更することで、APIの負荷を軽減。これにより、ユーザーの通信に伴うストレスも軽減。
- 子コンポーネント側にAPI取得のロジックを書いていたが、親コンポーネント側にロジックをロジックを移行することで、より可読性の高いコードにした。

発揮したバリュー2

【課題・状況】

チームで毎日行うMTGのために会議室を予約する必要があったが、それをバックオフィスの人に依頼して毎日手動でOutlookから予約してもらっていた。

【行動】

この作業を自動化するために、Power Automateというローコードツールを用いて、毎日特定の時間になったら自動でOutlookの予約のメールをくるようにした。

【結果】

毎日手動でおこなっていた1分ほどの時間を削減することができた。