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、インフラ を幅広く
エンジニア歴	3年（4 年目）
Zenn	https://zenn.dev/felipe
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

言語・ツール	経験年数
JavaScript	3年
TypeScript	1.5年
React（Next.js）	1.5年
Vue（Nuxt.js）	1.3年
GraphQL	1.5年
Jest, React Testing Library, Playwright	1.5年
Storybook	3年

バックエンド

言語・ツール	経験年数
TypeScript	1.5年
NestJS	1.5年
PostgreSQL	1.5年

言語・ツール	経験年数
Jest	1.5年
GraphQL	1.5年
GitHub Actions	1.5年

インフラ

言語・ツール	経験年数
Google Cloud	1年
Terraform	1年

Google Cloudで使用経験のあるマネージドサービス

Compute Engine, Cloud Build, Cloud Functions, Cloud KMS, Cloud Scheduler, Cloud Storage, Cloud Run, Cloud Pub/Sub Cloud SQL, Cloud Logging, Cloud Tasks, IAM, Load Balancing, Secret Manager, Workflows

経験の概要

フロントエンド

要件定義、設計（画面設計含む）、開発、テスト、コードレビューの経験あり。

バックエンド

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

クラウドとIaCを用いたマネージドサービスの使用やインフラの構築経験あり。

企画

機能の企画経験あり。

マネジメント

4人の開発チームリーダーとしてマネジメント経験あり。

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
（2024.2〜2024.4）新規プロダクトの開発メンバー
（2024.4〜）上記プロダクトの保守＆複数プロダクトにまたがる大きな機能の開発

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。
今のところフロントエンドの経験が長いが、最近はバックエンド中心。
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 | Google Cloud

発揮したバリュー 1

【課題・状況】

サーベイ（アンケート）を全従業員にメールで配信する機能がもともとあったが、メールだけではサーベイの配信に気づかない従業員がいたのでチャットツールでも通知が飛ぶようにしたいという要望が複数顧客からあった。

【行動】

実際にこの機能を実現するために企画、要件定義、画面設計、基本設計、開発、リリースを主導で行なった。具体的には以下。

- 【企画】そもそも本当に必要かどうか考え直し、以下の理由からやる価値があると判断して PdM（プロダクトマネージャー） と相談して自ら企画。
    - サーベイの回答率を上げることはプロダクトの価値を存分に発揮するためには必須であること
    - 営業が商談している動画を見て実際に複数の顧客から要望がありビジネスインパクトが高そうなこと
    - 海外の文献やデータを調査して実際にチャットツールと連携するとサーベイやアンケートの回答率が上がったデータが複数あること
- 【要件定義】できるだけ早く機能をリリースして顧客に価値を届けるために以下を行いながら要件を定義。
    - 顧客のターゲット層が IT 企業が多く、Slack 通知をして欲しいという声が実際にあったことから、初期は チャットツールの中でも Slack だけにスコープを絞った。
    - PdM から欲しいと言われた、競合が実装しているいくつかの機能について、自分なりに本当にその機能が必要かどうか考え直し初期リリースでは不要と考えたものについては PdM と交渉することでスコープを削りクリティカルな機能に絞った。
- 【画面設計】自分でワイヤーフレームを作成し、それをもとにデザイナーに依頼。デザイン作成後に PdM 含めてディスカッションして画面作成。
- 【基本設計・開発】他チームのプロダクトも関係する部分であったが、他のチームメンバーが忙しそうだったので BE （バックエンド）周りを 1 人称で担当（FE は他のメンバーが作成したものをレビュー）。具体的には以下。
    - Slack App の作成

    - Slack App と連携する際の認証周りの実装
    - Cloud KMS を用いたアクセストークンの暗号化
    - Slack APIを用いた通知の設計・実装
        - 工夫した点；数千人ほどのSlackユーザーに予約通知を送る際にSlack APIが提供する予約通知用のAPIではrate limitが厳しくタイムアウトになってしまうので、Cloud Tasksとrate limitがゆるい即時通知用のAPIを組み合わせて時間差で通知するような設計にすることで大量のユーザーに対しても通知を送れるようにした。
    - DB 設計・作成
    - API 設計・開発
    - 上記開発に伴う Terraform の作成・修正
- 【リリース】開発完了後に他チームやビジネス側とのリリースのスケジュール調整や本番リリース、リリース後の不具合対応を1人称で実行。

発揮したバリュー２

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。PdM 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３

【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画面単位のビジュアルリグレッションテストを自ら導入。
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】

やってみたが、画面全体のスクリーンショットの撮影だと CI の実行が遅いこと、結果が安定しないことからあまり有効的には活かせなかった。ただ、コンポーネント単位で留めておいた方が有用であるという知見は得ることができた。

発揮したバリュー４

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

- Google Cloud・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５

【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 Google Cloud 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】

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 8

【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。