職務経歴書

基本情報

key	value
名前	岡本 侑樹
住所	東京都
年齢	26 歳（1999.1.8 生まれ）
社会人歴	4 年
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
在籍大学・学部	社会人大学生としてUniversity of the Peopleで Computer Science 専攻中（2023.11〜）
その他	1 日密着動画

経験の概要

フロントエンドエンジニア → フルスタックエンジニア → 開発リーダーを経て、現在はプロジェクトマネジメントやプロダクトマネジメント、技術営業を中心に行うプロダクトエンジニア

プロダクトマネジメント、プロジェクトマネジメント、その他上流業務

経験	経験年数
ユーザーヒアリング	0.5 年
顧客折衝（要件の調整）	0.5 年
法人営業（テレアポ、商談）	0.5 年
企画	0.5 年
要求定義	0.5 年
要件定義	1 年
開発ディレクション	1 年
プロジェクトマネジメント	1 年
チームマネジメント	0.3 年
メンバーの教育・サポート	0.5 年

経験	経験年数
基本設計	1 年
DB 設計	1 年

フロントエンド

言語・ツール	経験年数
JavaScript	3.5 年
TypeScript	2 年
React（Next.js）	2 年
Vue（Nuxt.js）	1.3 年
GraphQL	2 年
Jest, React Testing Library, Playwright	2 年

バックエンド

言語・ツール	経験年数
TypeScript	2 年
NestJS	2 年
PostgreSQL	2 年
Jest	2 年
GraphQL	2 年
GitHub Actions	2 年

インフラ

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

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）新規タレントマネジメントプロダクトの開発メンバー、プロジェクトマネジメント、プロダクトマネジメント

担当業務（経験した業務）

プロダクトエンジニアとして、プロダクトマネジメント、プロジェクトマネジメント、チームマネジメント、開発、法人営業の経験あり。

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】

有償利用顧客がゼロの状態からのスタートで、ある企業の担当者が「毎月の組織図の更新作業が Excel では煩雑で手間がかかる」と課題を抱えていることが判明した。

【行動】

この状況に対して、まず目の前の顧客の課題を解決して自プロダクト初の売上を獲得するために、プロダクトマネージャー兼プロジェクトマネージャーとして企画・要件定義・開発マネジメントを一貫して担当。

- まずは n=1 の顧客にフォーカスし、ニーズを深掘り。売上獲得と顧客満足の最大化を両立するため、業務フローに深く入り込み、実際の課題を反映した組織図編集機能の実現を目指した。
- ヒアリングを複数回実施し、課題を温度感の高い順に整理。実現可能性とインパクトのバランスを踏まえて提案機能を決定。
- MVP（Minimum Viable Product）思考で開発スコープを明確化。must と nice-to-have を切り分け、顧客と合意を取りながら初期スコープを圧縮。早期リリースとフィードバックループの高速化を図った。
- 要件定義段階からエンジニア・デザイナーと連携し、技術面・UX 面での妥当性を検証。編集者間のコンフリクト管理においては、事業部で前例のなかったイベントソーシング的アプローチが必要だったため、テックリードと共に早期に設計方針を決定し、実装リスクを最小化。
- 開発体制は、バックエンド 2 名・フロントエンド 1 名・技術サポート 1 名、デザイナー 1 人の構成で、2 ヶ月半というタイトなスケジュールの中で QCD をすべて満たすプロジェクト運営を実施。
- 新たに参画したバックエンドエンジニア 2 人との認識の統一のため、自らドメインモデルの初期叩き台を作成し、開発チーム全体でのドメインモデリングを主導。
- コスト削減の観点では、開発前段階でのスコープ調整、最小限のリソースによる開発推進、加えて QA チームの打鍵作業を自ら巻き取ることで QA リソースのソース削減を行なった。
- クオリティ向上の観点では、フロントエンド・バックエンド含めたコードレビュー、QA チームの打鍵作業の巻き取りを行なった。
- スケジュール遵守に向けては、初期スコープ調整・顧客との進行管理・チームメンバーの進捗確認に加えて、遅延発生時には自らが一部開発タスクを巻き取ることでリカバリー対応を実施。

【結果】

スケジュール優先から途中でクオリティ優先への方針転換を行いながらも、難易度の高い技術要件や新規メンバーとの協業といった不確定要素の多い状況下でもプロジェクトを安定運用し、期日内に高品質なリリースを実現。
顧客からは「これまで話してきた内容が正確に反映されていて良い」とのフィードバックを獲得。
その後、顧客の予算都合により当該案件での売上にはつながらなかったが、本機能を他の有力顧客へ自ら技術営業として提案中。

発揮したバリュー２

【状況】

タレントマネジメント領域において、上司と部下の性格相性をもとに最適な配属案を自動で提案する、競合にはない革新的な機能の企画がプロダクトマネージャーより立案された。

#####　【行動】 プロジェクトマネージャーとして、要件定義・開発マネジメントに加え、営業活動まで一貫して担当。

- 社内人事部門と連携し、性格検査結果に基づいて上司・部下の相性を数値化する独自のマッチングアルゴリズムを開発。新規性が高かったため、発明として特許出願に至った。
- MVP（Minimum Viable Product）思考で開発スコープを明確化。must と nice-to-have を切り分け、早期リリースとフィードバックループの高速化を図った。
- バックエンド１名、フロントエンド１名、デザイナー１名という少人数構成の中、2 ヶ月での開発完了を目標に QCD 管理を徹底。スケジュール・品質・リスクマネジメントを担い、安定した進行を実現。
- 機能の戦略的価値を社外へ広めるべく、営業リスト（従業員 3,000 名以上）に対して 170 件のテレアポを自ら実施し、3 件の商談獲得に成功。また、自身が獲得した商談は自らプレゼン・提案を担当。他営業担当が獲得した案件にも開発責任者として同席し、技術的・企画的な観点から価値提案を行った。

【結果】

現在も複数企業への提案・交渉を継続中

発揮したバリュー３

【課題・状況】

既存機能ではサーベイ（アンケート）をメールで一斉配信していたが、メールに気づかず未回答となる従業員が一定数存在していた。複数の顧客から「チャットツールでも通知したい」との要望があり、回答率向上と管理工数削減が期待されていた。

【行動】

本件において企画・要件定義・画面設計・基本設計・開発・リリースまでを一貫してリード。

- 【企画立案】 PdM と連携しつつ、「本当にやる価値があるのか」をゼロベースで検討。以下の観点から、プロダクトとしての価値が高く、かつビジネスインパクトが大きいと判断し、自ら提案・企画。
    - 回答率向上はプロダクトの価値向上に直結する
    - 実際の営業商談動画から複数の顧客要望が確認できた
    - 海外文献や実データを調査し、チャット連携により回答率が上昇した事例が多数あることを確認

- 【要件定義】 顧客価値の早期提供を重視し、MVP 設計を意識してスコープを明確化。
    - 顧客層が IT 企業中心である点から、初期実装は Slack に絞り、他ツール連携は将来的な拡張とした
    - 競合の類似機能については実装価値を再評価し、PdM と交渉の上、初期リリースでは非対応とし開発負荷を最適化
- 【画面設計】 自らワイヤーフレームを作成し、デザイナーと連携。PdM・デザイナーとディスカッションを重ね、プロダクト思想に沿った画面にブラッシュアップ。
- 【基本設計・開発】 他チームのプロダクトとの連携を要する複雑な構成だったが、BE 領域を 1 人称で担当。具体的な技術的取り組みは以下。
    - Slack App の作成および認証処理
    - Cloud KMS によるアクセストークンの暗号化
    - Slack API を用いた通知処理の設計・実装
        - 特に予約通知のスケーラビリティに課題があり、Slack API の Rate Limit 回避のために Cloud Tasks を用いた時間差通知処理を設計。これにより数千人規模のユーザーへの安定通知を実現
    - DB 設計および API 開発
    - 関連する Terraform 定義の整備・更新
    - FE 実装のコードレビュー対応

【リリース・運用】 他チーム・ビジネス部門と連携してスケジュールを調整し、リリースまでを完遂。リリース後の不具合調査・改修までを一貫して対応。

【結果】

社内での結果

- 回答率の上昇（90%→92%）
- マネージャー層が未回答者に人力でリマインドしていたものが自動でリマインドされるようになり工数削減

発揮したバリュー 4

【課題・状況】

事業部内で他開発チーム、他職種とのコミュニケーションが少なかった。

【行動】

定期的に他開発チームや他職種を交えたランチや MTG を開催。具体的には以下。

- 2 週間に 1 回、異なる開発チームからメンバーを集め、今やっている業務や各チームの雰囲気について話題を振って各人の関係づくりやナレッジの共有ができるような場を醸成
- 1 ヶ月に 1 回、営業と開発を交えたランチを開催。各職種のメンバーに対してチームの状況や課題感について話題を振って各人の関係づくりや職種を超えた情報交換ができるような場を醸成
- 自分が開発した機能について、チーム内だけではなく営業も含めてレビュー会（機能お披露目会）を実施して、実際に機能を触ってもらったフィードバック（FB）を回収。いただいた FB については優先度や対応の可否を PdM と相談し、対応した FB については対応した旨、対応しなかったいくつかの FB については対応しなかった理由を営業に共有することで、開発と営業の信頼関係構築に貢献。

発揮したバリュー５

【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、Flaky なテストになってしまったからあまり有効的には活かせなかった。ただ、結合テストで行なった方が CI の実行時間が短く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６

マイクロサービス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７

【課題・状況】

① BE のリポジトリにおいて、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８

【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、BE、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９

【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。

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。