私の家族は香港にいるので、時々香港にいることもあります。

聞こえますか？、もう一度話していただけますか？もう少しゆっくり話していただけますか？

私は日本で生まれ、海外で育ち(そだち)ました。

咳(せき)と風邪（かぜ）をひいています  
日本に戻ってくるのにそれほど時間はかかりませんでした。

今フリー開（あいて）いている状態です。今は案件とかに入ってないです。

# 最寄りの駅はどこですか。

最寄り（もより）の駅は赤嶺駅（あかみねえき）です。ここからその駅まで徒歩（とほ）で１８分かかります。

# Would you speak more slowly?

私は日本人ですが、ほとんどの時間を海外で過ごしているので、日本語のスキルにはあまり自信がございません。恐れ入りますが、もう少しゆっくりお話しいただけますでしょうか。

# 自己紹介

私は1994年６月14日、沖縄県那覇市宇栄原で生まれました。3歳のときに父の仕事の関係で家族で香港に引っ越し（ひっこし）、それ以降は香港で暮らしてきました。

学校は香港の南屯門官立中學を卒業し、2012年に香港の嶺南大学データサイエンス学部に入学し、コンピュータ工学を専攻いたしました。

# **沖縄県那覇市宇栄原（おきなわけん なはし うえばる）**

1994年 6月 14日

2006年４月南屯門官立中学校入学。

2012年３月卒業、嶺南大学入学

2016年５月から香港の屯門中銀理財中心で中規模銀行向けのWebページの開発に携わりました。

2019年5月からDrape Fit会社で国外市場向けe-commerceサイトの制作に携わりました。

2022年から1月から香港のFora Soft会社で遠隔医療プラットフォームのWebシステム開発に携わる

2023年６月からIT管理スペシャリスト会社でバリューアプリケーションズ株式会社保育および介護業界向けのアプリ開発に携わりました。

**自己紹介をお願いします。**

はじめまして、**南出 創**と申します。どうぞよろしくお願いいたします。

# 私は**2012年**に香港の嶺南大学データサイエンス学部に入学し、コンピュータ工学を専攻いたしました。

# **2016年**に同大学を卒業し、屯門中銀理財中心に入社しました。主な案件は中規模銀行向けサビスシステムの開発でした。

担当フェーズは「要件定義、ニュザインタフェイース（UI）デザイン」で、その他に「フロントエンド開発、API統合」も行ってきました。開発環境として「windows、MYSQL、vscode」を使用し、チームのメンバーは10人でした。プロジェクトにおける私の役割は「システムエンジニアとプログラマー」でした。特に、UIの設計を洗練させたり、ジャバスクリプト（JavaScript）、やリアクト（React.js）を使ってコンポーネントベースのインターフェースを作成したり、**React Hooks**や**Context API**を使用して状態管理を行いました。その結果、銀行業務の処理速度が向上し、サービス全体の効率化と顧客満足度の向上に貢献することができました。ここでは、ネットワークエンジニアとしてのスキルだけでなく、大切な研究習慣も身につけることができました。(自分の才能をもっと活かしたいと思いました。)

# **2020年**に医療サービスを提供するFora Softに入社し、様々プロジェクトに参加いたしました。

特に、遠隔医療プラットフォームのWebシステム開発に関わりました。担当フェーズは「要件分析、検索アルゴリズムの作成、フロントエンド開発」で、その他に「テストおよび最適化」も行ってきました。開発環境として「windows、PostgreSQL、VScode」を使用し、チームのメンバーは約50人ぐらいでした。プロジェクトにおける私の役割は「プログラムエンジニア」でした。特に、薬剤データベースに適（てき）したデータ構造を設計し, フルテキスト検索からAI検索まで、さまざまな検索方法を提案いたしました。それにより、検索システムの最適化に貢献することができました。このプロジェクトには多くの開発者が関わっており、チームスピリットと協調性の重要性を深く「ふかく」実感いたしました。(2020年契約社員として、入社しました。契約は2022年に終了しました。)

# **2023年**からIT管理スペシャリストというオンライン会社でシステムエンジニアとサブリーダとして働いておりました。

主な業務は国外市場向けのe-commerceサイトの制作でした。仕入管理システムのフロントエンド、バクエンドの開発を中心に、要件定義から運用まで一貫して関わりました。**フロントエンドでは、**TypeScript、React、Tailwind CSS を用いて、PC およびスマートフォンの両方で快適に動作するレスポンシブなユーザーインターフェースを実装しておりました。**バックエンド**ではAPI駆動型アーキテクチャを実装するためのドキュメントを作成し、MongoDBを用いてデータベースを構築いたしました。その結果、システムの安定性や拡張性を向上させることができました。このプロジェクトを通じて、コンピューターエンジニアとして貴重な経験を積むことができました。

# 以上のように, これまで約**9年間**、ソフトウェアエンジニアとして、Webシステムの開発に携わってまいりました。

特に、JavaScriptやReact.js、そしてPHPやLaravelを使ったフロントエンド・バックエンド両方の開発を得意としております。そうした経験を活かして、国内外の市場に向けて実用的で最適なシステム開発に貢献できると自負しております。現在は、自分のスキルや経験を活かせるお仕事を探しております。

# 統合

私は**2012年**に香港の嶺南大学データサイエンス学部に入学し、コンピュータ工学を専攻いたしました。

**2016年**に/同大学を卒業し、屯門中銀理財中心に入社しました。主な案件は/中規模銀行向けサービスシステムの開発でした。担当フェーズは「要件定義、ニュザインタフェイース（UI）デザイン」で、その他に「フロントエンド開発、API統合」も行ってきました。開発環境として「windows、MYSQL、vscode」を使用し、チームのメンバーは10人でした。プロジェクトにおける私の役割は/「コンピューターンジニア」でした。特に、UI設計の洗練化に注力し、JavaScriptやReact.jsを用いて/コンポーネントベースのインターフェースを構築いたしました。また、 React HooksやContext API Reduxtoolkitを活用して、効率的な状態管理を実施しておりました。

その結果、銀行業務の処理速度が向上し、サービス全体の効率化と顧客満足度の向上に/貢献することができました。ここでは、ネットワークエンジニアとしてのスキルだけでなく、大切な研究習慣も身につけることができました。

**2020年**にアメリカで医療サービスを提供する/Fora Softに入社し、様々プロジェクトに参加いたしました。特に、遠隔医療プラットフォームのWebシステム開発に関わりました。ここでは、薬剤検索向けのAI検索アルゴリズムの作成、ユーザーのニーズを重視した/フロントエンド開発を中心に、バックエンドの統合および/最適化にも/関わってまいりました。バックエンドにカスタムAI検索エンジンを組み込む/必要があったので、Python、Djangoを使って/開発しました。検索システムを完成させるために、NLPのQAモデルである deepset/roberta-base-squad2、さらに、大LMM（規模マルチモーダルモデル）である mistral-instruct や OpenAIのChatGPTを使用してみました。

これらのモデルの中から適切なものを選定し、ファインチューニングや適切なプロンプト設計を行うことで、薬剤検索システムの最適化を実現することができました。このプロジェクトには多くの開発者が関わっており、チームスピリットと協調性の重要性を深く/実感(ng)いたしました。

**2023年**からIT管理スペシャリストというオンライン会社で/システムエンジニアと/サブリーダとして働いておりました。主な業務は国外市場向けのe-commerceサイトの制作でした。仕入管理システムのフロントエンド、バクエンドの開発を中心に、要件定義から運用まで一貫して/携わってまいりました。**フロントエンドでは、**TypeScript、React、Tailwind CSS を用いて、PC およびスマホの両方でスムズな動作する/レスポンシブなユーザーインターフェースを実装しておりました。**バックエンド**ではNode.js, Express,（php, laravel） MongoDBを用いてAPI駆動型アーキテクチャを構築しました。具体的には、RESTful APIの設計と実装を担当し、認証・認可（JWTによるトークンベース認証）や、エラーハンドリング（handling）機能を実装しました。その結果、システムの安定性や拡張性を向上させることができました。このプロジェクトを通じて、システムエンジニアとして貴重な経験を積むことができました。

以上のように, これまで/約**9年間**、システムエンジニアとして、Webシステムの開発に携わってまいりました。特に、TypescriptやReact.js、そしてPHPやLaravelを使った/フロントエンド・バックエンド両方の開発を得意としております。そうした経験を活かして、国内外の市場に向けて実用的で最適な/システム開発に/貢献できると自負しております。御社やクライアント先のプロジェクトでも、私の経験とスキルが/きっと役立つ（やくだつ）と思います。よろしくお願いいたします。

# 自己紹介をしてください

自己紹介のために、私の職務経歴書を共有させていただきたいと思います。

私は**2012年**に香港の嶺南大学データサイエンス学部に入学し、コンピュータ工学を専攻いたしました。

**2016年５月**に同大学を卒業し、屯門中銀理財中心に入社いたしました。主な案件は/中規模銀行向けサービスシステムの開発でした。担当フェーズは「要件定義、ニュザインタフェイース（UI）デザイン」で、その他に「フロントエンド開発、API統合」も行ってきました。プロジェクトにおける私の役割は/「コンピューターンジニア」でした。特に、ユーザーのニーズを重視した要件定義、画面設計の洗練化に注力し、JavaScriptやReact.jsを用いて/コンポーネントベースのインターフェースを構築いたしました。特に、React HooksやReduxtoolkitを活用して、効率的な状態管理を実施しておりました。また、Laravel使用したインターフェースとバックエンドの統合を完成しました。React Hooks：リアクト フックス、Reduxtoolkit：リダックス ツールキット

Laravel：ララベル

**2019年５月**からアメリカにあるDrape Fit （ドレープフィット）に入社し、フリランスとして働いておりました。主な業務は国内市場向けのe-commerceサイトの制作でした。**フロントエンドでは、**TypeScript、React、Tailwind CSS を用いて、PC およびスマホの両方でスムズな動作するレスポンシブなユーザーインターフェースを実装しておりました。**バックエンド**ではNode.js, Express,（php, laravel） MongoDBを用いてAPI駆動型アーキテクチャを構築しました。具体的には、RESTful APIの設計と実装を担当し、認証・認可（JWTによるトークンベース認証）や、エラーハンドリング（handling）機能を実装しました。また、GitHub CI/CDを用したDigitalOcean環境でのシステム構築・デプロイを実装しました。その結果、システムの安定性や拡張性を向上させることができました。GitHub：ギットハブ、DigitalOcean: デジタルオーシャン

**2022年1月**から、香港にあるFora Soft に入社し、遠隔医療プラットフォームのWebシステム開発に関わりました。特に、薬剤検索向けの全文検索アルゴリズムやAI検索アルゴリズムの構築を中心に、フロントエンドおよびバックエンドの開発に携わってまいりました。フロントエンドでは、Material（マテリアル） UIを用いてユーザフレンドリなUIを実現いたしました。バックエンドにカスタムAI検索エンジンを組み込む/必要があったので、Python、Djangoを使って/開発しました。検索システムを完成させるために、まず、PostgreSQLを用いて全文検索アルゴリズムを構築いたしました。その後、曖昧な検索語にも対応するため、AI検索アルゴリズムの構築に取り組みました。そのために、NLPのQAモデル（RealTime QA）やLLMのOpenAIのChatGPTモデルを使ってみました。これらのモデルの中から適切なものを選定し、ファインチューニングや特別なプロンプト設計を行うことで、薬剤検索システムの最適化を実現することができました。

**202３年６月**から、バリューアプリケーションズ株式会社でサブリーダとして、保育および介護業界向けのアプリケーション開発に従事（じゅうじ）しました。このプロジェクトでは、**Ionic + React**を用いたハイブリッドアプリのフロントエンド開発を担当し、iOS・Android両方に対応するマルチプラットフォームアプリを開発しました。iOS・Androidアイオーエス・アンドロイド

バックエンドは**Node.js + Express（エクスプレス）**で構築し、REST APIの設計・実装を担当しました。特に、非同期処理を活用してリクエストの応答速度を約30%改善し、パフォーマンス最適化に貢献しました。**インフラ面ではAWSを活用**し、EC2によるアプリケーションサーバの構築・運用、S3による画像や動画などのユーザー生成コンテンツの保存と配信（はいしん）を行いました。S3との連携では、セキュリティとコスト効率を両立させる構成を提案し、ユーザー数の増加（ぞうか）にも耐（た）えられるスケーラブルな仕組みを実現しました。

また、開発の品質とスピードを両立させるため、CI/CD環境の導入および運用を行いました。

このプロジェクトを通じて、**フロントエンドからバックエンド、インフラまで一貫して担当**し、システムエンジニアとしての貴重な経験を積むことができました。

以上のように, これまで/約**9年間**にわたり、システムエンジニアとしてWebシステムの開発だけでなく、AI関連サービスのシステム開発にも関わってまいりました。そうした経験を活かして、国内外の市場に向けて実用的で最適な/システム開発に/貢献できると自負しております。御社やクライアント先のプロジェクトでも、私の経験とスキルが/きっと役立つ（やくだつ）と思います。よろしくお願いいたします。

## English

Thank you very much for giving me the opportunity to introduce myself. To begin, I would like to share my professional background.

I entered the Faculty of Data Science at Lingnan University in Hong Kong in 2012, where I majored in Computer Engineering. After graduating in May 2016, I joined the Tuen Mun branch of Bank of China Wealth Management Center.

There, I worked primarily on service system development for mid-sized banks. My responsibilities included requirements analysis and UI design, and I was also involved in frontend development and API integration. My role in the project was as a computer engineer, with a strong focus on user-centered requirements definition and refined screen design. I developed component-based interfaces using JavaScript and React.js, incorporating React Hooks and Redux Toolkit to manage state efficiently. In addition, I implemented backend integration using Laravel.

In May 2019, I began working as a freelance developer for **Drape Fit**, a company based in the United States. My main task was the development of an e-commerce site targeting the domestic market. On the frontend, I built responsive interfaces optimized for both desktop and mobile using TypeScript, React, and Tailwind CSS. For the backend, I used Node.js, Express, PHP (Laravel), and MongoDB to construct an API-driven architecture. I was responsible for designing and implementing RESTful APIs, handling authentication and authorization using JWT, and implementing robust error handling.

In January 2022, I joined **Fora Soft**, a company in Hong Kong, where I contributed to the development of a web system for a telemedicine platform. My main focus was building a search engine for pharmaceutical information using both full-text and AI-powered search algorithms. On the frontend, I utilized Material UI to design a user-friendly interface. On the backend, I developed a custom AI search engine using Python and Django. To implement the search feature, I first built a full-text search algorithm using PostgreSQL. Then, to support fuzzy queries, I explored and integrated AI technologies such as NLP-based QA models (like RealTime QA) and OpenAI’s ChatGPT model. I evaluated and fine-tuned these models and designed prompts to optimize the pharmaceutical search experience.

Since June 2023, I have been working at **Value Applications, Inc.** as a sub-leader, developing applications for the childcare and nursing care industries. I led the frontend development of a hybrid mobile application using **Ionic + React**, ensuring compatibility across both iOS and Android platforms. The backend was built with Node.js and Express, where I designed and implemented RESTful APIs. Notably, I improved response times by approximately 30% through asynchronous processing, contributing to overall performance optimization.

On the infrastructure side, I leveraged AWS, handling EC2 instance setup and operation for the application servers, and using S3 to store and deliver user-generated content such as images and videos. I proposed an architecture that balanced security and cost-effectiveness, and built a scalable system capable of handling increasing user loads. I also introduced and maintained a CI/CD environment to enhance both development speed and product quality.

Through these experiences, I’ve gained extensive knowledge in frontend, backend, and infrastructure, and I’ve grown as a full-stack system engineer. Over the past 9 years, I’ve been involved not only in web system development but also in building AI-powered services.

With this experience, I believe I can contribute to the development of practical and high-quality systems for both domestic and international markets. I am confident that my skills and experience will be valuable to your company and your clients’ projects.

Thank you very much for your time and consideration.

# 自己PRをしてください

私はWebサービス開発において、特にJavaScriptフレームワークなどの最新技術に強みを持っております。2016年からこれまで、中規模から大規模なプロジェクトに携わってきており、豊富な実務経験を積んでまいりました。

また、2020年以降は、OpenAIのような学習済みAIモデルを活用したプロジェクトにも関わっており、日々進化する技術に対して柔軟に対応する力を磨いています。

業務では、納期をしっかり守ることはもちろん、チーム内の円滑なコミュニケーションを意識しており、信頼されるメンバーとして行動することを大切にしています。

常に新しい技術やトレンドにもアンテナを張り、自分のスキルを高め続けることで、会社に貢献できる存在でありたいと考えています。

# 学生時代に学習した内容を教えてください

大学ではc++, JavaScript や PHP などのプログラミング言語を用いたウェブアプリケーションの開発に携わり、データ構築の設計やデジタル信号(しんご)処理に関する研究を行っておりました。  
また、AI技術についても学び（まなび）、研究していました。特にディープラーニングを用いた画像（がぞう）処理について研究していました。  
現在はウェブシステムエンジニアとして、クライアント向けおよび専門分野向けのさまざまなネットサービスシステムの開発に従事（じゅうじ）しております。

私の論文（ろんぶん）のタイトルはYOLOモデルを用いたリアルタイム物体検出（ぶったいけんしゅつ）です。

# **なぜ当社を志望されましたか？(**現在の会社に入社を決めた理由は何ですか？)

まず第一に、私のスキルセットが御社の事業（じぎょう）内容に非常によくマッチしていると感じたためです。特に、これまでに「医療サービスシステム」の開発経験が豊富ですので、御社の「医療パッケージシステム」の開発に大変興味を持っております。  
また、御社の求人ポスタを拝見し、幅広い分野のシステム開発に携われる環境や、技術力だけでなくビジネススキルも磨ける点に大きな魅力を感じました。  
さらに、御社が掲げる『技術と信頼で、未来の可能性をリノベート（renovate）する』というビジョンにも深く共感し、志望いたしました。

私はこれまで「JavaScriptやPHPを中心に」フルスタック開発を行ってきました。また、御社は最新の技術を積極的に取り入れ、エンジニアの成長環境を支援しています。そうした環境で、自分のスキルを磨き、新しい技術にも意欲的に挑戦（ちょうせん）したいと考えています。その結果として、企業の成長に貢献できると確信しています。

# バリューアプリケーションズ株式会社(今の会社での仕事内容を教えてください)

2023年6月から、バリューアプリケーションズ株式会社で保育（ほいく）および介護（かいご）業界向けのアプリケーション開発に従事（じゅうじ）しました。

Kindyは、保育園（えん）と保護者（ほごしゃ）のコミュニケーションを円滑（えんかつ）にするためのWebアプリケーションです。登園連絡、デジタル連絡帳、クラス写真の共有、健康記録の管理などの機能を提供し、保育士と家庭（かてい）の双方（そうほう）にとって育児体験（いくじ）を向上させます。

* 保護者はスマホを使って、子どもの登園状況を簡単に保育園に通知できます。忙しい朝でも迅速な連絡が可能です。
* デジタル連絡帳では、子どもの一日の様子（ようす）（機嫌（きげん）、食事（しょくじ）、体温（たいおん）、排便（はいべん）、午睡（ごすい）時間など）をリアルタイムで共有でき、保護者とスタッフの間（あいだ）で透明（とうめい）なコミュニケーションが促進（そくしん）されます。
* スタッフは、遊び（あそび）や散歩など日々（ひび）のクラス活動の写真をアップロードでき、保護者は子どもの日常（にちじょう）を常に感じることができます。
* もし、すべての保護者に通知が必要な状況が発生した際には、保育園はアプリを通じて重要なお知らせや緊急（きんきゅう）連絡を保護者へ直接送信できるため、迅速かつ効果（こうか）的なコミュニケーションが可能です。
* このアプリでは健康記録の管理機能もございます。保育園で実施された健康診断（しんだん）の結果がデジタル化され、保護者は子どもの健康データを便利に確認できます。

このアプリは東京児童（じどう）協議会（きょうぎかい）内の16の保育園に導入されております。

このプロジェクトでは、**Ionic + Reactを用いたハイブリッドアプリ**のフロントエンド開発を主に担当し、iOS・Android両方に対応するマルチプラットフォームアプリを短期間で開発しました。UIは保育士（ほいくし）や保護者の両方が直感的に操作できるよう、アクセシビリティとユーザビリティを重視しました。

バックエンドは**Node.js + Express**で構築し、REST APIの設計・実装を担当しました。特に、非同期処理を活用してリクエストの応答（おうとう）速度を約30%改善し、パフォーマンス最適化に大きく貢献しました。また、**認証・認可機能**も自ら（みずから）設計・実装し、セキュアなシステム構築に努め（つとめ）ました。

【Ionicによるアプリ開発】

🔹ステップ①：環境構築について

Node.jsとnpmをインストールし、その後、Ionic CLIをグローバルにインストールしました。また、AndroidアプリのビルドにはAndroid Studio、iOSアプリにはXcodeを使用しました。

🔹ステップ②：プロジェクトの作成

プロジェクトの作成には、Ionic CLIの ionic start コマンドを使用し、blank（ブランク） テンプレートを選択しました。（例：ionic start myApp blank --type=angular）

🔹ステップ③：開発（ページとUIの実装）

開発では、Ionicが提供するUIコンポーネント（IonButton、IonCardなど）を活用し、ユーザーフレンドリーな画面を構築しました。  
ページの作成には ionic generate page を使用しました。(generate: ジェネレート)  
たとえば、ionic generate page home で「ホーム」画面を追加し、その中にボタンやリストを配置しました。

**🔹ステップ④：動作確認・ローカルテスト**

作成したアプリは ionic serve を使ってローカルサーバー上で確認しました。  
また、Capacitor（**カパシター**）を使用してAndroid Studioと連携し、実機やエミュレーター(emulator)上でも動作検証(けんしょう)を行いました。

**🔹ステップ⑤：ネイティブ機能の実装**

カメラ機能などのネイティブ機能は、Capacitorプラグイン（plugin）を使用して実装しました。  
たとえば、@capacitor/cameraを使って、写真を撮影（さつえい）し、アプリ内で表示させる機能を実装しました。  
コマンドは npm install @capacitor/camera を使い、npx cap sync で同期しました。

**🔹ステップ⑥：本番ビルド・公開準備**

最後に、ionic build で本番用のビルドを行い、npx cap copy でネイティブプロジェクトにコピーしました。  
その後、Android Studioを使ってAPKファイルをビルドし、動作確認やストアへの提出（ていしゅつ）に備え（そなえ）ました。

はい、フロントエンドとバックエンドの連携実装において、バックエンドに Next.js を使用する場合は API Routes を作成しました。次に、フロントエンドから fetch() （フェッチ）や axios（アクシィオス） を用いて、先ほど作成した API にリクエストを送信します。このとき、非同期処理を使ってデータを取得し、React の状態管理（useState や useEffect）を通じて、UI にデータを表示させます。また、追加処理について、認証（にんしょう）、エラー処理、読み込みインジケーター（indicator）を行います。

ログイン認証JWTトークン認証とOAuth2認証方法を活用いたしました。

tokenやsession情報をrequest header（ヘッダー）に付与(fuyo)して送信し、バックエンド側でその認証情報を検証(kenshō)します。

エラー処理: backendの応答（response）をチェックし、errorが発生(hassei)した場合は、ユーザーに適切なエラーメッセージを表示できるようにしました。

**認可機能つきましてはRBACを**用いた実施いたしました。

**インフラ面ではAWSを活用**し、EC2によるアプリケーションサーバの構築・運用、S3による画像（がぞう）や動画（どうが）などのユーザー生成コンテンツの保存と配信を行いました。S3との連携では、セキュリティとコスト効率を両立させる構成を提案し、ユーザー数の増加（ぞうか）にも耐（た）えられるスケーラブルな仕組みを実現しました。

また、GitHubによるバージョン管理とコードレビューの仕組みをチームに導入し、CIの自動化にも取り組むことで、開発フローの品質（ひんしつ）と効率を向上させました。

このプロジェクトを通じて、**フロントエンドからバックエンド、インフラまで一貫して担当**し、ユーザー目線に立った開発と、スケーラブルかつ安全なシステム構築の重要性を学びました。特に、保育・介護という現場で使われるプロダクトを支える開発には、高い信頼性と直感的なUIが求められることを実感しました。

御社のプロジェクトでも、これまで培ってきたフルスタックの技術力と、現場の課題解決に貢献してきた経験を活かし、価値あるプロダクト開発に貢献したいと考えております。

どうぞよろしくお願いいたします。

# 仕事上で一番苦労したこと、どうやって乗り越えたか

# やりがいを感じる瞬間は何ですか？

# 希望職種（しょくしゅ）はなんですか？

はい、私の希望職種はWebアプリケーションエンジニアです。  
特に、JavaScriptや PHPを用いたフロントエンド・バックエンドの開発を得意としております。  
最近では、PythonやDjangoを用いたAI関連サービスの開発にも携わってまいりました。こうした経験を活かし、幅広い技術に柔軟に対応できる職務を希望しております

# 転職活動を開始した背景 [はいけい]は何ですか？次の転職先では、どのような環境を求めていますか？

転職活動を開始した背景：主な理由は、より安定した環境で成長したいと考えたからです。2023年6月から、バリューアプリケーションズ株式会社でサブリーダーとして、保育および介護業界向けのアプリケーション開発に従事してまいりました。3名体制（organization）のチームでの開発を通じて、リーダーシップを発揮するとともに、業界に特化した知識を深める（ふかめる）ことができました。

今後は、より大規模な企業において、これまで培ってきたスキルを活かしながら、多くのメンバーと協力して、大規模な開発経験を積み、先端技術への理解をさらに深めていきたいと考え、転職を志望しております。  
現在はフリーランスとして活動していますが、長期的なキャリアを考えると、自国の企業での経験を積むことが重要だと思いまた。

# **前職を退職[たいしょく]された理由は何ですか？**

大学卒業後、屯門中銀理財中心で、銀行業務に携わり、さまざまな経験を積むことができました。2019年5月に、アメリカで活躍したいという思いから、オンラインでDrape Fit会社に入社しました。ここでは国内市場向けのe-commerceサイトの制作について、多くの経験を積むことができました。

2022年1月より、友人の紹介により、香港にあるFora Softに契約社員として入社いたしました。2023年5月に契約期間が満了いたしました。

2023年6月から、バリューアプリケーションズ株式会社でサブリーダーとして、保育および介護業界向けのアプリケーション開発に従事してまいりました。3名体制のチームでの開発を通じて、リーダーシップを発揮するとともに、業界に特化した知識を深める（ふかめる）ことができました。

今後は、より大規模な企業において、これまで培ってきたスキルを活かしながら、多くのメンバーと協力して、大規模な開発経験を積み、先端技術への理解をさらに深めていきたいと考え、転職を志望しております。

# **入社後にどのような仕事に取り組みたいですか？**

入社後は、これまでの経験を活かして、御社のWeb開発やシステム構築に貢献できればと思っています。JavaScriptやPHPを活用した開発とPythonを活用した開発には自信があり、これらの言語をベースにフレームワーク開発を担当したいと考えています。また、御社は最新の技術を積極的に取り入れ、エンジニアの成長を支援されているため、そうした環境の中で自分のスキルを磨き、新しい技術にも意欲的に挑戦していきたいと考えております。

# **あなたの強み（つよみ）・弱み（よわみ）を教えてください。**

**強み：**私は、JavaScript、PHP、Pythonなど、幅広い技術スタックを扱って（あつかって）まいります。そこで、フロントエンドからバックエンドまで柔軟に対応可能なフルスタックエンジニアとしてのスキルを有しております。また、サブリーダーとして、システム設計から実装、テスト、リリースまで、一貫した開発経験も積んでまいりました。

弱み[よわみ]：私は子どもの頃[ころ]から香港で暮(く)らしておりましたので、**日本語**はまだあまり得意ではございませんが、読み書き、話すのは大丈夫です。その代わりに英語と中国語は得意です。私の父は日本人ですが、いつも忙（いそが）しくて、私に日本語を教える（おしえる）時間がありませんでした。また、私の母は中国人なので、家庭（かてい）では中国語で話していました。時々、日本語の教材（きょうざい）を渡してくれましたので、それを使って私は独学で日本語を勉強してきました。

**日本での業務経験**が少ないですが、すぐに慣れると思っています。「Spring Boot」の実務経験は浅いですが、類似する技術の経験が豊富ですので、スムーズに習得できると考えております。

# **どのようなキャリアプラン（career plans）を描[か]いていますか？**

フルスタックエンジニアとして、フロントエンドからバックエンド、インフラまで幅広く（はばひろく）対応できるエンジニアを目指(めざ)しています。将来的には、チームリーダーとして大規模プロジェクトをリードできる力（ちから）も身につけたいです。

短期（1～2年）：

まずは貴社の開発環境やチームに適応し、担当するプロジェクトにおいて主要な開発者として活躍したいと考えております。また、AWSを活用したシステム設計・開発のスキルをさらに強化したいと考えております。

中期（3～5年）：

システム設計や要件定義に関わる上流[じょうりゅう]工程[こうてい]に進出し、プロジェクトの技術的リーダーとして活躍していきたいと考えております。

チームのマネジメントスキルを習得し、開発リーダーやテックリードを目指していきたいと考えております。

長期（5年～）：

CTO候補（こうほ）として、システム全体の設計や技術選定に関与（かんよ）していきたいと考えております。

# **給与 [きゅうよ＝**年収**]はどのくらいを希望していますか？**

年収につきましては、500万円から600万円を想定しております。

月額料金（montly rate）,給料（きゅうりょう）

**月給[げっきゅう**]50万

**時給**  [じきゅう] 2000円

# **これまでの開発経験について教えてください。**

私はWeb開発を中心に、JavaScriptやReact を活用したフロントエンド開発、PHPやLaravelそしてnode.js、expressを使ったバックエンド開発の経験があります。最近ではpythonやDjangoを用いたAI関連サービスにも関わってまいりました。特に、国外市場向けのEコマースサイトや医療・保育分野のWebサービスシステムの開発に携わってきました。また、サブリーダーとして、システム設計から実装、テスト、リリースまで、一貫した開発経験も積んでまいりました。さらに、システム全体の最適化やセキュリティ対策にも注力してまいりました。

# **最後に何か質問はありますか？**

* フルリモートの場合、会社はどのような働き方（kata）や制度を提供していますか？
* 現在開発されている「ECプラットフォーム」について、技術的な課題があれば、もう少し詳しく教えていただけますでしょうか？
* 案件選択制度では、どのようなプロセスでエンジニアに最適な案件が選択されるのでしょうか？
* 貴社で最も重要視されているエンジニアのスキルセットについて、もう少し詳しく教えていただけますか？

# **当社以外に応募している企業はありますか？**

現在、貴社を第一志望として考えておりますが、他のIT企業にも並行（[へいこう）して応募しております。

ただし、貴社の業務内容は私のスキルと非常に適合しているだけでなく、エンジニアとして成長できる環境にも強く魅力を感じており、貴社でのキャリアを心より志望しております。

# **チーム開発で大切にしていることは何ですか？**

**回答例:**コミュニケーションを大切にしています。チームワークを意識しながら、状況に応じて柔軟に対応し、業務を進めてまいりました。プロジェクトの進捗状況を共有し、困ったときには助けを求めることで、チーム全体での生産性を高めることができると考えています。

# リモートワークへの対応は可能ですか？

回答例：

はい、リモート環境でも効率的に仕事を進められるように、自主[じしゅ]的なタスク管理やこまめなコミュニケーションを大切にしています。海外のチームとリモートで連携しながら開発を行ってきた豊富な経験がございますので、リモートワークにも柔軟に対応可能です。

# チムリダとしてのけいけんがありますか？

**English:**I had experience working as a sub-leader at an online company, where my team consisted of eight members including myself. In order to ensure effective teamwork, we have a standard meeting 2 or 3 times in a week, if we face a trouble problem, we suggest the problem and discuss how to tackle the problem and solve it. In addition, we use the collaborative tools such as Git for checking and ensuring everybody's progress."

**Japanese:**

私はオンライン企業でサブリーダーとして働いた経験があり、チームは私を含めて 8 名で構成されていました。効果（こうか）的なチームワークを確保するため、週に2～3回の定例ミーティングを開催（かいさい）しています。何かトラブル（trouble）があったら、それについて話し合って、どうやって対応していくかを決めています。また、Gitなどのコラボレーション(collaboration)ツールを活用し、全員の進捗状況を確認・把握(はあく)しています。

# 趣味は何ですか？

My hobby is playing ping pong. When I have some free time, I play ping pong with my friend.

趣味(しゅみ)は卓球（たっきゅ）でございます。時間がある際には、友人と卓球を楽(たの)しんでおります. 私が応援している卓球選手は張本(はりもと)選手です。

# パソコンを受けたりません。

はい、その認識（にんしき）で大丈夫です。

## にほんの方と一緒におしごとすることってありますか。

また、日本企業のお客様のお仕事にも関わってまいりました。特に、フリーランスとして活動しており、副業（ふくぎょう）として、WordPress（ワードプレス）などのCMSを利用して日本市場向けのWebサイトを構築した経験があります。

# Excuse.

私の日本語力はそれほど高くありませんが、英語と中国語は得意です。

また、プロジェクトに詳細(しょうさい)なドキュメントがあれば、作業（さぎょう）に問題はありません。

私にテストワークを依頼していただければ、決してがっかりさせません。何卒ご理解のほどお願い申し上げます

## 出社可能性

私の家族は現在香港に住んでおりますので、私も時々香港に滞在することがあるかもしれません。現在は出社が難しい状況ですが、10月以降であれば出社が可能になると考えております。

## あなたには日本人ですか。

私は日本人です。実は私の父は日本人ですが、母は中国人です。また、子供の頃から香港で過ごしているますので、日本語はあまり得意でございません。ただ、英語と中国語は得意です。

## フリランサとして働く理由はなんですか？

現在は日本語がまだ十分ではないため、まずは1年ほどフルリモートのフリーランスとして働く予定です。将来的には、日本語力が向上した段階（だんかい）で、正社員として働くことを目指しています。そのため、スキルを活かしながら日本語の学習にも力（ちから）を入れて取り組んでいます。

# コードを書く上でのこだわり・ポリシーを教えてください。

**可読性（かどくせい）・保守性・再利用性・セキュリティ**などを意識した観点で答える（こたえる）と、技術者としての信頼感が高まります。

# なぜ、プログラムミングトとITに興味を持った、原因というかきっかけは何かござぢますか。

My journey into programming began with something very personal. When I was a student at Lingnan University, majoring in Computer Engineering, I found myself drawn to web applications. This wasn’t just because it was my field of study, but because of a deeper motivation—my mother.

She worked tirelessly at a small general store, spending long hours calculating and managing sales data by hand. I wanted to ease her burden. So, I decided to develop a simple web-based sales tracking system for her. It wasn’t perfect, but through that project, I learned more than I ever had in a classroom. It was the moment I realized that web development has the power to make people's lives better.

Since then, creating useful and human-centered applications has been my passion—and my purpose.

私がプログラミングの道（みち）に進んだきっかけ（opportunity）は、身近な家族の存在でした。  
私は香港の嶺南大学でコンピュータ工学を専攻しており、在学中にWebアプリケーションに強く惹かれるようになりました。ただの専攻科目というだけでなく、心から「誰かの役に立てる」と感じたからです。

母は長年、個人経営の雑貨店で働いており、販売商品の数や種類を手作業で管理していました。毎日遅くまで作業する姿を見て、私は「母を助けたい」と思い、販売管理システムを自作し始めたのが最初のプロジェクトでした。

その経験から多くを学び、「人の役に立つものをつくることができる」というWeb開発の魅力に夢中になりました。それが、私のエンジニア人生の原点です。

# A big turning point or challenge in your career and how you overcame it? 大きな転機や課題、そしてそれをどのように克服しましたか?

After graduating, I started my career at Tuen Mun BOC Wealth Management Center. It was a completely new environment for me, and at first, I struggled to adapt.

But instead of giving up, I doubled down on my efforts. I worked hard, asked questions, and embraced learning from others. Through this experience, I learned that resilience and communication are just as important as technical skills. Teamwork, especially, became something I value deeply. Working with others to solve problems, sharing knowledge, and building something together—that’s what truly excites me now.

大学を卒業してから、屯門中銀理財中心に入社したんですが、最初はまったく(absolutely)新しい環境に戸惑(とまど)って、なかなか（easily）慣れることができませんでした。それでもあきらめ(giving up)ずに努力(どりょく)を重ね（かさね：pile, heap）て、積極的に質問したり、周り(まわり)から学ぶ姿勢（しせい）を大事（だいじ）にしてきました。そのおかげで、少（すこ）しずつ環境に慣れて、自信も持てるようになりました。  
この経験を通して、技術力だけじゃなくて、「粘り強さ：ねばりつよさ」や「チームとのコミュニケーション能力」も、仕事ではすごく大事なんだって気づきました。

# From Colleagues to Best Friends

When COVID-19 spread across the world, everything changed. Many lives were lost, and we faced global uncertainty. But at the same time, one thing began to accelerate—**internationalization**.

As remote work became the norm, I joined a diverse team of eight people. Though we were scattered across different countries, we worked together online every day. I am Japanese. Kevin is from England. 世雄 (Shiyu) is from China. All of us came from different cultures and languages, but that didn’t stop us from connecting deeply.

As we tackled projects, supported one another, and shared our lives, we naturally grew closer. What started as a team gradually became a circle of true friendship. We are now more than colleagues—we are best friends.

That experience taught me something I’ll never forget: **even through screens and distance, human connection has no borders**.

**🇯🇵 同僚から親友へ 〜国境を越えた絆〜**

コロナウイルスが世界中に広がり、多くの命が失われ、私たちの生活も一変しました。しかしその一方で、「国際化」は急速に進みました。

リモートワークが当たり前になった中、私は8人のメンバーとともにオンラインでプロジェクトに取り組むことになりました。国籍はバラバラで、私は日本人。Kevinはイギリス人。そして世雄（Shiyu）は中国人。それぞれ背景や文化が異なりましたが、日々のやり取りを通して、私たちは深く理解し合えるようになりました。

一緒に課題を乗り越え、互いに支え合い、時にはプライベートな話も共有しながら、私たちの関係は自然と深まりました。ただのチームではなく、心から信頼し合える「親友」になったのです。

この経験から私は、**人とのつながりに国境はない**ということを心から実感しました

# **キャッチコピー・好きな言葉（補足あり）**

**Together with your friends, keep your promises and cherish love!**

**仲間と(なかまと)ともに、約束（やくそく）を守り（まもり）、愛（あい）を大切！**  
チームワークを重んじ、一緒に働く仲間への思いやりを忘れず、どんなプロジェクトでも「約束＝信頼」を最優先に行動します。クライアントとの信頼関係を築きながら、人と人との“つながり”を大切にするエンジニアでありたいと思っています。仕事には情熱を、仲間には感謝を、そしてすべてに愛を込めて取り組んでいます。

Weather.

今は雨が(あめが)降っています。(It’s raining.)

今は晴（は）れています。（It’s sunny now）

今は曇り（くもり）です。（It is cloudy now）

。【一般的な質問想定】 1自己紹介をしてください 2自己PRをしてください 3学生時代に学習した内容を教えてください 4現在の会社に入社を決めた理由は何ですか？ 5今の会社での仕事内容を教えてください（実績も）。 6「仕事上で一番苦労したこと、どうやって乗り越えたか」を教えてください 7やりがいを感じる瞬間は何ですか？ 8転職理由は何ですか？ 9なぜ当社だとその転職理由は叶うと思いますか？ 10志望動機を教えてください 11これまでの経験やスキルを当社でどう活かすことができると思いますか？ 12今後身に付けたいスキルは何かありますか？ 13入社したらどんな仕事がしたいですか？ 14長所・短所を教えてください 15今後のキャリア目標は？5年～10年後どうなりたいか？ 16採用されたらいつ入社できますか？ 17何か質問ありますか？

//

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 履　歴　書 | | | | 2025年 5月 13日現在 |
| 氏　名: | 南出創 | | | |
| ふりがな 　　みなみで　はじめ | | | | |
|  | 1994年 6月14日生　（満3１歳） | | | | 性別 男 |
| 出生地（〒） 沖縄県那覇市宇栄原1丁目 | | | | | |
| E-Mail | mamuruheart@gmail.com | | | | 電話 098-987-4323 |
|  | | | | | |
| 年 | | 月 | 学　歴 ・ 職　歴（各別にまとめて書く） | | |
| 2009 | | 4 | 南屯門官立中學学校 入学 | | |
| 2012 | | 3 | 南屯門官立中學学校 卒学 | | |
| 2012 | | 4 | 嶺南大學、データサイエンス学部・コンピュータ工学科 入学 | | |
| 2016 | | 3 | 嶺南大學、データサイエンス学部・コンピュータ工学科 入学(学士 取得) | | |
| 2016 | | 5 | 屯門中銀理財中心(BOCHK.comの支社) 入社 | | |
| 2019 | | 4 | 退社 | | |
| 2019 | | 5 | Drape Fit Inc.US (https://www.drapefit.com/) | | |
| 2021 | | 12 | 退社 | | |
| 2022 | | 1 | Fora Soft (香港, 屯門）(https://www.forasoft.com/) | | |
| 2023 | | 5 | 退社 | | |
| 2023 | | 6 | バリューアプリケーションズ株式会社(https://valueap.jp/) | | |
| 2025 | | 4 | 退社 | | |
|  | |  |  | | |
|  | |  |  | | |
|  | |  |  | | |
|  | |  |  | | |
|  | |  |  | | |
|  | |  |  | | |
|  | |  |  | | |
|  | |  |  | | |

|  |  |  |
| --- | --- | --- |
|  | | |
| 年 | 月 | 免　許 ・ 資　格 |
| 2016 | 3 | コンピュータ工学学士 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | | | |
| 志望の動機、特技、アピールポイントなど | | | | |
| 私は、Webサービス開発において、最新技術を積極的に取り入れながら、効率的かつ高品質な成果を生み出すことを得意としております。特に、ReactやVue.js、そしてExpress、Laravel使ったフロントエンド・バックエンド両方の開発を得意としております。また、サブリーダとして要件定義から実装・テスト、運用に至るまで、一貫した対応を行ってまいりました。チームワークを大切にし、円滑なコミュニケーションを図りながらプロジェクトを進めることで、常に納期を厳守し、顧客満足度の高いサービスを提供してきた実績がございます。  AIサービスシステムの開発にも携わってまいりました。具体的には、NLPのQAモデル、LMMのChatGPTモデルをカスタマイズし、医療サービスや検索システムといった専門的な業務システムの開発を行っておりました。  御社のさらなる成長に貢献できるよう、これまで培ってきた知識と経験を活かし、常に最新技術の動向にアンテナを張りながら、誠心誠意取り組んでまいります。 | | | | |
|  | | | | |
| 本人希望記入欄（特に待遇・職種・勤務時間・その他についての希望などがあれば記入） | | | | |
| 待遇面については貴社の規定に準じますが、面接時にご相談させていただければ幸いです。コンピュータ工学を専攻しておりますので、今後もソフトウェア開発に携わり続けたいと考えております。  勤務時間については、1日8時間、週5日程度で働くことを希望しています。 | | | | |

////

職 務 経 歴 書

2025年5月13日現在

氏名　南出 創

■職務要約

香港にある屯門中銀理財中心に入社後、約9年間にわたりシステムエンジニアとしてWebシステム開発に携わってまいりました。中規模銀行向けサービスシステム、医療サービスシステム、ならびに商品の販売・仕入管理システム、保育管理の開発を担当いたしました。また、AIモデルを活用した検索システムの開発にも関与し、検索の正確性を大幅に向上させることに貢献いたしました。プロジェクトの要件定義書の作成から製品リリースまでの一連の工程に豊富な経験があり、あらゆるプロジェクトに柔軟に対応できる点が強みです。

■活かせる経験・知識・技術

・コンピュータ工学を専攻

・約8名規模のサブリーダー及び開発協力能力を経験

・データの処理と分析知識

・JavaScript・Typescript/PHP/Python/SQL言語を知識

・Laravel/React/jQuery/Django/FastAPI によるフレームワーク/ライブラリ知識

・MySQL/PostgreSQL/MongoDB/Cloud DBMSデータベースの利用技術

・OpenAI/GPT/LLM/OpenCV/PaddleOCRを活用

■職務経歴

|  |  |  |
| --- | --- | --- |
| 2023年6月～2025年4月／ バリューアプリケーションズ株式会社 | 開発環境 | 規模 |
| 【プロジェクト概要】  保育および介護業界向けのアプリ開発に携わり、スマーホとWebの両プラットフォームで利用可能なサービスを構築しました。ReactやIonicを用いたハイブリッドアプリ、Node.js + Expressを活用したバックエンド、およびAWSインフラによる信頼性と拡張性のあるシステムの構築を行いました。  【担当フェーズ】  ・Ionic + Reactを用いたマルチプラットフォーム対応のUI設計と開発 ・Node.js + Expressを使用したAPI開発およびバックエンドロジックの実装  ・AWS EC2を用いたアプリケーションサーバ構築・運用  ・AWS S3によるユーザー生成コンテンツ（画像・動画等）の保存・配信対応  ・全体的なシステムアーキテクチャ設計と保守  【業務内容】  ・ReactおよびIonicを用いたユーザー向けアプリケーションのUI実装 ・ExpressベースのREST API設計・開発、認証・認可機能の実装 ・AWS EC2へのデプロイ自動化、SSL設定、ロードバランシング対応 ・AWS S3を使用したメディアファイル管理ロジックの実装 ・GitHubを使用した開発の品質とスピードを両立させるためにCI/CD環境の導入  【実績・取り組み】  ・Ionicを採用することでiOS・Androidの両OSに対応したアプリを短期間で開発し、クライアントからの満足度を大きく高めました。 ・AWS S3と連携したメディア保存機能では、セキュリティ面とコスト効率を両立させる構成を提案・実装。ユーザー数が増加しても安定運用可能なシステムを実現しました。 ・バックエンドはNode.js + Expressで構築し、非同期処理を駆使したパフォーマンス最適化を行い、リクエスト応答時間を約30%短縮。 ・Reactによるモジュール分割とコンポーネント設計により、保守性と拡張性に優れたフロントエンドアーキテクチャを構築しました。 | 【言語】  JavaScript  【OS】  Ubuntu 22.04  【DB】  MongoDB  【フレームワーク】  React  Express  Ionic  【Cloud】  AWS EC2  AWS S3 | 全3名 |
| 2022年1月～2023年5月／ Fora Soft (香港, 屯門） | 開発環境 | 規模 |
| 【プロジェクト概要】  遠隔医療プラットフォームのWebシステム開発に携わり、薬剤検索向けの全文検索アルゴリズムやAI検索アルゴリズムの構築を中心に、フロントエンド開発やバックエンドとの統合にも関与してまいりました。  【担当フェーズ】  ・ユーザーのニーズ、要件分析  ・Material UIを用いてフロントエンド開発  ・PostgreSQLを用いて全文検索アルゴリズム構築  ・OpenAIのChatGPTを用いてAI検索アルゴリズム構築  ・Python、Djangoを使ってバックエンドとの統合  【業務内容】 ・PostgreSQLを使用してデータベースのインデックス設計  ・AI検索方法を使用して薬剤検索正確性を向上  ・Material UIを使用して入力フィールドの設計、ローディングスピナー  ・AWS API Gatewayを使用してバックエンドと統合、API最適化  【実績・取り組み】  携わったプロジェクトでは、AIを核とした薬剤検索システムの開発に挑戦しました。特に、遠隔医療システムへの応用を視野に入れ、患者さん一人ひとりの診療情報を反映した「特別なプロンプト」を設計。これにより、AIに与える問いかけを高度にカスタマイズし、それぞれのケースに最適化されたモデル構造を緻密に組み上げました。このプロジェクトを通じて、AIプロンプトの設計からモデルアーキテクチャの最適化、そして複数検索手法の融合まで、一貫して “患者ファースト” を貫いたシステム構築の経験を深めることができました。今後はさらに医療データの多様化に対応し、よりスマートで安全な遠隔医療インフラの実現を目指していきたいと考えています。 | 【言語】 Typescript  Python  【OS】  Ubuntu 22.04  【DB】  PostgreSQL  【フレームワーク】  Next.js  Django  【AIモデル】  QA  (RealTime QA)  ChatGPT  (GPT-3.5 Turbo)  【Cloud】  AWS EC2  AWS Lambda | 全51名 |
| 2019年5月～2021年12月/ Drape Fit Inc.US |  |  |
| 【プロジェクト概要】  国外市場向けのEコマースサイトの開発において、集中型システムアーキテクチャのスケーラビリティを高めるために、分散型システム構成を導入。  【担当フェーズ】  ・TypeScript、React、Tailwind CSS によるフロントエンド開発  ・Node.js, Express, MongoDBによるバックエンド開発  ・API駆動型アーキテクチャ作成  ・RESTful APIの設計、認証・認可、エラーハンドリング機能を実装  ・GitHub CI/CDを用したDigitalOcean環境でのシステム構築・デプロイ  【業務内容】  ・API駆動型アーキテクチャの実装に向けたドキュメント作成  ・MongoDBを使ったデータベース構築してnode.jsを使用  ・JavaScriptのライブラリReactを活用しフロントエント開発  ・証明および権限管理のための適切なロールベースアクセス制御（RBAC）の　　実装  【実績・取り組み】  システムの規模拡大に向けて、API駆動型分アーキテクチャの実装し、適切なデータベースフォーマットを採用することで、システムのパフォーマンス向上に貢献しました。 | 【言語】  Typescript  【OS】  Ubuntu 22.04  【DB】  MongoDB  【フレームワーク】  React.js  Tailwind  Express  【Cloud】  DigitalOcean  Droplet | 全8名 |
| 2016年5月～2019年4月／屯門中銀理財中心 (BOCHK.netの支社） | 開発環境 | 規模 |
| 【プロジェクト概要】  中規模銀行向けのサービスシステムの開発に従事し、顧客の要望に応えるために、ユーザーフレンドリーで直感的なインターフェースの設計・実装に尽力しました。  【担当フェーズ】  ・要件定義  ・ニュザインタフェイース（UI）デザイン  ・フロントエンド開発  ・React.jsとAxiosを用いてREST APIとの連携を行う  【業務内容】 ・ユーザーのニーズを重視した要件定義  ・顧客要望に基づく画面設計変更や機能改善提案  ・JavaScriptなどのプログラミング言語や Bootstrap 、React.js使用したフロントエンド開発  ・React HooksやRedux toolkitを活用して、効率的な状態管理を実施  ・Laravel使用したインターフェースとバックエンドの統合  【実績・取り組み】  銀行の顧客のニーズとビジネス目標を理解するために、潜在的な顧客を調査し、彼らの痛点や期待を特定しました。その後、インターフェースを設計するために、ユーザーの特定の機能要件を収集し、画面仕様を完成しました。これに基づいて。フロントエンド開発やAPI統合を行ってきました。結果として、銀行業務の処理速度が向上し、サービス全体の効率化と顧客満足度の向上に貢献することができました。 | 【言語】  JavaScript  【OS】  CentOS  【DB】  MySQL  【フレームワーク】  Bootstrap  React.js  Laravel  【Cloud】  Azure | 全10名 |

■テクニカルスキル

|  |  |  |  |
| --- | --- | --- | --- |
| 種類 | | 使用期間 | レベル |
| OS | Windows | 9年 | 使用可能 |
| Linux (Ubuntu, CentOS) | 6年 | 使用可能 |
| 言語 | JavaScript | 9年 | 最適なコード記述と、指示、改修が可能 |
| Typescript | 5年 | 基本的なプログラミングが可能 |
| PHP | 8年 | 最適なコード記述と、指示、改修が可能 |
| Python | 4年 | 基本的なプログラミングが可能 |
| SQL | 7年 | 基本的なクエリ作成可能 |
| DB | MySQL | 10年 | 基本的な環境構築が可能 |
| PostgreSQL | 9年 | 基本的な環境構築が可能 |
| MongoDB | 9年 | 基本的な環境構築が可能 |
| フレームワーク | Bootstrap | 10年 | 最適なコード記述と、指示、改修が可能 |
| jQuery | 10年 | 最適なコード記述と、指示、改修が可能 |
| Laravel | 9年 | 基本的なプログラミングが可能 |
| React | 6年 | 最適なコード記述と、指示、改修が可能 |
| Vue | 4年 | 基本的なプログラミングが可能 |
| Django | 6年 | 基本的なプログラミングが可能 |
| FastAPI | 5年 | 基本的なプログラミングが可能 |
| AI関連ライブラリ | OpenAI | 5年 | 最適なコード記述と、指示、改修が可能 |
| OpenCV | 8年 | 最適なコード記述と、指示、改修が可能 |
| PaddleOCR | 3年 | 基本的なプログラミングが可能 |
| Cloud | AWS | 5年 | 環境設計・構築が可能 |
| Azure | 8年 | 環境設計・構築が可能 |
| Firebase, | 3年 | 環境設計・構築が可能 |

■資格

|  |  |
| --- | --- |
| コンピュータ工学学士 | 2016年3月取得 |

■自己PR

私は、JavaScriptフレームワークをはじめとするWebサービス開発における最新技術に特に精通しており、どのようなご要望にも柔軟に対応できる力を発揮することで、貴社に貢献できると確信しています。  
2016年以降、中規模プロジェクトのみならず、大規模プロジェクトにも携わり、Webサービス開発において豊富な経験を積んできました。2020年から、オープンAIモデルを用いたプロジェクトや、学習済みAIモデルのカスタマイズにも携わっています。約9年間の業務を通じて、チームワークやコミュニケーション能力を向上させ、常に納期を厳守する意識を持って業務に取り組んでいます。また、自身のスキル向上にも力を注いでおり、会社にとって欠かせない存在となるべく、常に新しい技術やトレンドにアンテナを張り、継続的な自己研鑽を大切にしています。

以上

# **Copyright(c) Japan Online School coporation**

# **Frontend Appealing**

* problem in SPA?
* <provider>, <authprovider>, <suspense>
* Business understanding, Payment, security for E-commerce
* Cache in local storage

Backend Appealing

Validation

# **AI dental keyword**

**1-34-year-old patient, 55 kg, with Crohn's disease taking immunosuppressants(免疫「めんえき」を抑えるやくです。). Requiring multiple dental extractions.  
2-88-year-old patient with Alzheimer’s disease on unknown medication, 75 kg, with treated prostatic hypertrophy. Requiring root canal treatment.  
3- 33-year-old patient, 54 kg, taking oral contraceptives and oral antiplatelet agents. Requiring dental implant placement.  
4- 55-year-old patient, 90 kg, with hypercholesterolemia and treated hypertension. Requiring tooth preparation for crown placement.  
5- 45-year-old anticoagulated patient on Eliquis, 79 kg, allergic to NSAIDs and penicillins, requiring dental extraction.**

**Full stack engineer including AI()**

# **Development version**

| **技術** | **最新バージョン** | **リリース日** | **備考** |
| --- | --- | --- | --- |
| **PHP** | 8.4.5 | 2025年3月13日 | PHP 8.4系の最新セキュリティリリース。PHP 8.5は2025年11月にリリース予定です 。 |
| **React** | 19.1.0 | 2025年3月 | React 19系の最新バージョン。新機能や改善点が含まれています 。 |
| **Laravel** | 12.x | 2025年3月7日 | Laravel 12がリリースされ、新しいスターターキットや依存関係の更新が行われました 。 |
| **Python** | 3.13.3 | 2025年4月8日 | Python 3.13系の最新バージョン。複数のバージョンが同日にリリースされています 。 |
| **JavaScript** | ECMAScript 2024 (ES15) | 2024年6月 | ECMAScriptの最新バージョンで、JavaScriptの新機能が追加されています 。 |
| **Vue.js** | 3.6 (予定) | 2025年内予定 | Vue 3.6のリリースが予定されており、新機能や改善点が含まれる予定です 。 |
| **Node.js** | 23.6.1 | 2025年1月21日 | Node.js 23系の最新バージョン。セキュリティアップデートが含まれています 。 |
| **Django** | 5.2 | 2025年4月2日 | Django 5.2がリリースされ、新機能や非互換の変更点が含まれています 。 |

# In conclusion

I’m a freelancer.so, I am available to respond requests from your company at any time. With my experience as a full-stack engineer, I have the skills and flexibility to handle a wide range of projects with precision and care. I take great pride in delivering high-quality results and building long-term trust with my clients. I am confident that I can contribute positively to your company.

私は現在フリーランスとして活動しており、御社からのご依頼には・**いつでも**迅速に対応させていただくことが可能です。また、フルスタックエンジニアとしての経験を活かし、様々のプロジェクトにも柔軟的に対応することができます。私は御社に積極的に貢献できると確信しております。

# Novik

## English

I’ve been involved in building several service systems for medical institutions—things like hospitals, clinics, and even pharmacies. Today, I’d like to walk you through one particular project I handled as a full‑stack engineer, meaning I took it from frontend to the backend by myself.

*So, let me share my screen…*

Can you see my screen?  
Can you see it now? Great. This project is called **Novik**. It’s a smart AI‑powered dental assistant designed to help clinicians make safe, accurate treatment decisions. By analyzing a patient’s symptoms, it suggests the most appropriate treatment options.

First, let me log in using my Gmail account.

(…logging in…)  
Once you’re in, you’ll find the **Dashboard**, the **Home** page, and the **Contact** and **Partner** pages—these last two pages let customers and partners contact with our service.

Now, let me show you the main functionality. On the **New Patient** page, I can type something like “Hi.” Ok the system can response. result is out, It’s time to input patient’s real dental symptoms—and it requires their age and weight, since drug choices and usage can vary from person to person.

Ok， let me input． for example， 31 years old, 55Kg, for wemon， the most often question: taking oral **contraceptive**, requiring dental implant placement. **31 years old, 55Kg, taking oral contraceptive, requiring dental implant placement**

Because the AI model does its magic on the server, it might take a few seconds to respond…

…and here’s the result! As you can see, the result includes several categories:

* **Anesthetics（麻酔薬：ますいやく）**
* **Antibiotics（抗生物質：こうせいぶっしつ）**
* **Analgesics（鎮痛薬：ちんつうやく）**
* **Anti‑inflammatory drugs（抗炎症薬：こうえんしょうやく）**  
  …and so on. This result makes it really easy for a dentist to choose the right **medication**.

Then what will be happen if you input something that is not related to dental care? How can the system response? For example, who is the better, Ronaldo and Messi? The Novik didn’t respond correctly about the question —because our AI model is fine-tuned only for dental‑assistant tasks.

In order to develop the service system, I used **React** with **Tailwind CSS** on the frontend, and **Django** (Python) on the backend—mainly because we needed to integrate a custom AI search engine with backend. If you want to see the Source code, I can share the code. Do you agree. Please ask me if there are something you’d like to know.

Here’s a quick look at the workflow:

1. **Routing**: These are all the routes our app uses.
2. **State management**: With React hooks, user inputs are stored in state variables.
3. **Submission**: When you hit the “Enter” button, the input goes to our backend route.
4. **Processing**: On the dashboard route, the backend assembles the conversation context (we call that a “thread”) and then runs the AI model to generate a single response.
5. **Error handling**: If you type something that’s not related to dental care, Novik simply won’t respond—because our AI model is fine-tuned only for dental‑assistant tasks.

As you can see the input information is stored in Input variable using the hook React. If you click the button-Enter, the input information is delivered to backend with this route. Here is dashboard route. The function responses the right answer corresponding to question. First, create **thread** to representthe entire conversational context and then create run to generate a single assistant response within that thread.

If you input some content which is not related to dental assistant, then the Novik can’t response, that means the professional AI model is only adjusted to dental assistant service. I think that’s it.

## Japanese

私はこれまで、病院（びょういん）やクリニック、薬局(やっきょく)といった医療機関向けのサービスシステムの構築に携わってきました。本日は、その中でも私がフルスタックエンジニアとして担当したプロジェクトの一つをご紹介したいと思います。

ちょっと画面を共有させていただきます。少々お待ちください。画面が見えていますか？

今はどうですか？ あっ、見えましたね、ありがとうございます。

このプロジェクトは”Novik”と呼ばれており、AIを活用したスマートな歯科（しか）用アシスタントです。（医師（いし）が安全かつ正確な治療（ちりょう）判断（はんだん）を行えるよう支援します。）このシステムは患者さんの症状（しょうじょう）を分析して、最適な治療オプションを提案してくれる仕組みです。ではまず、Gmailアカウントでログインしてみます。

（…ログイン中…）ログインしています。

ログインすると、新規患者（ダッシュボード（dashboard））、ホームページ、連絡先ページ、パートナー ページが表示されます。最後の 2 つのページでは、顧客やパートナーがこのサービスに問（と）い合わせることができます。それでは、メイン機能をご紹介いたします。

「新規患者」のページでは、たとえば「こんにちは」と入力して、システムは応答できます。結果が出ましたね。“こんにちは！私は「Novik」と申します。歯科医の先生方に、臨床（りんしょう）的な推奨（すいしょう）を提供するために設計されたAIアシスタントです。”

それでは、実際の歯科症状（しょうじょう）を入力してみます。

このシステムでは、薬（くすり）の選択（せんたく）や用量（ようりょう）が個人によって異（こと）なるため、年齢と体重を入力していた必要です。例えば、英語でいうと**31 years old, 55Kg, taking oral contraceptive, requiring dental implant placement**「31歳・55キロの女性（じょせい）で、経口（けいこう）避妊薬を服用（ふくよう）中、インプラント治療を希望」/というような情報を入力しました。

AIのモデルはサーバー側で処理を/行うので、少し時間がかかることもありますが……  
はい、結果が出ました！

ご覧（ごらん）のとおり、結果はいくつかのカテゴリに分かれて表示されます。

* **Anesthetics（麻酔薬：ますいやく）**
* **Antibiotics（抗生物質：こうせいぶっしつ）**
* **Analgesics（鎮痛薬：ちんつうやく）**
* **Anti‑inflammatory drugs（抗炎症薬：こうえんしょうやく）**

もし、具体的くすりを押すと詳細的説明が出ました。

この結果があることで、歯科医は適切な薬（くすり）を簡単に選ぶことができます。

では、歯科とは/関係のない内容を入力したら、どうなるでしょうか？たとえば、「ロナウドとメッシ、どっちがすごい？」といった質問を入れてみます。このような質問には、Novikは正確に応答（おうとう）できません。というのも、このAIモデルは歯科アシスタント業務に特化してファインチューニングされているためです。このサービスシステムの開発には、（フロントエンドにReactとTailwind CSS、バックエンドにはDjango（Python）を使用しました。理由としては、バックエンドにカスタムAI検索エンジンを組み込む必要があったためです。）もしソースコードをご覧になりたい場合は、共有することも可能です。このサービスシステムでご不明な点がございましたら、どうぞお気軽（きがる）にお問（と）い合わせください。

# 銀行向けサビスシステム

## Why is CentOS used for medium bank?

### English

In my experience developing service systems for financial institutions, **CentOS** has often been the preferred choice — especially for **medium to large banks**.

One major reason is **stability**. Unlike some operating systems that change rapidly, CentOS (and its upstream, RHEL) provides long-term support and predictable behavior, which is critical for banking systems that require maximum uptime.

Another reason is **security**. CentOS comes with built-in **SELinux**, which allows fine-grained access control. This is very useful in regulated industries like finance, where data protection and system auditing are essential.

Also, CentOS is highly **compatible with enterprise tools** — such as Oracle Database, SAP, and other middleware that banks often rely on. These tools are usually certified for RHEL-based environments.

Compared to Ubuntu or Windows, CentOS is more **minimal**, reducing the risk of unnecessary packages or services running in production. And with proper hardening, it becomes highly resilient against external threats.

Overall, I think CentOS — or its modern alternatives like **AlmaLinux or RHEL** — is ideal for building reliable, secure, and compliant systems in the banking sector.

### Japan

金融（きんゆう）機関向けのサービスシステムを開発する中で、**CentOS** はよく選ばれるOSだと感じています。特に**中規模から大規模の銀行**では、その傾向（けいこう）が強いです。

理由のひとつは、まず **安定性**です。CentOS（および上位互換のRHEL）は長期サポートを提供しており、動作も予測(よそく)しやすいため、**常に稼働が求められる銀行システム**には非常に適しています。

次に、**セキュリティ**です。CentOSには **SELinux** が標準で組み込まれており、細かい（こまかい）アクセス制御（せいぎょ）が可能です。これは、情報漏えい（ろうえい）のリスクが許（ゆる）されない金融業界において、大きなメリットです。

また、CentOSは **企業向けのツールやミドルウェア**と高い互換性（ごかんせい）があります。たとえばOracleやSAPなど、多くの銀行が使っている製品は、RHEL系のOSでの動作が公式にサポートされています。

UbuntuやWindowsと比べ（くらべ）ても、CentOSは必要最低限（さいていげん）の構成になっているため、**不要なサービスやパッケージを排除（はいじょ）しやすく、セキュリティ強化（きょうか）がしやすい**点も魅力です。

そのため、信頼性が高く、セキュリティ対策（たいさく）も万全（ばんぜん）で、このCentOSは銀行のような業界にとって適切な選択だと思っています。

## 要件定義とUI設計

### English

### Japan０

#### 要件定義

UI設計は要件定義、画面仕様、ワイヤーフレーム(wireframe)、モックアップの各段階を経（へ）て・進めてまいりました。

要件定義の段階では、システムが解決すべき業務上の課題（かだい）を明確し、機能要件および非機能要件を整理することが求められます。簡単に言うと、**ユーザー視点**（たとえば顧客や窓口（まどぐち）の担当者）と、**システム視点**（パフォーマンスやセキュリティなど）の両方から要件を確定していくことが重要です。例えば、**機能要件**としては、**顧客ログイン**（customer login）、**残高照会**（ざんだかしょうかい balance inquiry）、**振込・振替**（ふりこみ、ふりかえ Funds transfer）、**取り履歴**（transaction history）、そして**口座（**こうざ**）開設** ―＞Account Opening）などがあります。

また、非機能要件については、システムの裏側（[うらがわ]）で保証（ほしょう）されるべき性能、たとえば―「200 ms以内の**応答**すること」や、「高負荷（こうふか）時でも500ユーザーが**同時**に**接続（**せつぞく**）**できること」、**監査**（かんさ）**ログ**の**取得**、99.9%の**可用性**―を確保することが求められます。こうした点も考慮する必要があります。

要件定義を進める際には、具体的な手法(しゅほう)として、**ヒアリングシート**（hearing sheetインタビューシート:interview sheet）や**業務フロー図(ず)**（BPMN）、**ユースケース図**（Use Case Diagram）を作成する方法が有効（ゆうこう）です。さらに、ユーザーストーリーマッピング（User Story Mapping）を活用して、ユーザー視点で、どの機能を優先するか決めたり、JiraやBacklogなどのツールを使って、最初の要望（ようぼう）がちゃんと反映（はんえい）されているかを確認することも大切です。

アジャイル（agile）開発でよく使われるJiraを利用し、スプリント(sprint)計画(けいかく)やタスクのステータス管理（To Do、In Progress、Done）を行っています。

Backlogでは、タスクのチケット管理やガントチャート（Gant chat）を活用し、進捗を可視化（かじか）して開発のスケジュールを管理しています。

**監査ログの主な目的：**

「監査ログの取得（かんさログのしゅとく）」とは、**システムの中で「誰が・いつ・何をしたか」を記録（きろく）すること**です。

**セキュリティの確保：**  
誰が、いつ、何をしたかを追跡（ついせき）できるようにして、不正アクセスや操作ミスの調査が可能になります。

**トラブルシュート：**  
障害（しょうがい）が発生した際、原因を特定するための手がかりになります。

**コンプライアンス対応：**  
金融・医療・行政などの業界では、操作記録を保存することが法律や業界基準で義務付けられている場合があります。

| **ID** | | | **機能名 (Feature)** | **概要 (Description)** |
| --- | --- | --- | --- | --- |
| FR-01 | 顧客ログイン (Customer Login) | | | ID/パスワードでの認証。二要素認証（OTP）対応。 |
| FR-02 | | 残高照会 (Balance Inquiry) | | ログイン後、各口座(かくこうざ)の現在残高を表示。 |
| FR-03 | | 振込・振替 (Funds Transfer) | | 自己・他行（たこう）あて振込。手数料（てすりょう）自動計算、振込依頼キャンセル機能。 |
| FR-04 | | 取引履歴 (Transaction History) | | 過去（かこ）3ヶ月分の入出金履歴を検索・絞り込み（期間／金額／種類）して表示。 |
| FR-05 | | 口座開設申請 (Account Opening) | | オンラインで必要情報を入力し、新規口座開設を申請。本人確認書類のアップロード機能付き。 |

#### 画面仕様例（UI）

画面仕様は、本システムにおける各画面の基本情報を整理するために必要な仕様です。各画面について、画面ID、画面名、主な要素、バリデーションや表示条件などを記載（きさい）し、設計や実装の指針（ししん）となるよう/構成いたします。このサービスシステムにおいて、基本的に３つの画面がありまして、たとえば、ログイン画面、残高照会画面、振込依頼画面です。

要件定義・画面仕様の作成に基ついて、直感的な**ワイヤーフレーム（wireframing）**や**モックアップ(mockup)**を作成しました。色彩 [しきさい]、タイポグラフィ(typography)、アイコン(icon)などの選定にこだわることで、最終的には、見た目の魅力と使いやすさの両立を意識しながら、UIを完成させました。」

| **画面ID** | | **画面名** | **主な要素** | | **バリデーション・表示条件** |
| --- | --- | --- | --- | --- | --- |
| SCR-01 | | ログイン画面 | ユーザーID入力欄、パスワード入力欄、ログインボタン 等 | | **空欄チェック**、**ID/パスワード不正時はエラー表示** |
| SCR-02 | 残高照会画面 | | 口座一覧テーブル、残高、通貨単位 | | ログイン必須。データが0件時は「該当データなし」表示 |
| SCR-03 | 振込依頼画面 | | | 振込先リスト、金額入力、手数料表示、実行ボタン | 金額は1,000円以上。上限は本人設定限度額まで |

**空欄チェック：入力欄が空白でないことを確認する**

**IDやパスワードが誤っている場合：エラーメッセージを画面に表示**

**UI Example (ワイヤーフレーム例：wireframe)**

**以下は 振込依頼画面 の簡易ワイヤーフレーム例です。**

**+──────────────────────**

**| [Bank Logo] [Help] [Logout]|**

**+──────────────────────+**

**| From Account: [普通 123-4567890 v] |**

**| To Account: [三菱UFJ 銀行 ○○支店 普通 0987654]▽|**

**| Amount: [¥ \_\_\_\_\_\_\_\_\_ ] 手数料: ¥ 210 |**

**| Note: [\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_] |**

**| |**

**| [Calculate Fee] [Submit Transfer] |**

**+───────────────────────+**

**| ← Back Next → |**

**+────────────────────────+**

## **Frontend実装０**

はい、私は銀行向けのWebシステム開発において、主にフロントエンドの設計・実装を担当しておりました。具体的には、ユーザー中心のUI設計を心がけ、要件定義の段階から画面仕様書の作成、ワイヤーフレームやモックアップの作成まで一貫して関わりました。

フロントエンドの実装段階ではまず、React.jsを用いて、ボタンやフォームなどの再利用可能なコンポーネントを設計し、さらにReact HooksやRedux Toolkitで効率的な状態管理を行っておりました。また、バックエンドとはAxiosを使って・REST APIで連携し、ログイン情報、口座残高、取引履歴などのデータを・リアルタイムで画面へ表示しています。

特に、アクセシビリティやレスポンス速度にも配慮（はいりょ）しながら、使いやすく・信頼性のあるフロントエンドを目指して開発してまいりました。

**アクセシビリティ向上のためにできること**

| 対策内容 | **説明** |
| --- | --- |
| HTMLの意味的タグを正しく使う | <header>, <nav>, <main>, <button>, <label>などの適切なタグを使うことで、スクリーンリーダーが正しく情報を読み取れます。 |
| ARIA（アリア）属性（ぞくせい）の活用 | aria-label, aria-hidden, aria-expandedなどを用いて、補助技術に情報を提供します。 |
| キーボード操作への対応 | タブキーだけで主要な機能にアクセスできるようにし、フォーカスインジケーター（outline）も見えるようにします。 |
| コントラスト（contrast）比（ひ）の確保 | 背景色と文字色のコントラストを十分に高く（WCAGの基準：4.5:1以上）保ちます。 |
| 代替テキストの設定 | 画像にalt属性を設定し、視覚に頼らず情報を伝えられるようにします。 |
| フォームラベルの明示 | <label for="id">を使ってフォームとラベルを関連づけます。 |

**レスポンス速度（表示・操作の速さ）を上げる方法**

| 対策内容 | **説明** |
| --- | --- |
| コード分割（Code Splitting） | ReactではReact.lazy()やdynamic import()を使って、必要なコードだけを読み込むことで初期（しょき）ロードを速く（はやく）します。 |
| 画像の最適化 | サイズを小さくし、WebP形式やLazy Loadingを使って表示速度を向上させます。 |
| 不要な再レンダリングの防止（ぼうし） | React.memo, useMemo, useCallbackを適切に使用して無駄（むだ）な処理を減ら（へら）します。 |
| CDN利用 | 静的ファイル（画像、JS、CSSなど）をCDNから配信することで、読み込み速度を向上させます。 |
| 圧縮とミニファイ | JavaScriptやCSSファイルをminifyし、データ量を削減します。 |
| キャッシュの活用 | Service WorkerやHTTPキャッシュ設定を適切に活用することで、再訪問時のパフォーマンスを改善します。 |
| APIレスポンスの最適化 | バックエンド側でAPIのレスポンス速度を向上させることで、画面の反応（はんのう）も速くなります。 |

## **API統合０**

**API（Application Programming Interfaceの略）統合**とは、外部システムや内部サービスと連携するために、**フロントエンド（画面側）からバックエンドのデータや機能にアクセス**する仕組みを実装することです。つまり、**ユーザーインターフェースとサーバー側ロジックの橋渡し（はしわたし）**を行ったということです。

API統合においては、**React.jsとAxios**を用いてREST APIとの連携を行いました。例えば、口座情報や取引履歴などを取得するために、API提供するエンドポイントに対して非同期リクエストを実装しました。その後、バックエンドから取得したデータを React Hooks や Redux Toolkit を用いて状態管理し、コンポーネントに反映させて画面に表示しています。また、JWTトークンを用いた認証付きAPIの統合にも対応し、セキュアなデータ取得と更新を可能にしました。

バクエンドについて、Laravelフレイムワクを使用しました。Laravelをバックエンドに採用し、RESTful APIを設計・実装することで、React.jsベースのインターフェースとシームレス（seamlessly）に統合された口座管理システムを構築しました。Eloquent ORM(エロクアント)を活用してMySQLとのデータ連携を行い、口座情報の取得・更新・削除などの処理を効率的に行える仕組みを実現しました。

## Laravel利用

### English

In Laravel, I apply the MVC architecture effectively. The framework provides strong support for organizing logic into Models, Views, and Controllers.

I define database schemas using Migrations and associate them with Eloquent Models, where each column can be customized with specific properties and relationships. This structure not only improves maintainability but also supports features like soft deletes, timestamps, and model events.

The View layer is typically handled with Blade templates under the resources/views directory. Blade makes it easy to create dynamic and reusable UI components while keeping the logic clean and readable.

Controllers act as intermediaries between the View and the Model. They handle HTTP requests, validate input using Laravel’s powerful validation system, and interact with the database through the Eloquent ORM.

Additionally, I make full use of Artisan commands to efficiently generate Models, Migrations, Seeders, and Controllers. Artisan accelerates development and encourages clean code architecture from the beginning of the project.

Overall, Laravel helps me develop scalable, secure, and testable web applications by providing a robust framework with well-structured conventions.

### Japanese

Laravelでは、MVCアーキテクチャを効果的に活用しながら、保守性の高いWebアプリケーションを開発しています。

モデル（Model）では、マイグレーション(migration)を通じてデータベースのスキーマを定義し、それぞれのカラムにプロパティ(property)やリレーション(relation)を設定できます。Eloquent(エロクアント)を活用することで、ソフトデリート(soft delete)やタイムスタンプ、モデルイベント(model envent)など、柔軟で拡張性の高い設計が可能です。

ビュー（View）は主に resources/views(リソース スラッシュ ビューズ) 配下（はいか）のBladeテンプレートで管理しています。Bladeを使うことで、ロジックとデザインを分離しつつ、再利用性の高いUIコンポーネントを効率的に構築できます。

コントローラー（Controller）は、ビューとモデルを連携を担当、HTTPリクエストの処理、バリデーション、データベースとのやり取りを一貫して行います。Laravelのバリデーション機能を活用することで、安全で信頼性の高い入力処理が可能です。

さらに、Artisan(アーティザン)コマンドを使用して、モデル・マイグレーション(migration)・シーダー(seeder)・コントローラー(controller)を迅速に生成し、開発スピードを高めつつ、コードの統一性とクリーンアーキテクチャ(clean architecture)を維持(いじ)しています。

Class UserTabel extends Migrations

Laravelの豊富な機能と設計思想（しそう）により、スケーラブルでセキュア、かつテストしやすいアプリケーション開発を実現しています。

Laravelフレイムワクを使用しました。Laravelをバックエンドに採用し、RESTful APIを設計・実装することで、React.jsベースのインターフェースとシームレス（seamlessly）に統合された口座管理システムを構築しました。Eloquent ORM(エロクアント)を活用してMySQLとのデータ連携を行い、口座情報の取得・更新・削除などの処理を効率的に行える仕組みを実現しました。

# バリューアプリケーションズ株式会社

Reactを用いてフロントエンドを開発していた際、子どもの動画や写真を共有する機能において、リアルタイム性の保証という課題が発生しました。最初は一般的な方法でファイルをアップロードしていましたが、20MB程度の動画ファイルでは、再生のためのダウンロードに5秒以上かかり、ユーザー体験に影響していました。

そこで、まず高速なプレビュー表示のためにサムネイル画像を生成して読み込むようにしました。さらに、動画を全体ダウンロードするのではなく、視聴している部分だけを読み込めるようにHTTPレンジリクエストを使ったストリーミング再生を実装しました。その結果、動画の初期表示までの遅延を1秒以下に抑えることができ、リアルタイム性の高い動画共有が実現できました。

## 実装

このプロジェクトでは、**Ionic + Reactを用いたハイブリッドアプリ**のフロントエンド開発を担当し、iOS・Android両方に対応するマルチプラットフォームアプリを開発しました。UIは保護者や保育士（ほいくし）の方が直感（ちょっかん）的に操作できるよう、アクセシビリティとユーザビリティを重視しました。

**アクセシビリティ（Accessibility）：** **誰（だれ）もがアクセスしやすいように配慮すること**です。

**ユーザビリティ（Usability）：** **使いやすさや操作のしやすさ**を意味します

### Ionicによるアプリ開発0

**Ionic（アイオニック）** は、**モバイルアプリやWebアプリを効率的に開発できるオープンソースのフレームワーク**です。  
主に **HTML・CSS・JavaScript（またはTypeScript）** をベースにしており、**Angular（アンギュラー）、React、Vue（ビュー）などのフロントエンドフレームワークと連携**して使用することができます。  
特徴（とくちょう）として、**iOSとAndroidの両方に対応したクロスプラットフォーム（cross-platform）開発が可能**で、**ネイティブのようなUIコンポーネント**を簡単に実装できる点が挙げ(あげ)られます。

🔹ステップ①：環境構築について

Node.jsとnpmをインストールし、その後、Ionic CLIをグローバルにインストールしました。また、AndroidアプリのビルドにはAndroid Studio、iOSアプリにはXcodeを使用しました。

🔹ステップ②：プロジェクトの作成

プロジェクトの作成には、Ionic CLIの ionic start コマンドを使用し、blank（ブランク） テンプレート（template）を選択しました。（例：ionic start myApp blank --type=angular）

🔹ステップ③：開発（ページとUIの実装）

開発では、Ionicが提供するUIコンポーネント（IonButton、IonCardなど）を活用し、ユーザーフレンドリーな画面を構築しました。  
ページの作成には ionic generate page を使用しました。(generate: ジェネレート)  
たとえば、ionic generate page home で「ホーム」画面を追加し、その中にボタンやリストを配置しました。

**🔹ステップ④：動作確認・ローカルテスト**

作成したアプリは ionic serve を使ってローカルサーバー上で確認しました。  
また、Capacitor（**カパシター**）を使用してAndroid Studioと連携し、実機やエミュレーター(emulator)上でも動作検証(けんしょう)を行いました。

**🔹ステップ⑤：ネイティブ機能の実装**

カメラ機能などのネイティブ機能は、Capacitor(カパシター)プラグイン（plugin）を使用して実装しました。  
たとえば、@capacitor/cameraを使って、写真を撮影（さつえい）し、アプリ内で表示させる機能を実装しました。  
コマンドは npm install @capacitor/camera を使い、npx cap sync で同期しました。

**🔹ステップ⑥：本番ビルド・公開準備**

最後に、ionic build で本番用のビルドを行い、npx cap copy でネイティブプロジェクトにコピーしました。  
その後、Android Studioを使ってAPKファイルをビルドし、動作確認やストアへの提出（ていしゅつ）に備え（そなえ）ました。

### Backend implement0

バックエンドは**Node.js + Express**で構築し、REST APIの設計・実装を担当しました。特に、非同期処理を活用してリクエストの応答（おうとう）速度を約30%改善し、パフォーマンス最適化に大きく貢献しました。また、**認証・認可機能**も自ら（みずから）設計・実装し、セキュアなシステム構築に努め（つとめ）ました。

はい、フロントエンドとバックエンドの連携実装において、バックエンドに Next.js を使用する場合は API Routes を作成しました。次に、フロントエンドから fetch() （フェッチ）や axios（アクシィオス） を用いて、先ほど作成した API にリクエストを送信します。このとき、非同期処理を使ってデータを取得し、React の状態管理（useState や useEffect）を通じて、UI にデータを表示させます。また、追加処理について、認証（にんしょう）、エラー処理、読み込みインジケーター（indicator）を行います。

ログイン認証:JWTトークン認証とOAuth2認証方法を活用いたしました。

tokenやsession情報をrequest header（ヘッダー）に付与(fuyo)して送信し、バックエンド側でその認証情報を検証(kenshō)します。

エラー処理: backendの応答（response）をチェックし、errorが発生(hassei)した場合は、ユーザーに適切なエラーメッセージを表示できるようにしました。

**認可機能つきましてはRBACを**用いた実施いたしました。

### **Infrastructure implement0**

**インフラ面ではAWSを活用**し、EC2によるアプリケーションサーバの構築・運用、S3による画像（がぞう）や動画（どうが）などのユーザー生成コンテンツの保存と配信を行いました。S3との連携では、セキュリティとコスト効率を両立させる構成を提案し、ユーザー数の増加（ぞうか）にも耐（た）えられるスケーラブルな仕組みを実現しました。

また、GitHubによるバージョン管理とコードレビューの仕組みをチームに導入し、CIの自動化にも取り組むことで、開発フローの品質（ひんしつ）と効率を向上させました。

また、開発の品質とスピードを両立させるため、CI/CD環境の導入および運用を行いました。

このプロジェクトを通じて、**フロントエンドからバックエンド、インフラまで一貫して担当**し、ユーザー目線に立った開発と、スケーラブルかつ安全なシステム構築の重要性を学びました。特に、保育・介護という現場(げんじょう)で使われるプロダクトを支える開発には、高い信頼性と直感的なUIが求められることを実感しました。

御社のプロジェクトでも、これまで培ってきたフルスタックの技術力と、現場の課題解決に貢献してきた経験を活かし、価値あるプロダクト開発に貢献したいと考えております。

どうぞよろしくお願いいたします。

# **AI technique.（薬剤検索システム）**

## English

In 2023, I started working on a project to build an AI system that helps search for medicine. The first thing I did was choose the right AI model. At first, I used a **pre-trained** model made for answering questions — it was called **deepset**/roberta-base-squad2. In the beginning, it worked pretty well. But soon, I noticed some problems. Sometimes it couldn't find new drugs or give correct details, like how much to take or what ingredients were in the medicine. Since medical info has to be super accurate, I realized **fine-tunning** of this model is needed. To do that, I needed a lot of corpus, and collecting that would take a lot of time and effort. So instead, I tried using **multimodal-** **Mistral-Small**-3.1-24B-Instruct-2503 to the system. But sadly, that model didn’t perform well either. Finally, I focused on **OpenAI’s ChatGPT-4.0**. I created special prompts that included things like the patient’s age, weight, and symptoms. That made the answers much more accurate and useful. In the end, I was able to build a smart and helpful drug search system that gives better, more accurate results for users.

What kind of model did you use in ChatGPT and what is the result of the model?

There are many model in ChatGPT, I selected gpt-4o-mini. The result includes the name of drug, recommened dosage, warning, the function of drug ans so on. In particular, the result includes the URL adderss which include more detaild explanation of drugs according the user's needs.

How do you make the special prompt? there are two kind of prompt such as user prompt and system prompt. In user prompt, symptoms, attached image, attached file are included as array. In system prompt, the contents of the model's output are included as string. for example, the string of system is just like "the result of the model must be included reference link", "the result of the model must be included dosage of drug"

## **Japanese**

### AI abstract

御社でAI開発が重視されていることから、私がこれまで関わってきたプロジェクトの中から、AIに関連するプロジェクトについて、より具体的にご説明させていただきます。

私はDify（ディファイ）というLLMアプリ開発プラットフォームを活用して、ノーコードでチャットボットやAI業務支援ツールを構築した経験があります。（「Dify（ディファイ）」は、**オープンソースの生成AIアプリケーション開発フレームワーク**です。 ノーコードまたはローコードで、**チャットボットやカスタムAIアプリを簡単に作成・公開(こうかい)できる**のが特徴です。）

具体的には、OpenAIのGPTモデルを使ってFAQ対応システムやドキュメント検索アプリなどを開発しました。

たとえば、医療サービスを提供するFora Soft会社において、AIモデルを活用した薬剤検索システムや歯科医療サービなど、さまざまなAI関連プロジェクトに携わってまいりました。その過程を通じて、AIモデルおよびその応用に関して、一定の経験を積むことができました。

まず、AIモデルを使った開発においては、適切なAIモデルを選ぶことが第一工程だと思います。薬剤検索システムの開発について、事前学習済みモデルとしてNLPのQA(realtime QA)を選びました。最初はシステムの性能が比較的良かっだと思いますが、すぐにいくつかの問題点を発見しました。例えば、症状による最新薬剤が検索できないことがあったり、薬剤の一日使用量とか薬剤の成分が正しく表示されないことがありました。このような欠点（けってん）を克服（こくふく）し、薬剤検索の正確性を向上されるためには、モデルのファインチューニングが必要でした。そのために、薬物に関連するコーパスを取得し、ファインチューニングを実施しました。ただ、検索の正確性はあまり改善されませんでした。それで、マルーチモデルのMistral-Small-3.1-24B-Instruct-2503というモデルを使ってみました。ただ、残念ながらそのモデルも期待していた性能を出せませんでした。それに、最終的に、OpenAIの chatGPTに注目いたしました。このモデルを薬剤検索に適応するために、適切なプロンプトを作成しました。プロンプトではユーザプロンプトとシステムプロンプトガあり、ユーザプロンプトとして、患者さんの性別（せいべつ）や年齢（ねんれい）、体重そして症状を入力する必要があります。システムプロンプトとして、治療法や推奨薬剤を入力することができます。

以上のように患者さんの具体的症状情報をベースにプロンプトを作成することで、薬剤検索の正確性が大幅に向上し、このシステムを無事に完成することができした。

### LangChain

私が担当していた歯科医療サービスシステムもAI モデルを使用した別の例だと思います。このプロジェクトは”Novik”と呼ばれており、AIを活用したスマートな歯科用アシスタントです。このシステムは患者さんの症状（しょうじょう）を分析して、最適な治療オプションを提案してくれる仕組みです。この歯科サービスシステムに関して、「質問→回答」の形式で応答するQAモデルとして、deepset/roberta-base-squad2やBioGPTを試験的に使用してみました。しかしながら、このサービスシステムの性能を向上するために、ファインツニングが必要でした。でも、ファインチューニングにはすごくたくさんのコーパス（学習データ）が必要で、大量の歯科用コーパスを収集するのは難しかったです。代（か）わりに、多くな歯科関連ドキュメントファイルがあります。それはLangChain Frameworkを使うことに決めました。

LangChain Frameworkを使って、推奨薬や治療法がくつかのカテゴリに分かれて表示されます。

また、具体的くすりを押すと詳細的説明が出ました。

この結果があることで、歯科医は適切な薬（くすり）を簡単に選ぶことができます。

1. まず、Langchain（ランチェーン） frameworkを使って文書（ぶんしょ）を分割して**埋め込み**（うめこみ：embeded）ました。

Langchain framework？大規模言語モデル（LLM）を使ったアプリケーションを簡単に構築するためのPython/JavaScript向けのオープンソースフレームワークです。

LangChainの目的：簡単な「質問 → 回答（かいとう）」ではなく、LLMを“チェーン（Chain）”として組み合わせて、より複雑なタスクを実現することが目的です。

LangChainの主な機能：Chain構築、検索付き応答（RAG：Retrieval-Augmented Generation）構築、Vector Store連携

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.embeddings import OpenAIEmbeddings

# テキストを読み込み・分割

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=500, chunk\_overlap=50)

docs = text\_splitter.create\_documents([your\_raw\_text])

# Embedding

embedding = OpenAIEmbeddings(model="text-embedding-ada-002") # コスト効率が高い

1. 次に、**類似するベクトル**（特徴ベクトルvector）を高速・高効率で検索するためにFAISS（ファイス）で検索インデックスを作成します。

from langchain.vectorstores import FAISS

# ベクトル検索用インデックスを作成

faiss\_index = FAISS.from\_documents(docs, embedding)

# 保存も可能

faiss\_index.save\_local("faiss\_medical\_index")

1. LangChainで先ほど作成したFAISS indexを使ってRAG（検索付き応答）を構築いたします。

 ユーザーが「頭痛に効く薬は？」と入力

 その質問をベクトルに変換

 FAISSの検索インデックスと照合して、**類似質問や文書をすばやく取得**

 取得した情報を元に回答を生成（RAG構成）

from langchain.chains import RetrievalQA

from langchain.chat\_models import ChatOpenAI

# GPT-4 Turboを使用

llm = ChatOpenAI(model\_name="gpt-4-turbo", temperature=0.2)

# インデックスから検索 → GPTで応答

qa\_chain = RetrievalQA.from\_chain\_type(

llm=llm,

retriever=faiss\_index.as\_retriever(search\_kwargs={"k": 3}),

return\_source\_documents=True

)

# 実行例

response = qa\_chain.run("歯の神経を抜いた後の注意点は？")

print(response)

1. FastAPI/Reactを使ってフロントエンド接続（せつぞく）し、

LangChainバックエンドをAPI化（FastAPI推奨）

チャットUIを作成（StreamlitやReactなど）

患者からの質問をAPI経由で処理

### RASA

この歯科医療サービスシステムにRASAフレイムワクを活用いたしました。このRASAフレイムワクは入力する質問を分類し、intent検出（けんしゅつ）をすることで、適切なモデルを利用されるようにします。それで、RASAフレイムワクを利用することは大規模のAI業務の前（ぜん）処理として考えことができます。

この歯科医療サービスシステムおいて、もし、基本質問と関係のない内容を入力すれば、RASAフレイムワクでのアクションガ例外（れいがい）処理を行います。

このRASAフレイムワクを活用するために、特定のプロトコルに従って（したがって）プロセスを実行する必要があります。

まず、config.yml ファイルのpipelineにおいて、diet classifierや mistral modelなど、基本的なmodelを設定いたします。

次に、domain.ymlファイルにおいて、入力する質問を分類することで利用されたintentsを定義いたします。

その後でnlu.ymlファイルにおいて、先ほど作成したintentと関係される例文（れいぶん）を作成いたします。

最後にdomain.ymlファイルにおいて、入力する質問に応える（こたえる）アクション（action）を定義いたします。それにより、RASAフレイムワクで入力する質問に応じて、適切な応答をすることができます。

### **音声合成（TTS）および音声認識（STT）のAPI活用経験**

自動カルテ生成STT

日本語対応の音声インターフェースを備えたサービスシステムを開発するため、TTS（Text to Speech）および STT（Speech to Text）技術を積極的に検証（けんしょう）・実装しました。  
具体的には、TTS には **ElevenLabs（イレブンラボズ） API、GPT-4o API、Azure（アジュール） TTS** を使用し、それぞれの音声自然性（しぜんせい）・応答速度・カスタマイズ性を比較（ひかく）しました。中でも **ElevenLabs API** は、日本語の発音（はつおん）精度と感情表現（かんじょうひょうげん）が最も優（すぐ）れており、ナレーション（narration）や対話(たいわ)アプリケーション向けに最適であると・判断（はんだん）しました。

一方、STT には **OpenAI の Whisper（ウィスパー） API** を使用し、高精度（こうせいど）な音声認識が必要な対話型(たいわ)アプリケーションや文字（もじ）起こし（おこし）機能において非常に高い認識精度とノイズ（noise）耐性(たいせい)を確認しました。特に日本語話者（わしゃ）特有（[とくゆう）のイントネーションや文脈（ぶんみゃく：context）の認識にも対応しており、**リアルタイム認識や長文音声の変換（へんかん）において実用的な性能**を有しています。

これらの技術を用いて、ユーザーとの音声インタラクション(interaction)を実現するプロトタイプを構築した経験があります。今後は、音声合成と音声認識を組み合わせたインテリジェントな会話システムの構築に貢献できると確信しております。

# **API design and implementation(仕入れ管理)**

## English

In one of my recent projects, I built a purchase management system for an e-commerce site used by a mid-sized bank that was selling to overseas markets. I was responsible for the backend API. Since the system was API-driven, it was really important to keep the data flow smooth and make the system easy to maintain. I started by deciding **what types of HTTP** requests we’d need for handling products—like using GET to fetch product info, POST to add new products, PUT to update them, and DELETE to remove them. Then, I organized the **API routes** following RESTful design, using endpoints like /api/purchases and /api/purchases/:id. This made it easy for both frontend and backend teams to understand and use. After that, I defined **the input and output** formats, based on the business needs. For example, we used fields like productId, quantity, and purchaseDate. And for responses, we returned status codes, confirmation messages, and saved data. To make sure all data was valid, I added middleware for validation and set up reusable rules using JSON schema. This helped block any wrong or broken data from getting into the system. Then I worked on the core API logic—things like checking inventory, verifying suppliers, and calculating prices. I kept this logic separate from the routing code so the system would be easier to manage and update later.

## Japanese

国外市場向けのe-commerceサイトの仕入れ管理システムにおいて、私はバックエンドAPIの設計および実装を担当いたました。プロジェクトでは、API駆動型アーキテクチャを採用しており、スムーズなデータ連携と高い保守性を実現する/必要がありました。E-コマースサイトの仕入れ管理システムにおいて、

1. （**リクエストタイプ**定義）まず各処理に適した**HTTPメソッド**を定義しました。たとえば、**仕入れ履歴の取得には**GET、新規仕入れの登録にはPOST、誤（ご：incorrect, mis）登録の削除[さくじょ]にはDELETE(ディリート)を使用しました。
2. （API**ルーティング**定義）次に、**RESTful**の原則に基づいてAPI**ルーティング** (エンドポイント)を設計しました。たとえば、仕入れ履歴を表示する場合は/api/purchasesを、 特定の仕入れ情報を取得、修正、削除する場合は/api/purchases/:id といった形でルーティングを整理し、それで、フロントエンド・バックエンド両方がわかりやすくなるように整理(せいり)しました。purchases/:idー>パーチェシズ スラッシュ コロン アイディ
3. （**入力/出力パラメータ**定義）そのあとは、業務要件に基づいて、APIの**入力と出力**のルールを厳密[げんみつ]に設計しました。たとえば、入力では 商品Id や 数量（すうりょうquantity）、購入日（こうにゅびpurchaseDate） などを受け取って、出力データにはステータスコードや確認メッセージ、登録された内容を含めるようにしました。
4. （**バリデーションチェック**）それから、間違ったデータが登録されないように、**ミドルウェア**を使ってサーバー側で**バリデーション**処理を行いました。また、JSONスキーマで再利用可能な検証 [けんしょう]ルールを定義し、データの整合性を確保しました。
5. （APIロジック構築）：在庫チェックや、仕入先の確認、価格（かかく）の計算などの**ビジネスロジック部分**を実装いたしました。このロジック部分をルーティング部分から分けて、**保守性**（ほしゅせい）を高めました。

**その結果**、APIは安定性・拡張性・ドキュメント性に優れ（すぐれ）たものと・なりました。そのため、クライアントとサーバー間のエラーが大幅に減少（げんしょう）し、チーム全体の開発効率が向上しました。

## Reusable Json schema for purchase management system.

{

"$**schema**": "https://json-schema.org/draft/2020-12/schema",

"$**id**": "https://example.com/schemas/purchase-order.json",

"**type**": "object",

"**properties**": {

"orderId": { "type": "string" },

"orderDate": { "type": "string", "format": "date-time" },

"customer": { "$ref": "#/$defs/customer" },

"items": {

"type": "array",

"items": { "$ref": "#/$defs/orderItem" }

},

"**totalAmount**": { "type": "number", "minimum": 0 }

},

"required": ["orderId", "orderDate", "customer", "items", "totalAmount"],

**"$defs**": {

"customer": {

"type": "object",

"properties": {

"customerId": { "type": "string" },

"name": { "type": "string" },

"email": { "type": "string", "format": "email" }

},

"required": ["customerId", "name", "email"]

},

"orderItem": {

"type": "object",

"properties": {

"product": { "$ref": "#/$defs/product" },

"quantity": { "type": "integer", "minimum": 1 },

"unitPrice": { "type": "number", "minimum": 0 }

},

"required": ["product", "quantity", "unitPrice"]

},

"**product**": {

"type": "object",

"properties": {

"productId": { "type": "string" },

"name": { "type": "string" },

"category": { "type": "string" }

},

"required": ["productId", "name", "category"]

}

}

}

# E-commerceサイトの制作において、特に重要なことは何ですか？

## English

“Throughout my career, I’ve had the opportunity to design and develop multiple e-commerce platforms, but one project that really stood out for me was Drape Fit — a personalized fashion subscription service. In that project, I came to understand that building a successful e-commerce platform requires mastering three essential areas: UI/UX design, security, and payment management.

1️⃣ User-Centered and Fashion-Aware UI Design

In any e-commerce service, the first impression comes from the UI. I always approach interface design by asking myself: Is this layout attractive? Does it match the user's taste?

For Drape Fit, I took special care to follow design trends and make data-informed decisions. A few years ago, white and minimalistic themes were dominant. But recently, I've seen a shift toward dark themes, especially for fashion-related platforms. To adapt to user preferences, I designed components that could easily switch between light and dark themes using CSS variables and global state management.

I also prioritized mobile-first responsive design, since the majority of our traffic came from mobile devices. The goal was to reduce friction and make shopping intuitive, even on small screens.

2️⃣ Security: Protecting Data and Access

Security is non-ne**go**tiable in e-commerce. I focused on both authentication and authorization.

Authentication: I implemented JWT-based token authentication for stateless API communication and added OAuth2 integration for third-party login options like Google and Facebook, improving convenience without sacrificing security.

Authorization: I applied RBAC (Role-Based Access Control) to define permissions for admins, stylists, and customers. This prevented unauthorized access to sensitive features, such as order management and payment records.

I also enforced HTTPS-only communication, protected API routes, and sanitized all inputs to avoid SQL injection or XSS attacks.

3️⃣ Stripe-Based Secure and Seamless Payment System

For code

But perhaps the most critical part of any e-commerce platform is the payment system.

For Drape Fit, I implemented a full Stripe integration from scratch, ensuring the experience was both secure and frictionless for users. Here’s how I structured it:

🧾 Implementation Steps:

Customer Registration in Stripe

At the time of user signup, I created a corresponding Stripe customer object and stored the Stripe customer ID securely in our database.

Secure Card Registration

I issued a setupIntent from the backend and used Stripe.js on the frontend to tokenize the card data. Our server never saw raw card numbers, ensuring PCI DSS compliance.

Saving and Selecting Card Info

Once a card was registered, I fetched the metadata (last 4 digits, expiration, brand) from Stripe and saved it in a custom DB table. This allowed users to reuse cards for future payments easily.

Creating a PaymentIntent

For styling fees and product purchases, I created a paymentIntent with the "manual" confirmation method to support 3D Secure.

Handling Payment Confirmation and Success

After successful payment, I recorded the transaction in payment, payment\_gateway, and updated the product checkout status automatically.

Webhook Integration for Event Tracking

I implemented a Stripe webhook listener to track events like checkout.session.completed and log them for later audits or debugging.

🚀 Final Words

Through this project, I learned how to align user expectations, security protocols, and financial systems in one seamless e-commerce experience. It sharpened my skills not only as a developer but also as a designer and strategist. I’m confident these experiences will directly contribute to any team that values robust, scalable, and user-focused e-commerce solutions.

## Japan０

これまでいくつかのECサイトを設計・構築してきましたが、その中でも特に印象(いんしょう)に残（のこ）っているのが、Drape Fitというパーソナルスタイリング(personal styling)サービスの開発です。

Drape Fitは、ユーザーの好み（このみ）に合わせて/スタイリストが服（ふく）を選び、自宅に届（とど）けるパーソナルスタイリングの定期購入（ていきこうにゅう）サービスです。

この経験から、ECサイトの成功（せいこう）には、**UI/UX設計・セキュリティ対策・決済基盤（けっさいきばん）の構築**の3つの要素が最も重要だと考えています。

1️⃣ **ユーザーの好みに寄り添ったファッショナブルなUI設計**

ECにおいて最初にユーザーの心（こころ）を掴む（つかむ）のはUIです。私は「この画面は今のトレンドに合っているか？ユーザーが使いたくなるか？」を常に意識して設計しています。たとえば、数年前まではホワイトテーマが主流([しゅりゅう])でしたが、最近はブラックテーマの人気が高まっていることに着目（ちゃくもく：attention）し、**ダークモード(dark mode)対応UI**をCSS変数と状態管理で柔軟に切り替えられるようにしました。また、モバイル利用が多かったため、**モバイルファーストの設計方針(ほうしん)を採用し、レスポンシブ対応**を徹底（てってい）することで、どんなデバイス環境でも快適（かいてき）に操作、閲覧（えつらん）できるようにしました。

2️⃣ **セキュリティの徹底：認証と認可の実装**

セキュリティは、ECサイトの信頼性そのもの/と言えます。セキュリティには大きく分けて2つの要素があって、1つは「認証（にんしょう）」、もう1つは「認可（にんか）」です。

認証機能として、JWT（JSON Web Token）を利用したトークンベース認証や、OAuthプロトコル（protocol）を用いたSNSログイン機能を実装し、ユーザーにとって安全かつ便利な認証フローを実現しました。

認可機能として、RBAC（ロールベースアクセス制御）を導入し、管理者・スタイリスト・顧客などの各ロールに応じてアクセスできる機能を制限(せいげん)することで、適切な権限管理を実現しました。

3️⃣ 安心と使いやすさを両立した決済システム

ECの心臓部（しんぞうぶ）とも/いえるのが/「決済機能」です。

Drape Fitでは、最新の決済プラットフォーム（Stripe）を使い、完全(かんぜん)にセキュアでスムーズな決済システムをゼロから設計・構築しました。Stripeは、カード登録や認証、決済処理などを安全に行える最新の決済プラットフォームです。トークン化、SetupIntent、3Dセキュアなどの高度（こうど）な機能により、PCI準拠（じゅんきょ）でユーザーフレンドリーな決済体験を提供します。私はStripeを使って、カード情報をサーバーに通さない安全で再利用可能な決済フローを構築しました。

**技術的な流れ(ながれ)を簡潔（かんけつ）に説明すると：**

**１．Stripeの顧客アカウント作成**

ユーザー登録と同時に、決済プラットフォーム上に**ユーザー専用(せんよう)のアカウント**を作成。

2. カード登録（setupIntent）

クレジットカード登録時は、カード情報が**弊社サーバーを一切(いっさい)通（とお）らない仕組み**で、業界基準（きじゅん）に準拠（じゅんきょ basis）した方法で安全に登録。具体的に、フロントエンドではStripe.jsを使ってカード情報をトークン化し、サーバー側では一切（いっさい）カード番号を扱わない構成にすることで、セキュリティを確保しています。

３．カード情報の保存と再利用

登録されたブランド名や下4桁（したよんけた）、有効期限（きげん）などの情報を取得し、ユーザーが支払い時（じ）にカードを選択できるようにしました。

４．決済Intentの作成（3Dセキュア対応）

決済時には本人確認機能付き（つき）の安全な決済プロセスを導入し、ユーザーにも安心して支払ってもらえる体験を提供。具体的に、決済時には、まずユーザーの支払い情報をStripeに送信し、PaymentIntentというオブジェクトを生成します。これは「これから支払いをします」と伝えるしくみで、3Dセキュアが使えるカードなら、SMSやワンタイムパスワード(one-time password)などで本人確認が必要になります。

3Dセキュア：カード発行会社＋加盟店（かめいてん）＋インフラ提供者（Visa, Mastercard）

５．支払い成功後の処理

支払いが完了したら、**取引（とりひき）記録や注文(ちゅうもん)ステータスを自動で更新**します。

６．Webhook連携でイベント管理

決済イベントの監視（かんし）には**自動ログシステム**を活用し、トラブル時にも迅速に対応できる体制を整えました。

このプロジェクトを通して、私は「**ユーザー体験・セキュリティ・ビジネス要件**」の3つをバランスよく設計・実装する力を身につけました。単なる(たんなる)技術者ではなく、サービス全体を見渡（みわ）せる開発者として、次のプロジェクトにも大きな価値（かち）を提供できると確信しています。

# AWSの活用経験0

はい、AWSの活用経験がございます。特に、Fora Soft社で携わった**遠隔医療プラットフォーム**のプロジェクトでは、**インフラ設計**から**運用**まで一貫してAWSを活用いたしました。このシステムでは、患者さんごとにカスタマイズされた薬剤検索を行うAIエンジンを備え(そなえ)ており、そのために**高いスケーラビリティと可用性、**そして**セキュリティ**が必要でした。

まず、バックエンド**はPythonとDjangoで**構築しており、EC2（ツ）上にアプリケーションを**デプロイ**いたしました。Amazon EC2（イーシーツー）は、Amazon Web Services（AWS）が提供するクラウド上の仮想（かそう）サーバーサービスです。

その上(うえ)で、クライアントサイドとの通信（つうしん）には/**API Gateway**を使用し、**CORS（コース）**対応や**スロットリング**（throttling：）などの設定を行い、**セキュア**で安定したAPI基盤(きばん)を構築いたしました。CORS（コース）はCross-Origin Resource Sharingの略（りゃく）で、異なる（こどなる）ドメイン間で安全にデータをやり取りできるようにするWebの仕組みです。APIサーバーを外部サービスとして使うときに必要です。Throttlingを使ってアクセス回数(かいすう)や速度を制御（せいぎょ）できます。

さらに、一部の非同期処理やシンプルなデータ処理については**AWS LambdaをPythonで**実装し、**スケーラビリティ**とコスト効率を向上させました。（**AWS Lambda？**Amazon Web Services（AWS）が提供している サーバーレスコンピューティングサービス です。プログラムコードだけを用意（ようい）すれば、**サーバーの準備や管理なしで、自動的に実行できる仕組み**です。）

また、AI検索用のインデックスや履歴データの高速検索には**DynamoDBやChromaDB**を利用し、フロントエンドの検索スピードを大幅に向上させることができました。

**ログデータ**や静的（せいてき）ファイルは**S3（スリー）に保存**し、バージョン管理(かんり)とアクセス制御も適切に行っています。S3: Amazon Web Services（AWS）が提供する**クラウド上の大容量（だいようりょう）ストレージサービス**

**成果としては**、検索APIのレスポンスタイムを約33%短縮し、サーバーレス構成によって月間（げっかん）のインフラコストを15%削減（さくげん）することができました。さらに、CloudWatch（クラウドウォッチ）やワフ（WAF：web application Firewall）などを活用することで、安定稼働率99.99%を達成（たっせい）しています。CloudWatch（クラウドウォッチ）とは、**Amazon Web Services（AWS）** が提供する **監視サービス（モニタリングサービス）** です。WAF（ワフ）とはWeb Application Firewall（ウェブアプリケーションファイアウォール）\*\*の略です。WAFは、Webアプリを守る「防御壁（ぼうぎょうかべ）」です。悪いアクセスが来てもブロック（block）して、安全にサービスを提供できるようにしてくれます。

AWSのさまざまなサービスをプロジェクト全体で連携・最適化する中で、実際の運用に耐える（たえる）医療システムの構築に貢献できたと自負しております。

RDSはAmazon Web Services（AWS）が提供するrelational database**サービスとして、**databaseの設定や、操作、スケールを簡単にできます。

**AWS LambdaをPythonで**実装

Step 1: Write Your Python Lambda Function

Step 2: Create the Lambda Function in AWS Console

1. Log in to the [AWS Management Console](https://console.aws.amazon.com/)
2. Go to Lambda

Step 3: Set the Trigger (When It Runs) トリガーを設定する（実行タイミング）

Step 4: Deploy Your Code

Step 5: Give Proper Permissions

# CI/CDワークフロ０

これまでに携わった複数のプロジェクトにおいて、CI/CDパイプラインを構築・運用してまいりました。CI/CDとは、継続（けいぞく）的インテグレーション（Continuous Integration）および継続的デプロイメント（Continuous Deployment）を表示いたします。  
私は、以下の手順でGitHub Actionsを活用してCI/CDパイプラインを構築しました。

1. まず、AWS EC2インスタンス上にデプロイ専用（せんよう）ユーザー（deployment user）を作成します。このユーザーには、SSHアクセス権限（けんげん）およびデプロイに必要なパーミッション(pamission)を付与（ふよ）します。
2. 次に、GitHubリポジトリ上のSecrets（シークレッツ）に、EC2インスタンスのホスト名（Host）、ユーザー名（Username）、SSH秘密鍵（ひみつキー Private Key）などの情報を安全に登録します。これにより、GitHub Actionsから安全にEC2へアクセスできるようになります。
3. その後、リポジトリのルートに .具体的にgithub/workflowsにおいて、 フォルダ（folder）を作成し、ここにCI/CD用のワークフローファイル（YAML形式「けいしき」）を配置(はいち)します。
4. ワークフローファイル内では、例えば push（プッシュ） イベントをトリガー（trigger）として、テスト実行・ビルド・EC2へのSSH接続（せつぞく）・デプロイスクリプト(deploy script)の実行といった一連の処理を定義します。

このCI/CDパイプラインを構築することで、開発環境から本番環境へのリリース作業を自動化でき、デプロイのスピードと正確性を向上させることができます。また、手動作業を削減(さくげん)することで、人為的(じんいてき)なミスを防ぎ（ふせぎ）、安定した本番デプロイが実現可能になります。また、チーム全体で共通のフローを使えるため、開発プロセスの透明性や一貫性も高まります。

一方で、注意（ちゅうい）点としては、初期構築に少し手間（てま:labor）がかかることや、セキュリティ管理（例えばSSH鍵やGitHub Secrets）をしっかり行う必要があることです。また、複雑なワークフローになると、トラブル発生時（じ）の調査に少し時間がかかることもあります。

それでも、CI/CDを導入することで、最終的には開発・運用の効率と品質を大きく向上させることができると考えております。

# Docker experience

English: Can you tell me about your experience using Docker?

Certainly. I used Docker extensively in a client project called **"Health Care Plus"**, where I was responsible for backend development and environment setup. To streamline the development process and ensure a consistent and portable environment, I created a customized **Docker Compose file** for the backend.

The Compose file defined multiple settings, such as container names, volume mappings, and Node.js package dependencies. This allowed all team members to spin up the same environment with minimal setup, which significantly reduced onboarding time and eliminated "it works on my machine" issues.

Compared to traditional virtualization tools like VMware, Docker offered a much lighter and more efficient solution, as it shares the host OS kernel and optimizes resource usage. This reduced overhead was especially helpful in maintaining fast iteration cycles during development.

After completing the development and improvements, I built a Docker image that included all necessary configurations and delivered it to the client. This ensured a smooth deployment process without configuration drift. Through this project, I deepened my practical knowledge of containerization and now actively use Docker to enhance DevOps practices, development speed, and infrastructure consistency.

Japanese: Dockerの使用経験について教えてください。

はい。クライアントのプロジェクト「Health Care Plus」にて、バックエンドの開発と環境構築を担当した際に、Dockerを活用しました。開発環境の標準化[ひょうじゅんか]と効率化を目的[もくてき]として、バックエンド用にカスタマイズした Docker Compose「ドッカー・コンポーズ」ファイル を作成しました。このComposeファイルには、コンテナ名、ボリューム設定、Node.jsのパッケージ依存関係など、システム構成に必要な情報を詳細に定義しました。その結果、チームメンバー全員が同一の環境を簡単に構築できるようになり、初期[しょき]設定の手間[てま]を大幅に削減[さくげん]できました。また、「自分の環境では動く」といった問題も防ぐことができました。VMwareなどの従来の仮想化ソリューション（solution）と比較しても、DockerはホストOSのカーネル(kernel)を共有するため、リソースの使用効率が高く、負荷が少ないのが大きな利点です。この軽量性(けいりょうせい)により、開発スピードを維持[いじ]しながら作業を進めることができました。最終的には、必要な設定をすべて含んだDockerイメージを作成し、クライアントに納品[のうひん]しました。これにより、再現性の高いスムーズなデプロイが可能になりました。この経験を通じて、Dockerを活用した環境構築やDevOps（デブオプス）への応用に対する理解がより深まり、現在では日常的にコンテナ技術を活用しています。

# **(PostgreSQL：trouble )過去に技術的なトラブルやバグをどのように解決しましたか？**

For a drug search system, I initially implemented a full-text search algorithm. However, the system's performance was poor due to an inadequate database indexing design. To improve performance, I added a column of type **tsvector** to the drug table and stored **combined, tokenized data** for full-text search using SQL functions like **to\_tsvector**. Next, I created a **GIN** (Generalized Inverted Index) on the tsvector column to enable **fast searching**. Without a GIN index, PostgreSQL would have to scan every row, which is inefficient for large tables.

Finally, I performed the full-text search using SQL commands such as to\_tsquery.

As a result, the search speed improved by approximately 50%.

**回答例:**  
以前、薬剤検索システムを開発したときに、最初は全文検索のアルゴリズムを/自分で実装したんですが、データベースのインデックス設計がうまくできていなくて、パフォーマンスがすごく悪く（わるく）なるという問題が起（お）きました。そこで、まず薬剤テーブルに、tsvector型の新しいカラム（cloumn）を追加しました。その後、このカラムでto\_tsvector などの SQL 関数を使って全文検索用に**結合およびトークン化されたデータ**を生成しました。次に、SQLクエリを使ってそのカラムにGIN index（Generalized Inverted Index）を作成し、全文検索を高速化できるようにしました。GINインデックスがない場合、PostgreSQLはすべての行（ぎょう）をスキャン（scan）する必要があり、大規模なテーブルでは非常に非効率だからです。最終的に、to\_tsquery(クエリ)などのPostgreSQL検索クエリを使って全文検索を実行し、検索速度を約50％向上させることができました。この経験を通じて、適切なインデックス設計を行うことが、システムのパフォーマンスに直結(ちょっけつ)することを学びました。それ以来、開発時にはデータベース設計にも十分に配慮(はいりょ)し、常にパフォーマンスを意識した実装を/心がけております。」

| **Step** | **Action** | **Description** |
| --- | --- | --- |
| 1 | ALTER TABLE ... ADD COLUMN search\_vector tsvector; | Create the column |
| 2 | UPDATE drugs SET search\_vector = ... | Fill it with combined tokenized data |
| 3 | CREATE INDEX ... USING GIN (search\_vector); | Index it for fast search |
| 4 | SELECT \* FROM drugs WHERE search\_vector @@ to\_tsquery('pain & fever'); | Run full-text search queries |

# Next.js

私はこれまで3年間、Next.jsを使ったWeb開発に携わってきました。主に、SSR（サーバーサイドレンダリング）やSSG（静的サイト生成）、APIルート、画像最適化など、Next.jsの主要(しゅよう)機能を活かした実装を/行ってきました。特にSEO対策（たいさく）や表示速度の最適化に効果的なSSRやISRを使い分けて、ユーザー体験を向上させるよう努め（つとめて）てまいりました。

## **❓ 具体的な開発経験を教えてください**

あるECサイトの開発で、Next.jsをフルスタックに活用しました。商品詳細（しょうさい）ページにはSSRを、キャンペーンページ（campaign page）などにはSSGを、ユーザーの注文(ちゅうもん)履歴ページにはCSRを使用しました。また、APIルートを使って、Stripe（ストリップ）と連携した決済（けっさい）処理を実装したり、NextAuth.jsでログイン認証機能を構築した経験もあります。」

## ❓ Next.jsとは何ですか？なぜ使いますか？

Next.jsはReactをベースにしたフレームワークで、サーバーサイドレンダリング（SSR）や静的サイト生成（SSG）、APIルートなどの機能を備え(そなえ)ています。  
SEO対策（たいさく）がしやすく、パフォーマンスの高いWebアプリケーションを効率よく開発できる点が魅力で、特に商用(しょうよう)サービスやプロダクト開発において非常に有効（ゆうこう）だと感じています。

## **❓ ISR（Incremental Static Regeneration）とは？**

ISRは、Next.jsの静的生成の拡張機能で、デプロイ後でも一定の時間間隔（かんかく）でページを自動的に再生成できます。これにより、リアルタイム性のあるデータを扱い（あつかい）ながらも、パフォーマンスの高い静的配信を維持（いじ）できます。私のプロジェクトでは、商品一覧ページなどにISRを使って、在庫数や価格の変化(へんか)を反映させるようにしました。

## ❓ Next.jsのAPIルートはどのように使いますか？

「Next.jsのAPIルートを使えば、バックエンド処理をフロントエンドと同じプロジェクト内で簡単に実装できます。例えば、フォームの送信処理、ログイン認証、外部APIとの連携などに使いました。別途（べっと）サーバーを用意する必要がないので、開発効率も高いと感じています。」

## **❓ SSR・SSG・CSRの違いと使い分けは？**

「SSRはリクエストごとにサーバーでページを生成する方式（ほうしき）で、動的なページやログインユーザー向けのページに適しています。  
SSGはビルド時にHTMLを生成するので表示が非常に高速で、ブログ記事(きじ)（blog post）や製品（せいひん）紹介など、あまり変化（へんか）のないページに使います。  
CSRはクライアント側でレンダリングを行うため、ページ遷移（せんい）の滑（なめ）らかさを重視する場合や、一部のコンポーネントだけ更新したい場合に使います。

JavaScriptのライブラリであるReactは、典型的（てんけいてき）なCSR（クライアントサイドレンダリング）を採用しています。

私はそれぞれのケースに応じて、最適なレンダリング手法（しゅほう）を選択しています。」

# Ruby on Rails

## English

In one of my recent projects, called **"KagoAruoo JP"**, I was responsible for backend development using **Ruby on Rails**. The project was a real estate service system designed to recommend suitable apartments based on factors like size, price, furniture availability, and other preferences.

I leveraged **Rails’ MVC architecture** to build clean, maintainable code. For example, I used **ActiveRecord** extensively to manage the database layer, allowing me to query and update data using Ruby syntax without writing raw SQL.

I also set up the **routing system** to follow RESTful principles, which helped keep the controller logic organized and predictable. To speed up development, I utilized **scaffolding** features to quickly generate the CRUD interface for product listings and user management.

To resolve this, I implemented **eager loading using .includes()** in the query logic, which significantly reduced the number of database calls. As a result, **the introduction listing page load time improved by approximately 40%**.

This project deepened my understanding of Rails internals and best practices, and I’m confident in applying these techniques in larger, more complex systems.

## Japanese

最近携わったプロジェクトの一つに、\*\*「KagoAruoo JP」\*\*というシステムがあります。こちらの開発では、**Ruby on Rails** を用いてバックエンドを担当しました。このシステムは、広さ [ひろさ]・価格 [かかく]・家具 [かぐ]の有無 [うむ]など、様々な条件に基づいて最適な賃貸物件[ちんたいぶっけん]を推薦[すいせん]する機能を持っています。

**RailsのMVCアーキテクチャ**を活用し、可読性が高く、保守性の良いコードを意識して開発しました。データベース処理には **ActiveRecord** を使い、SQLを直接書かずにRuby構文で柔軟に操作できる点が非常に効率的でした。また、ルーティング設計には RESTfulなアプローチを採用し、コントローラーの構成を整理しやすくしました。さらに、スキャフォールディング機能を活用して、商品リストやユーザー管理のCRUD機能を素早く実装しました。

開発中には、**大量の紹介データを表示する際のページ表示速度の低下**という課題に直面しました。調査の結果、ActiveRecord における **N+1クエリ**が原因であることが判明しました。これを解決するために、\*\*.includes() を使った eager loading（先読み読み込み）\*\*を導入し、データベースへのアクセス回数を大幅に削減しました。その結果、**一覧ページの表示速度が約40%向上**しました。

このプロジェクトを通じて、Railsの内部構造やベストプラクティスに対する理解がさらに深まり、大規模かつ複雑なシステムでも応用できる自信があります。

# React Native and GraphQL

FrogQuest is a mobile-first social scavenger hunt app that uses React Native and Expo for a smooth cross-platform experience. We capture photos through **Expo’s camera module**, organize challenges and scoring through a GraphQL backend, and support real-time collaboration and competitive play among teams. It’s designed to make team building more dynamic and engaging.

**What is a Photo Scavenger Hunt?**

It’s like a **game** where people or teams try to **find and take photos** of specific things or complete fun tasks using a camera

Imagine you're given a list like this:

* ✅ Take a photo of your team doing a funny dance.
* ✅ Take a photo with something red and round.
* ✅ Take a photo of the team making a human pyramid.

Each time you complete one, you take a photo and submit it. You get **points** for each task!

**GraphQL** is a query language that supports a modern way to communicate between your **frontend** (like your React Native app) and your **backend/server**. Instead of using traditional REST APIs (like /users, /photos, /events), GraphQL lets you ask for exactly what you need — no more, no less.

Where You Use GraphQL in FrogQuest

Getting challenge list

→ getChallenges(eventId: ID)

Uploading a photo

→ submitPhoto(challengeId, photoUrl)

Getting scores

→ getLeaderboard(eventId)

Real-time updates (optional)

→ photoSubmitted (as a subscription)

GraphQL

# Vercel

Vercel（ヴァーセル）は、**Next.jsの開発元(もと)が提供しているクラウドプラットフォーム**で、Next.jsアプリケーションを簡単かつ高速にデプロイ（公開）できるサービスです。  
Next.jsとの親和性（しんわせい）が非常に高く、**SSR（サーバーサイドレンダリング）やSSG（静的サイト生成）、ISR（増分（ぞうぶん）静的再生成）などの機能を最大限（さいだいげん）に活かせる環境**が整っています。

**主な特徴（とくちょう）：**

* **Git連携による自動デプロイ**（GitHubやGitLabと連携して、プッシュするだけで公開）
* **グローバルCDNによる高速配信**
* **APIルートやサーバーレス関数のサポート**
* **プレビュー環境の自動生成**（開発中のブランチごとにURLを発行）
* **画像やリソースの最適化機能**

# CMS経験

2023年からフリーランスとして活動しており、副業（ふくぎょう）として、WordPress（ワードプレス）などのCMSを利用して日本市場向けのWebサイトを構築した経験があります。CMSを用いたシステム構築では、ユーザビリティ、拡張性・柔軟性、セキュリティ、SEO対応などを重視しております。また、ソフトウェアエンジニアとしては、フロントエンドおよびバックエンド機能のカスタマイズを行うことが基本作業となります。CMSにおいては、提供されているプラグインの活用が非常に重要であり、要件に応じて既存（きそん）のプラグインを組み合わせたり、カスタムプラグインを開発することも対応可能です。

# How do you use React hooks and context API?

## English

I used React Hooks like useState() and useEffect() to manage state variables and local logic such as fetching data or updating UI state. Context is for data shared across many components, without passing props down many levels. In order to share global data like login Info or permissions, I used the context API.

First, create context using the hook - createContext().

Second, define the global variables and functions by using context provider component.

Third, place the defined provider component at the top level to wrap around your entire component trees.

Fourth, define a function by using useContext() to access global variables and functions at any child component level using the function.

but there are some disadvantage of context API. for example,when a context value changes, every component that uses useContext(SomeContext will re-render, even if the part of the value they care about didn't change. this leads to unnecessary re-renders and cause performance bottleneck in a large or complex application. In order overcome this problem, you have to split the context into smaller so that only subscribe to the data they care about. and you can use the hook such as React.memo and useContextSelector.

## Japan０

状態変数とローカル ロジックを管理するために、useState() （ユーズ・ステート）や useEffect()（ユーズ・エフェクト） などの React Hooks を使用しています。さらに、useMemo()やuseCallback()を適切に活用することで、パフォーマンスのチューニングを行うことができます。

### useMemo()

useMemo() は、計算コストが高い処理をメモ化して、パフォーマンスを最適化するためのReactのフックです。Eラーニングサービスシステムを開発する際に、「useMemo」を効果的に活用しました。システムには、先生と学生間の質疑（しつぎ）応答をサポートするチャットアプリが搭載（とうさい）されています。たとえば、多数(たすう)のメッセージの中から、入力されたキーワードに一致するものだけを表示するケースがあります。キーワードが変更されていないにもかかわらず、毎回このような処理を再計算してしまうと、メッセージ数が多い場合にはパフォーマンスに悪影響（あくえいきょう）を及ぼす（およぼす：cause）可能性があります。そのため、useMemo() を使って、検索キーワードや元（もと）データが変更(へんこう)されたときだけ再計算されるように制御（せいぎょ）しています。

### useCallback()（ユーズ・コールバック））

In a recent e-commerce project, I implemented a feature that allows users to add products to their cart by clicking an “Add to Cart” button. The structure consisted of a parent component called ProductList, which renders multiple child components called ProductRow. Each ProductRow displays a single product and includes the “Add to Cart” button. In the ProductList component, I defined a function called onAddToCart, which handles adding a selected item to the cart. This function was passed down to each ProductRow as a prop. However, without optimization, this setup would cause every child component to re-render whenever the parent re-renders—even if the props haven’t changed.

To solve this, I wrapped the onAddToCart function in useCallback(). This memoizes the function so that it maintains the same reference between renders unless its dependencies change. This way, unnecessary re-rendering of child components is avoided, leading to better performance and efficiency in the UI.

Additionally, I’d like to point out the difference between useMemo() and useCallback(). useMemo() is used to memoize expensive **computed values**, while useCallback() is specifically used to memoize **functions**. In this case, because I wanted to avoid recreating the same function on every render, useCallback() was the better fit.

最近携わったECサイトの開発では、「カートに追加」ボタンをクリックして商品をカートに追加する機能を実装しました。ページは、商品リストを管理する親(おや)コンポーネント（ProductList）と、各商品を表示する子コンポーネント（ProductRow）で構成されています。子コンポーネントには「カートに追加」ボタンがあり、クリックで商品がカートに追加されます。商品を追加するために親コンポーネント側に 商品追加関数（onAddToCart）を定義し、それを子コンポーネントに props 経由（けいゆ）で渡しています。

しかしそのままでは、プロップスが変更されていない場合でも親コンポーネントが再レンダリングされるたびに、すべての子コンポーネントも再レンダリングされてしまい、パフォーマンスに悪（あく）影響を及ぼす（およぼす：cause）可能性があります。

ここでは、useCallback() フックを使ってこの関数をメモ化（memoize）することで、再レンダリングを防ぐことができます。これにより、子コンポーネントの不要な再レンダリングを防ぎ（ふせぎ）、パフォーマンスを最適化することができました。

また、useMemo() と useCallback() の違いについても触（ふ）れておきたいと思います。  
useMemo() は **計算結果（値）** をメモ化するためのもので、再計算を避ける（きける）際に使われます。一方 useCallback() は **関数** をメモ化するためのフックで、関数が毎回新しく生成されるのを防ぎます。

### Context API

コンテキストは、多くのレベルにプロップスを渡すことなく、複数のコンポーネント間でデータを共有するための仕組みです。たとえば、ログイン情報や権限[けんげん]などのglobal dataを共有するために、context API を使用しました。

そのためには、まず、createContext() を使用してコンテキストを作成いたします。

次に、context provider componentを使って、グローバルに共有した変数と関数を定義いたします。その後、作成したprovider（プロバイダー）componentをアプリケーションの最上位（さいじょういor ルート）レベルに配置（はいち）いたします。

最後に、useContext()フック を使用して、任意（にんい）の子コンポーネントレベルからグローバル変数と関数にアクセスします。これにより、各コンポーネントで必要な値(あたい)や処理を簡単に利用できるようになります。

Context()の欠点

しかし、コンテキスト API には欠点（けってん）もあります。たとえば、コンテキスト値（ち）が変更されると、useContext() を使用するすべてのコンポーネントは、重要な値の部分が変更されていなくても再レンダリングされます。これにより、不要な再レンダリングが発生し、大規模または複雑なアプリケーションでパフォーマンスのボトルネック(bottleneck)が発生します。この問題（もんだい）を解決するためには、コンテキストをより小（ちい）さく分割し、必要なデータのみにサブスクライブ（subscribe）するように/する必要があります。そして、React.memo や useContextSelector「セレクター」 などのフックを活用することができます。

“サブスクライブ（subscribe）するようにする必要があります。”の意味はデータの変化を監視（かんし）し、変化があったときだけ通知（つうち）を受け取るように登録するということです。

React.memo: 関数コンポーネントをメモ化して、子コンポーネントの props が変(か)わらなければ、再レンダリングされません。

useContextSelector：これにより、コンテキストの値(あたい)の一部だけを選択（せんたく）し、その部分が変更されたときにのみ再レンダーされるようにすることができます。

### Reduxを利用

Redux は **アプリ全体の状態（state）を一元（いちげん）管理するためのライブラリ**で、大規模な React アプリケーションなどでよく使われます。最近では **Redux Toolkit（RTK）** が推奨されており、従来（じゅうらい）の Redux よりも **簡単でコード量も少（すく）なく**なっています。

Reduxを利用するためには、まずcreateSlice「クリエイトスライス」関数を使って、状態とその変更ロジック（reducer：リデューサー）をひとまとめするsliceを作成します。

次はconfigureStore「コンフィギュアストア」関数を使って、ストア（store）を作成します。（Redux ストアは、**アプリケーション全体の状態（state）を1つの場所（ばしょ）で管理するオブジェクト**です。React コンポーネントなどから状態を取得したり、状態を変更するために使います）

その後、先ほど作成したストアを<Provider> コンポーネントで React に組み込みます。

最後にアクション（action）を送る(おくる)dispatch（ディスパッチ）関数を使って、コンポーネントで作成させた状態と変更ロジックを利用できます。

# How to implement backend intergration?

## English

1. Firtst, you create API in Django (or Next.js API Routes)

2. Call API from React/Next.js frontend using fetch() or axios

3. Handle responses and display data in your UI

4. (Optional) Add authentication, error handling, and loading indicators

## Japanese

はい、フロントエンドとバックエンドの連携実装において、バックエンドに Next.js を使用する場合は API Routes(ルート) を作成し、Django を使用する場合は Django REST framework などを用いて RESTful API を構築します。

次に、フロントエンドから fetch() （フェッチ）や axios（アクシィオス） を用いて、先ほど作成した API に**リクエスト**を送信します。このとき、非同期処理を使ってデータを取得し、React の状態管理（useState や useEffect）を通じて、UI にデータを表示させます。また、追加処理について、認証（にんしょう）、エラー処理、読み込みインジケーター（indicator）を行います。

RESTful API と Next.js の API Routes の大きな違いは、アーキテクチャとスケーラビリティにあります。

RESTful API（例えば Django REST Framework を使ったもの）は、標準化された設計に基づいており、スケーラブルで分離されたバックエンドの構築に適しています。シリアライザー（serialize）や認証、バリデーションなどの豊富な機能が用意されているため、大規模なシステムにも対応可能です。

一方で、Next.js の API Routes は、フロントエンドプロジェクト内にバックエンドの処理を直接組み込める仕組みで、軽量[けいりょう]なアプリケーションや迅速な開発に向(む)いています。ただし、認証や複雑なバリデーションなどを行う場合には、自分で実装する必要があります。

要（よう）するに、RESTful API は構造化された大規模なシステム向け、API Routes はシンプルな用途(ようと)や統合型アプリケーションに適していると言えます。

**認証**：tokenやsession情報をrequest headerに付与(fuyo)して送信し、バックエンド側でその認証情報を検証(kenshō)します。

エラー処理: backendのresponseをチェックし、errorが発生(hassei)した場合は、ユーザーに適切なエラーメッセージを表示できるようにしました。

# MongoDB利用（E-commerce site）

データ構造（こうぞう）を構築いたしました。

また、MongoDBでは、商品データや在庫情報、注文履歴などを効率的に管理するスキーマ設計を行い、データの正規化（せいきか）とパフォーマンスの最適化に努めました。その結果、スケーラブルでメンテナンス性の高いバックエンド基盤を構築することができ、今後の機能拡張にも柔軟に対応できるようになりました。

# SQL experience of E-learning system.

I developed an e-learning system equipped with features such as mark management, student management, and subject/course management. In particular, I was primarily responsible for the **student management module**, where I designed and implemented an intuitive and responsive **user interface**. For the frontend, I utilized the **Metronic admin dashboard template** along with **JavaScript** to ensure a seamless user experience. On the backend, I used **PHP** for server-side logic and **MySQL** for the relational database to store and manage structured data efficiently.

In fact, effective database design is fundamental to achieving optimal system performance. For the student management system, I chose **MySQL** as the database because the data—such as students, courses, grades, attendance, and marks—is inherently **relational**. I designed the schema by creating multiple normalized tables and established **foreign key constraints** to define clear relationships between entities. This allows users to efficiently retrieve all student-related information through relational queries, ensuring data consistency and maintainability.

私は、成績(せいせき)管理・学生管理・科目（かもく）管理といった機能を備え（そなえ）たeラーニングシステムの開発を担当しました。特に、学生管理モジュールの開発を担当し、直感的で使いやすいユーザーインターフェースを設計・実装しました。フロントエンドには、Metronicの管理テンプレートとJavaScriptを使用し、スムーズな操作性を実現しました。バックエンドにはPHPを用いてサーバーサイド処理を行い、データベースにはMySQLを採用して、構造（こうぞう）化されたデータを効率的に管理・保存しています。

実際のところ、効果的なデータベース設計は、システムの最適なパフォーマンスを実現するための基盤となります。 学生管理システムにおいては、MySQLをデータベースとして採用しました。なぜなら、学生・科目・成績・出席（しゅっせき）・点数といった情報は、リレーショナルな構造を持っているためです。 各種の正規化されたテーブルを作成し、外部キー制約(せいやく)（foreign key constraint）を活用して、テーブル間の明確なリレーション（関連）を定義しました。 これにより、ユーザーは学生に関連するすべての情報を効率的に取得できるようになり、データの整合性と保守性が確保されます。

# スキルシート

## E コマースプロジェクト

本プロジェクトは、商品のオンライン販売を目的としたE-commerceサイトの構築を行ったものです。Laravelをベースとしたバックエンドシステムと、Vue.jsを使用したフロントエンドを連携させ、ユーザーが快適に商品検索・購入・決済まで行えるように設計・開発を行いました。

主な機能としては、商品一覧・詳細ページ、ショッピングカート、注文処理、会員登録・ログイン機能、クレジットカード決済（Stripe連携）、注文履歴管理、在庫管理、管理者用ダッシュボードなどを実装しました。また、SEO対策やレスポンシブデザインにも配慮し、モバイル端末でも快適に利用できるUI/UX設計を行いました。

## 仕入れ管理システム

集中型で構成されていた既存の仕入れ管理システムのスケーラビリティ課題を解決するため、分散型システムアーキテクチャを導入しました。Node.jsおよびMongoDBによるバックエンド構築、Reactを用いたフロントエンド開発、ならびにRBAC（ロールベースアクセス制御）による認証・権限管理を実装。API駆動型アーキテクチャの採用により、モジュール間の疎結合化を実現し、システム全体の柔軟性と保守性を向上させました。また、仕入管理モジュールの設計段階においては、責務分離を意識したモジュール分解を行い、データベース構造の最適化とともに、パフォーマンスを意識した怠惰な荷積み（Lazy Loading）やページネーションの実装を通じて、大量データ処理時の応答速度改善にも貢献しました。

## 遠隔医療プラットフォーム

遠隔医療プラットフォームにおけるWeb検索機能の設計・開発を担当し、特に薬剤の検索精度向上に重点を置いたシステム開発を行いました。TypeScriptを用いたデータ構造分析および検索キーワードの最適化を行い、AIベースの検索アルゴリズムを導入。さらに、Material UIによる直感的なUI設計や、AWS API Gatewayを介したバックエンド連携を実装することで、検索速度とユーザビリティの両面で大きな改善を実現しました。

患者ごとの診療データ（既往歴、アレルギー、処方履歴など）を分析し、それに基づいたAIプロンプトを動的に生成・最適化する仕組みを構築。これにより、医師や患者にとって最適化された検索結果を返す医療特化型のインテリジェント検索体験を提供しました。プロジェクト全体を通じて、“患者ファースト” を軸に据えた開発姿勢を徹底し、医療分野でのAI活用における新たな可能性を開拓しました。

## 中規模銀行向けのサビースシステム

中規模銀行を対象としたWebページおよびモバイルアプリの開発プロジェクトにおいて、要件定義からフロントエンド実装まで幅広く担当しました。顧客のニーズや業務フローに基づき、直感的で操作性の高いユーザーインターフェースを設計。React.jsやVue.jsを活用したモダンなフロントエンド開発と、HTML/CSS/JavaScriptによる堅牢なUI構築を行いました。また、バックエンドとの統合も担当し、安定性とユーザビリティの両立を実現しました。UXリサーチからデザイン、実装まで一貫して関与することで、銀行業務に適したスマートなUIの提供に貢献しました。

# FAQ for interview

## 1.Hardest and proudest achievement in your career

### **最も成果**

これまで、銀行向けサビースシステムや医療系プラットフォーム、ECサイトの制作など、さまざまなWebシステムの開発に携わってきました。特に、Fora Soft社で携わった**遠隔医療**プラットフォームの開発では、**薬剤表示・検索システム**を担当しておりました。そのために、[**AI検索アルゴリズム**](#_Japanese)（キーワード一致検索、部分一致検索、ランキングベースの検索、ファジィ検索）を提案して、最適化を行っておりました。その結果、検索の**正確性**を10%向上させただけでなく、ユーザーの**検索時間**も短縮することができました。これにより、遠隔医療プラットフォームの人気向上にも貢献できたと考えております。

### もとも苦労した点

苦労した点としては、IT管理スペシャリスト会社での**仕入れ管理システム**の**分散化**プロジェクトです。私が担当した仕入れ管理システムは結構規模が大きかったので、しっかり最適化を考える必要がございました。そこで、まずシステムをいくつかのモジュール（見積もり依頼、入荷、支払いなどの）に分割し、分散化を実装いたしました。また、[**API駆動型アーキテクチャ**](#_Japanese_1)を採用することで、効率的なデータ処理を実現いたしました。モノリシックなシステムをAPI駆動型の分散アーキテクチャに移行する時に、**データの整合性**とか**パフォーマンスチューニング**が課題（かだい）だったんだけど、**MongoDB**を適切に設計しました。**遅延読み込み**や**ページネーション**を導入することで、スムーズな移行を実現したり、システムの最適化を実装いたしました。

* モノリシックなシステムとは、フロントエンド、バックエンド、データベースアクセスなどのすべての機能が、ひとつのアプリケーション内で動作(どうさ)する構成（こうせい）のことです。
* データの整合性：データが正しく保たれること（保つ：たもつ）
* 遅延読み込：必要になったときだけデータを読み込む
* ページネーション（データを分けて表示する）

## 2. What was your responsibility on the last project and how did you handle it?

前職では、2023年からIT管理スペシャリスト会社で、システムエンジニアとサブリーダとして働いておりました。主なプロジェクトは、国外市場向けのe-commerceサイトにおける仕入管理システムの開発でした。要件定義、アーキテクチャ設計、実装、テスト、デプロイ、運用まで、一貫して関わってまいりました。

e-commerceサイトのスーブシステム（仕入管理システム以外）

顧客管理（CRM）システム ユーザーの購入（こうにゅ）履歴・問（と）い合わせ履歴を管理

在庫管理システム：商品の在庫数をリアルタイムで管理し、売り切れ防止（うりきれぼうし）

発送・配送管理：注文後の発送ステータス、追跡番号、配送指示などを管理

### **最初のステップは要件定義です。**

**要件定義**では、システムが解決すべき業務上の課題を明確にし、機能要件および非機能要件を整理することが求められます。簡単に言うと、**ユーザー視点**と、**システム視点**の両方から要件を確定していくことが重要です。そのために、ビジネスユーザーと積極的にコミュニケーションを取り、ニーズを明確化しました。例えば、**仕入れリクエスト**の登録・更新・削除機能、**仕入先**（サプライヤー）情報の管理機能（登録・更新・検索）、**商品マスタ**ー管理、**発注書の自動**（じどう）**生成**・発行機能（はっこうきのう）、**入荷予定**管理、**在庫連携**機能、**仕入履歴の検索・閲覧**（えつらん）機能、**多通貨**（たつうか）対応、**権限管理**（けんげんかんり）機能が述べ（のべ）られます。

また、**非機能要件**について、**レスポンスタイム**、**システム稼働率**（かどうりつ）、**セキュリティ対策**（たいさく）、データバックアップ（backup）、拡張性、多言語（たげんご）対応、モバイル対応など考慮する必要があります。

【機能要件】

* 仕入れリクエストの登録・更新・削除機能
* 仕入先（サプライヤー）情報の管理機能（登録・更新・検索）
* 商品マスター管理（SKU、品名、カテゴリ、価格など）
* 発注書の自動生成・発行機能
* 入荷予定管理（予定日、数量、入荷遅延時のアラート通知）
* 在庫連携機能（入荷処理後、自動で在庫数更新）
* 仕入履歴の検索・閲覧機能
* 多通貨対応（為替レート自動更新・表示）
* 権限管理機能（一般ユーザー・管理者などの権限分離）
* 仕入データのエクスポート（export）機能（CSV（comma-separated values）出力など：データベースやシステムの中に保存されているデータ形式（けいしき））

【非機能要件】

* レスポンスタイム：主要画面で3秒以内に応答
* システム稼働率：99.9%以上（年単位）
* セキュリティ対策：通信暗号化（HTTPS）、SQLインジェクション防止、認証・認可強化
* データバックアップ：毎日自動実施、7日間保管
* 拡張性：将来的な仕入先追加や商品ラインアップ増加に耐えられる設計
* 多言語対応：英語・中国語へのインターフェース切り替え機能
* モバイル対応：スマートフォン・タブレットからも利用可能なレスポンシブデザイン

画面仕様は、本システムにおける各画面の基本情報を整理するために必要な仕様です。各画面について、画面ID、画面名、主な要素、バリデーションや表示条件などを記載（きさい）し、設計や実装の指針（ししん）となるよう/構成されています。

要件定義・画面仕様の作成に基ついて、直感的な**ワイヤーフレーム（wireframing）**や**モックアップ(mockup)**を作成しました。色彩 [しきさい]、タイポグラフィ(typography)、アイコン(icon)などの選定にこだわることで、最終的には、見た目の魅力と使いやすさの両立を意識しながら、UIを完成させました。

### **次に、アーキテクチャ設計ステップです。**

**要件定義**が何を作るかを決める段階ですが、**アーキテクチャ設計**がどう作るかを決める段階です。より具体的に言うと、要件定義の段階はビジネス上の目標（もくひょう）を明確にするための工程であり、アーキテクチャ設計の段階はシステムの品質（ひんしつ）を確保するための工程です。それにより、**アーキテクチャ設計**では、まず、**システム構成**を作成し、次にデータベース設計、API設計、システムフロなどを簡単に定義べきます。

**Design Architecture（アーキテクチャ設計）**

**2.1 システム構成（全体イメージ）**

* **フロントエンド**：
  + Webブラウザ対応
  + レスポンシブデザイン（PC/タブレット/スマホ）
  + 開発技術例：React.js または Vue.js
* **バックエンド**：
  + REST APIベース設計
  + 開発技術例：PHP（Laravel）、Node.js（Express）など
  + 業務ロジック（仕入管理、商品管理、発注管理）を実装
* **データベース**：
  + リレーショナルデータベース
  + 技術例：MySQLまたはPostgreSQL
* **認証・認可**：
  + ログイン認証（JWTトークン認証またはセッション認証）
  + 権限管理（管理者・一般ユーザー）
* **インフラ環境**：
  + クラウドサーバ（例：AWS EC2 / Lightsail、Azure、GCP）
  + ストレージ：S3など（CSVファイル出力などにも利用）
  + バックアップ機能（自動スナップショット）
* **セキュリティ**：
  + HTTPS通信必須
  + Web Application Firewall（WAF）設定
  + SQLインジェクション・クロスサイトスクリプティング（XSS）対策

**2.2 データベース設計（簡易版）**

* **Suppliers（仕入先）テーブル**
  + supplier\_id（主キー）
  + supplier\_name
  + contact\_info
  + address
* **Products（商品）テーブル**
  + product\_id（主キー）
  + product\_name
  + SKU
  + category
  + price
  + currency
* **Purchase Orders（発注）テーブル**
  + order\_id（主キー）
  + supplier\_id（外部キー）
  + order\_date
  + delivery\_date
  + status
* **Order Details（発注明細）テーブル**
  + detail\_id（主キー）
  + order\_id（外部キー）
  + product\_id（外部キー）
  + quantity
  + unit\_price
* **Users（ユーザー）テーブル**
  + user\_id（主キー）
  + username
  + password（ハッシュ化保存）
  + role（admin/user）

**2.3 API設計（例）**

* POST /api/orders：発注作成
* GET /api/orders/{id}：発注詳細取得
* PUT /api/orders/{id}：発注更新
* DELETE /api/orders/{id}：発注削除
* GET /api/products：商品一覧取得
* GET /api/suppliers：仕入先一覧取得
* GET /api/exports/orders：仕入履歴CSV出力

**2.4 システムフロー例（発注処理）**

1. ユーザーがログインする
2. 発注画面で商品と数量を選択
3. 発注ボタンを押すと、API経由でバックエンドにリクエスト
4. DBに新しい発注レコードと明細レコードを登録
5. 成功レスポンスを返却し、画面に「発注完了」表示
6. 必要に応じて発注書をPDF/CSV形式で出力可能

### 次のステップ**は実装**です**。**

**実装**では、フロントエンドとバックエンドの両方を担当しました。**フロントエンド**では、JavaScriptとReactを用いて、ユーザー操作（そうさ）に応じたスムーズな画面遷移（せんい）を実現しました。**バックエンド**では、Node.jsとMongoDBを使用し、Mongooseによるスキーマバリデーション、認証（にんしょう：authentication）・認可ミドルウェアの実装も行いました。常にRESTful設計を意識(いしき)し、モジュール性と保守性を重視しました。

### **次にテストフェーズです。**

**テストフェーズ**では、JestとMochaによる**ユニットテスト**、Postman(ポストマン)によるAPI自動テスト、さらにフロントエンドとバックエンドを統合した**エンドツーエンド**（end to end）テストを行いました。

**代表的なユニットテストツール】**

| **言語・環境** | **ツール名** | **説明** |
| --- | --- | --- |
| PHP | PHPUnit | PHP用の最も有名なユニットテストフレームワーク |
| JavaScript (Node.js) | Jest / Mocha / Jasmine | フロントエンド・バックエンドどちらでも広く使われる |
| Python | pytest / unittest | Python標準にもunittestがあり、pytestも人気 |
| Java | JUnit | Javaの世界では標準的な単体テストツール |
| C# (.NET) | MSTest / xUnit / NUnit | .NET環境向けのテストフレームワーク |
| Ruby | RSpec | Ruby用の有名なテストフレームワーク |
| Go | go test（標準装備） | Go言語は標準でテストツールを提供している |

### 次は**デプロイ（deploy）ステップです。**

**デプロイ**では、Docker(ドッカー)によるコンテナ化と、docker-compose(コンポーズ)を用いた環境構築を実施しました。本番(ほんばん)環境にはクラウドVPSを採用し、Nginx(エンジンエックス)でリバースプロキシ(reverse proxy)設定を行い、PM2でNode.jsプロセスを管理しました。GitHub Actionsにより、自動デプロイのパイプラインも構築しました。

**日本企業のデプロイ手順（例）**

① 開発完了後、ローカル環境またはテスト環境（ステージング環境）で十分にテストを実施する。

② テスト合格後、**本番（**production**）サーバーにSSHで安全（あんぜん）にログイン**する。

③ 本番サーバー上で、以下の手順（てじゅん）を手動（しゅどう）または半自動で行う：

* ソースコードをGitなどから最新のものに更新する
* 必要に応じてビルド（build）や環境設定ファイルの更新を行う
* 必要ならアプリケーションのプロセスを停止(ていし)・再起動する（例：systemctl, pm2など使用）

④ デプロイ後、動作確認テストを現地（げんち）またはリモートで行う。

⑤ エラーがなければ、リリース完了の報告をチーム・関係部署に行う。

⑥ 本番監視（かんし）ツール（Zabbix、Datadogなど）でしばらくシステム状態をモニタリング(monitoring)する。

**運用フェーズ**では、**サーバーログ**(server log)やデータベースメトリクス(database metrics)を常時(じょうじ)監視（かんし）し、MongoDBのクエリプラン（query plan）を分析してパフォーマンス改善を行いました。  
障害(しょうがい)発生時には迅速にホットフィックス(hotfixes)対応を行い、また、ユーザーフィードバック(user feedback)を基に在庫一括更新機能などの改善も随時実施しました。

このプロジェクトを通じて、技術力だけでなく、ビジネスニーズに応じた開発遂行能力も大きく向上させることができました。

## 3. What kind of tools did you use for ticket management and communication?

"For ticket management, I have used **Jira, GitHub Issues, and Trello**. In Jira, I managed tasks by creating tickets for bugs, new features, and improvements. I used **Kanban boards** to track the progress, assigning statuses like 'To Do,' 'In Progress,' and 'Done,' and I set priorities, deadlines, and responsible team members for each ticket.

In GitHub Issues, I mainly handled tasks directly linked to the source code. I created issues, linked them to pull requests, added labels like 'bug' and 'feature' and collaborated with the team through comments.

For communication, I mainly used Slack and Microsoft Teams. I used Slack for daily team communication, quick updates, and stand-up meetings. For more formal meetings, documentation sharing, and video calls, I used Microsoft Teams.

Depending on the project, I also used emails for client communication and project management tools like Asana or ClickUp to complement ticket tracking."

"チケット管理には、Jira、GitHub Issues、Trelloを使用しておりました。Jiraでは、バグ修正、新機能追加、改善（かいぜん）タスクなどのチケットを作成し、カンバンボード（Kanban board）を活用して、『To Do（未着手:みちゃくしゅ）』『In Progress（作業中：さぎょうちゅう）』『Done（完了：かんりょう』といったステータス（status）で進捗(しんちょく)管理を行っていました。各チケットには、優先度(ゆうせんど)、締切日（しめきりび）、担当者を設定し、タスクの可視化（かじか）と整理（せいり）を徹底していました。

GitHub Issuesでは、ソースコードに直接関連するタスク管理を行いました。Issueを作成し、Pull Requestにリンクさせ、'bug'（バグ）'feature'（機能追加）などのラベルを付与（ふよ）して分類（ぶんるい）し、チーム内でコメントをしながらタスクを進めていました。

コミュニケーションには、主にSlackとMicrosoft Teamsを使用していました。Slackは日々のチーム内コミュニケーションや、スタンドアップミーティング、簡単な情報共有に活用し、Microsoft Teamsは、より正式な打（う）ち合わせ、ドキュメント共有（きょうゆう）、ビデオ会議に使用していました。プロジェクトによっては、クライアントとのやり取りにメールを使ったり、AsanaやClickUpなどのプロジェクト管理ツールを補助的に活用することもありました。"

## ４. What kind of area are you interested in? or What kind of area do you wanna work on?

### 統合

はい、私はAI、フロントエンド、バックエンドのいずれの分野にも強い関心があります。

### AI分野

AI技術とその応用（おうよう）に興味があります。現在は、オープンAIモデルを用いたプロジェクトや、学習（がくしゅう）済み（ずみ）AIモデルのカスタマイズにも携わっています。今後は、機械（きかい）学習モデルを自分で設計・構築したいと思っています。つまり、MLエンジニアになりたいと思っています。

I don’t know about Rust language well yet

### Frontend

In frontend development, I am particularly interested in **component-based architecture** using React and TypeScript, focusing on **maintainable and scalable codebases**.

I'm also keen on optimizing **state management** with solutions like Redux Toolkit and React Query, and I’m passionate about creating **accessible (a11y-compliant)** and **responsive UIs** using Material-UI or TailwindCSS.

Furthermore, I would like to deepen my expertise in **progressive web apps (PWA)** development, **micro-frontend architecture**, and performance tuning techniques such as **lazy loading**, **code splitting**, and **pre-fetching**.

フロントエンドでは、ReactとTypeScriptを活用した**コンポーネントベースアーキテクチャ**に特に興味があります。**また、**Material-UIやTailwindCSSを使った**レスポンシブなUI構築**にも強い関心があります。  
今後はさらに、**プログレッシブウェブアプリ（PWA：**Webサイトだけど、アプリのように使える技術です。**）**の開発**、マイクロフロントエンドアーキテクチャへ（**Micro Frontend Architecture：小さ（ちいさ）な独立モジュールに分けて（わけて）開発・運用するアーキテクチャ）の**取り組み、また遅延読み込み**や**コード分割**、**プリフェッチ**などを活用したパフォーマンス最適化にも力を入れていきたいと考えています。

プリフェッチ（Prefetch）」とは、将来必要になりそうなデータやリソースをあらかじめ読み込んでおく技術です

### Backend

On the backend side, I am particularly passionate about designing **RESTful APIs** and **GraphQL APIs** with Node.js (Express, NestJS) and database modeling with **MongoDB** and **PostgreSQL**.  
I enjoy creating **scalable distributed systems**, applying techniques like **database indexing**, **query optimization**, and **caching strategies** (Redis, CDN integration) to maximize system performance.  
Security is also a key focus for me, particularly implementing **authentication/authorization** using OAuth2, JWT, and **RBAC (Role-Based Access Control)**.  
In the future, I want to specialize more in **microservices architecture**, **serverless backend development** (AWS Lambda, API Gateway), and building systems with strong emphasis on **observability** through logging, tracing, and monitoring (using tools like Prometheus, Grafana).

バックエンドでは、Node.js（Express、NestJS）を用いた**RESTful API**および**GraphQL API**の設計・開発に強い関心があります。MongoDBやPostgreSQLを使った**データベースモデリング**、そしてシステムパフォーマンス向上のための**インデックス（**index**）**設計や**クエリ**(query)**最適化**、**キャッシュ**(cache)戦略(せんりゃく)**（Redis、CDN連携）**にも興味を持っています。 **また、OAuth2やJWT、****RBAC（Role-Based Access Control制御）****を**用いた認証・認可設計にも注力してきました。今後はさらに**、マイクロサービスアーキテクチャ**(microservices architecture)**、サーバーレス**(serverless)**開発（AWS Lambda, API Gateway）、**そして**ログ（**logging**）・トレーシング**(tracing)**・モニタリング**(monitoring)による**オブザーバビリティ**(observability)向上（Prometheus、Grafanaなどのツール活用）にも力を入れ、スケーラブルで信頼性の高いシステム基盤を構築していきたいと考えています。

## 5. Which technologies will you use for our project development? Please tell me the reason.

**Frontend**:

Based on my experience and the needs of your project, I would propose using the following technologies, carefully selected for their flexibility, scalability, and performance:

* **React.js**: Given my extensive experience in frontend development, React.js is my preferred choice for building component-based user interfaces. It’s ideal for creating highly interactive, responsive, and maintainable applications. For instance, in my previous projects, I used React to build intuitive UIs and worked extensively with **React Hooks** and **Context API** for state management, ensuring efficient data handling.
* **Next.js**: I believe **Next.js** would be a great addition for your project due to its built-in features for **server-side rendering (SSR)** and **static site generation (SSG)**. This ensures better SEO, performance, and overall user experience. It’s also highly compatible with React, which I’m already familiar with, and has powerful routing and API features.
* **Tailwind CSS**: For styling, I prefer **Tailwind CSS** as it allows for faster development and highly customizable designs, ensuring both responsive and accessible user interfaces. It’s a utility-first framework that helps me create clean, reusable, and consistent designs without worrying too much about custom CSS.

私の経験と貴社のプロジェクトに必要な要件を考慮した上で、以下の技術を提案させていただきます。それぞれの技術は、柔軟性、スケーラビリティ、パフォーマンスを兼ね備えており、貴社のプロジェクトに最適だと考えています。

**フロントエンド**:

* **React.js**: フロントエンド開発において、私は**React.js**をよく使用しており、コンポーネントベースのインターフェースを作成する際に非常に有用だと考えています。ユーザーインターフェースのインタラクティブ(interactive)性と保守性を確保しながら、**React Hooks**や**Context API**を使用して状態管理を行いました。これにより、データの効率的な取り扱いが可能になります。
* **Next.js**: 貴社のプロジェクトには、**Next.js**が適していると考えます。Next.jsは、サーバーサイドレンダリング（Server Side Rendering: SSR）や静(せい)的サイト生成（SSG）に組み込まれている機能を活用することで、SEOやパフォーマンス、ユーザー体験を大幅に向上させることができます。Reactと高い互換性（ごかんせい）があり、ルーティングやAPI機能が強力です。
* **Tailwind CSS**: スタイリング（styling）には、**Tailwind CSS**を使用します。Tailwindは、ユーティリティファースト(utility-first)のフレームワークで、レスポンシブでカスタマイズ可能なデザインを迅速に構築でき、シンプルで再利用可能なクリーンなデザインを作成するのに非常に便利です。

**Backend**:

* **Laravel (PHP)**: On the backend, I have experience working with **Laravel**, which is a highly efficient PHP framework known for its elegant syntax and built-in tools like **Eloquent ORM**, **queues**, and **middleware**. Laravel is ideal for projects requiring clean code structure and high scalability, and I have used it extensively to create RESTful APIs and complex backend systems.
* **Node.js**: Additionally, for performance-oriented, real-time applications, I often leverage **Node.js**. With **Express.js** or **NestJS**, I can create fast and scalable APIs that handle asynchronous processing seamlessly, which is ideal for real-time data and applications with high traffic.
* **Database (MySQL/PostgreSQL)**: For databases, I’ve worked extensively with **MySQL** and **PostgreSQL**, both of which are reliable and efficient relational databases. **MySQL** is ideal for web applications where speed and reliability are critical, while **PostgreSQL** is suitable for complex, data-intensive applications that require advanced features like JSONB support and complex queries. I’m comfortable designing efficient schemas and optimizing queries in both databases.
* **Redis**: To enhance performance and manage high volumes of data, **Redis** would be used for caching. In my previous roles, I’ve implemented caching strategies to reduce latency and improve response times, particularly in applications that require high throughput.

**バックエンド**:

* Laravelは、エレガント(elegant)な文法（ぶんぽう）と、Eloquent ORM・キュー・ミドルウェアなどの便利な組み込みツールを備え（そなえ）たPHPフレームワークで、特にクリーン（clean）なコード構造や高いスケーラビリティが求められるプロジェクトに最適です。RESTful APIの設計や複雑（ふくざつ）なバックエンドシステムの構築に広く(ひろく)使用してきました。
* **Node.js**: リアルタイムデータや高(こう)トラフィック（traffic）のアプリケーションには、**Node.js**も活用します。**Express.js**や**NestJS**を使用し、非同期処理を活かした高性能でスケーラブルなAPIを構築できます。

リアルタイムデータ：たとえばチャット、通知、オンラインゲームのように、ユーザーの操作にすぐ反応（はんのう）する必要があるデータ。

高トラフィック：たくさんのユーザーが同時にアクセスするような状況（例：ECサイトのセール、人気SNS、ライブ配信など）。

* **データベース (MySQL/PostgreSQL)**: データベースには、**MySQL**と**PostgreSQL**を使用してきました。MySQL は、処理速度と信頼性が大切なウェブアプリに向いているデータベースです。PostgreSQL は、複雑なデータの取り扱いや JSON 形式のデータのサポートが必要な、大量のデータを扱う（あつかう）アプリケーションに向いています。どちらのデータベースでも、効率的なスキーマ設計とクエリの最適化を行っています。
* **Redis**: Redisは「超高速（ちょうこうそくい）なメモリ上のデータ保管庫（ほかんこ）」で、アプリのパフォーマンス向上に非常に役立つツールでパフォーマンス向上のために、**Redis**を使用してキャッシュ（cache）処理を行います。過去のプロジェクトでキャッシュ戦略(せんりゃく)を実装し、レイテンシ（latency）を減らし(へらし)、応答（おうとう）速度を向上させることに成功しました。これにより、高トラフィック（traffic）のアプリケーションでも優れた(すぐれた)パフォーマンスを維持（いじ）できます。

**Reasoning**:  
The combination of these technologies provides a robust, scalable, and efficient tech stack that would allow for rapid development, enhanced performance, and future-proof solutions. Additionally, I have practical experience with each of these technologies, which will allow me to hit the ground running and contribute effectively from day one.

**理由**:  
この技術スタックは、迅速な開発、パフォーマンス向上、将来にわたってスケーラブルなソリューションを提供するために非常に効果的です。また、私自身がこれらの技術を実務で活用してきた経験が豊富で、即戦力として貢献できると自信を持っています。

## 6. How did you lead or support your team as a sub-leader? チームリーダーやサブリーダーとしてどのようにチームをまとめましたか？

効果（こうか）的なチームワークを確保するため、週に2～3回の定例ミーティングを開催（かいさい）しています。何かトラブル（trouble）があったら、それについて話し合って、どうやって対応していくかを決めています。また、Gitなどのコラボレーション(collaboration)ツールを活用し、全員の進捗状況を確認・把握(はあく)しています。

## 7. what was the first trouble in frontend?

Although there used to be some troubles before, the first trouble I met was in CSS. The project was for E-learning system, material UI was used to make easy frontend. As you know, material UI was the most popular React UI framework but a little confused. The material UI has its own components and every component has own props and CSS. Sometimes we need to change a style sheet supported by material UI. But the adding style sheet might conflict with CSS defined by user. To solve this problem, I took a lot energy to solve it. The method is to use “withstyles” function supported by material UI to cover origin style.

## 8. How do you do global variables and functions in React?

In frontend, we often use “usestate”, “useEffect” hook to handle state in a component. But in a component tree, it needs to transfer state from parent component to child component or vice versa. You can handle every state using function property of component in props. But passing state down and up the tree through dozens of components is tedious especially for a complex application.

In react, there is a hook-“usecontext” to handle this.

First, create context using the hook - **createContext().**

Second, define the global variables and functions by using **context provider component**.

Third, place the defined provider component at the top level to wrap around your entire component trees.

Fourth, define a function by using useContext() to access global variables and functions at any child component levels using the function.

# General question.

## **2. How have you ensured high quality and maintainability in your code during long-term projects?**

**English:**"Over nearly a decade in software development, I’ve adopted best practices such as code reviews, proper version control with Git, comprehensive documentation, and unit testing. By following established coding standards and refactoring regularly, I ensure the code remains maintainable and scalable as projects grow."

**Japanese:**約１０年にわたるソフトウェア開発の経験から、**コードレビュー**、**Git** を用いた**バージョン管理**、充実（じゅうじつ）した**ドキュメンテーション**、**ユニットテスト**などのベストプラクティス（practice）を導入しています。確立（かくりつ）されたコーディング標準に沿って（そって）、定期的にリファクタリング（refactoring）を行うことで、プロジェクトが大きくなっても、コードの保守性やスケーラビリティ(scalability)をしっかり維持(いじ)できるようにしています。

## **3. What are the key differences between React and Vue, and when would you choose one over the other?**

**English:** React is javaScript library and React introduced a new idea: writing HTML inside JavaScript using JSX. The react is more flexible but needs extra libraries for routing, state management. On the other hand Vue is a framework-like and keeps HTML, CSS and JS more separated. Vue is more easier for beginners due to its simple template syntax.

**Japanese:** ReactはJavaScriptライブラリであり、JSXを用いてJavaScript内でHTMLを記述 [きじゅつ]するという新しい考え方を導入しました。Reactはより柔軟ですが、ルーティングや状態管理するため、追加ライブラリが必要です。一方[いっぽう]で、VueはJavaScriptフレームワークであり、テンプレートを用いて, HTML、CSS、JavaScriptをより分離しています。Vueはシンプルなテンプレート構文[こうぶん]のため、初心者(しょしんしゃ)にとってより扱い[あつかい]やすいです。

## 4. what is the difference between MySQL and MongoDB?

MySQL is a relational database whereas MongoDB is No SQL. In MySQL, data is stored in tables which have rows and columns whereas in MongoDB, the actual data is stored in documents. In MySQL, data is stored in a fixed schema whereas in MongoDB, data is stored in a flexible schema.

MySQLはリレーショナル（relational）データベースですが、MongoDBはNo SQLです。MySQLでは、データは行（ぎょう）と列（れつ）を持つテーブルに保存[ほぞん]されます、MongoDBでは、実際のデータはドキュメント(document)に保存されます。MySQLではデータは固定（そてい）なスキーマで保存（ほぞん）されますが、MongoDBではデータは柔軟なスキーマで保存されます。

## **5. How do you manage cross-browser compatibility issues in web applications?**

**English:**I typically use **progressive enhancement** and responsive design principles along with testing tools like **BrowserStack** to ensure compatibility across different browsers. I also rely on modern JavaScript features with proper polyfills and libraries to handle discr**e**pancies between browsers.

**Japanese:** プログレッシブエンハンスメント（progressive enhancement）やレスポンシブデザインの原則を用いて、BrowserStack などのテストツールを使って各ブラウザ間の互換性[ごかんせい]を確認しています。また、モダンなJavaScript の機能を適切なライブラリと併用[へいよう]することで、ブラウザごとの差異[さい]に対処[たいしょ]しています。

## **6. Describe a challenging bug you encountered and how you resolved it.**

**English:**"I once faced an issue with inconsistent data rendering on the client side caused by asynchronous API calls. After extensive debugging, I implemented a middleware to standardize data formats and improved error-handling mechanisms. This ensured that the frontend reliably displayed the updated information without lag."

**Japanese:** かつて、非同期API 呼び出しにより、クライアント側(がわ)でデータが一貫して表示されない問題に直面しました。徹底(てってい)的なデバッグの結果、データフォーマットを標準化(ひょうじゅんか)するミドルウェアを実装し、エラーハンドリング(handling)を改善(かいぜん)しました。その結果、フロントエンドがラグ（lag）なく確実に最新情報を表示できるようになりました。

## **7. How do you ensure effective teamwork and communication in a development team?**

**English:**" In order to ensure effective teamwork, we have a standard meeting 2 or 3 times in a week, if we face a trouble problem, we suggest the problem and discuss how to tackle the problem and solve it. In addition, we use the collaborative tools such as Git for checking and ensuring everybody's progress."

**Japanese:** 効果（こうか）的なチームワークを確保するため、週に2～3回の定例ミーティングを開催（かいさい）しています。何かトラブル（trouble）があったら、それについて話し合って、どうやって対応していくかを決めています。また、Gitなどのコラボレーション(collaboration)ツールを活用し、全員の進捗状況を確認・把握(はあく)しています。

## **8. How do you keep your technical skills current in a rapidly evolving field?**

**English:**"I actively participate in online courses, attend tech meetups, and follow influential blogs and repositories on platforms such as GitHub. Additionally, I contribute to open-source projects when possible, which helps me stay updated with the latest trends and best practices in web development."

**Japanese:** **オンラインコース**に積極的に参加し、**テックミートアップ**(tech meetup)に出席(しゅっせき)、また **GitHub** のようなプラットフォームで有力な**ブログ**(blogs)やリポジトリをフォロー（follow）しています。さらに、オープンソースプロジェクトに貢献することで、最新のトレンド(trend)やベストプラクティス(practice)を継続（けいぞく）的に学んでいます。

# JavaScript**編**

## **１．What is the DOM? DOMなんですか？**

DOM (Document Object Model) is an interface (mechanism) for manipulating HTML and XML documents programmatically. When a web page is loaded, the browser interprets the page structure as a tree structure (a collection of nodes) and generates the DOM.

DOM（Document Object Model） は、HTMLやXMLドキュメントをプログラムから操作するためのインターフェース（仕組み）です。Webページが読み込まれると、ブラウザはページの構造をツリー構造（ノードの集まり）として解釈（かいしゃく）し、DOMを生成いたします。

## **2. Explain event delegation. イベント デリゲーションについて説明せよ。**

This is one of the direct DOM manipulations using JavaScript. By using event delegation, for example, you can pass the same onclick to multiple li under a ul. The most commonly used delegation command is the EventTarget.addEventListener() method.

JavaScriptによる直接（ちょくせつ）的なDOM操作の一つでございます。イベント デリゲーションを用いると、たとえばul配下の複数のliに同一のonclickを渡すことができます。delegation命令で最もよく用いられるのはEventTarget.addEventListener()メソッドです。

## **3. Explain “this” function in JavaScript. JavaScriptにおけるthisの働きについて説明せよ。**

When this is called as a method or member of an object, this is a reference to the object it belongs to.

オブジェクトのメソッドやメンバとしてthisが呼ばれるとき、thisは自身の所属するオブジェクトを示す参照でございます。

## **4. Explain prototype inheritance プロトタイプ継承がどのように振る舞う[ふるまうbehave]か説明せよ。**

A prototype is the base that all JavaScript objects have, and is an internal connection to other objects. Prototype inheritance is equivalent to connecting the prototype chain. More specifically, when extended.a is commanded, JavaScript first searches the properties of the extended object, and if no suitable property is found there, it searches for .a in origin, which is the inheritance source.

プロトタイプとは、JavaScriptのすべてのオブジェクトがもつ原型(げんけい)であり、他のオブジェクトへの内部的な繋がり [つながり]でございます。プロトタイプの継承とは、プロトタイプチェーンを結ぶ（むすぶ）ことに相当(そうとう)いたします。より具体的にいうと、extended.aが命令されると、JavaScriptはオブジェクトextendedのプロパティをまず探し、そこに適切なプロパティが見当（みあ）たらないと、継承元（もと）であるoriginに.aがないかを探しに行きます。

## **５. nullとundefinedとundeclaredの違いについて説明せよ。**

The distinctions among these concepts depend on whether a variable has been declared and whether it has been assigned a value. In case of the undefined, a variable is declared but not assigned a value

In case of the undeclared, the variable has never declared with let, const or var.

In JavaScript, null is a value that means ‘nothing’ or ‘empty intentionally’.

これらの概念の違いは、変数が宣言されているかどうか、そして値(あたい)が代入されているかどうかによって異（こと）なります。undefined の場合、変数は宣言されていますが、値は代入されていません。undefined の場合、変数は let、const、または var で宣言されていません。JavaScript では、null は「何もない」または「意図的に空（そら）である」ことを意味する値です。

## **6. Describe closures. クロージャについて述べよ。クロージャとは何か、どのように機能するか、利用する理由について特に説明せよ。**

A closure is a function that references an independent variable. To be more specific, the concept of a closure splits the function itself and the environment in which the function is created.

When a function is defined within another function, the inner function uses the variables of the outer function. This inner function is just a closure. For example, from the React component Modal, the UserModal and PaymentModal, which share most of their behavior but have slightly different properties, can be generated from the same function. Closures increase the reusability of functions and provide a data structure similar to object-oriented.

クロージャは独立した変数を参照する関数でございます。もう少し詳しく言うと、クロージャという概念(がいねん)は関数そのものと関数が生成された環境に分割いたします。

関数が別の関数内で定義されている場合、内側[うちがわ]の関数は外側 [そとがわ]の関数の変数を使用いたします。この内側[うちがわ]の関数はクロージャでございます。たとえばReactコンポーネントModalから、ほとんどの動作が共通であり、しかし微妙[びみょう]にプロパティが異なるUserModalとPaymentModalは同じ関数から生成できます。クロージャは関数の再利用性を高め、オブジェクト指向[しこう]に似たデータ構造を提供いたします。

function createModal(modalType) {

// 外側の関数の変数（modalType）は内側でも使える

return function showModal(userName) {

console.log(`${modalType} モーダルを ${userName} に表示します。`);

};

}

// UserModal と PaymentModal を作る

const showUserModal = createModal("ユーザー");

const showPaymentModal = createModal("支払い");

// 呼び出してみる

showUserModal("山田さん"); // → ユーザーモーダルを 山田さん に表示します。

showPaymentModal("佐藤さん"); // → 支払いモーダルを 佐藤さん に表示します

## **７．オブジェクトのイテレーション（iteration）にどんな構文[こうぶんsyntax]を用いるか述べよ。**

* for/in
* forEach
* for(let i; i<n;i++){}

JavaScriptにはたくさんのループがありまして、気に入ったものを使ってきます。

## **8. Difference between foreach and map. foreachとmapの違いについて説明せよ。どのような場合にどちらを用いるのが適当か。**

As the words suggest, Array.foreach() refers to an operation performed on each element of an array, while Array.map() is a mapping applied to an array (vector). For this reason, the latter is expected to return a new array value. In terms of usage scenarios, for example, the former method is more suitable for operations such as registering elements one by one in a database. On the other hand, in cases where a linear mapping is applied to a vector, I think it is appropriate to use Array.map().

語義のままであるが、Array.foreach()は配列の一つずつに対して行う処理を示し、Array.map()は配列(ベクトル)に適用する写像（しゃぞう）である。こういう理由で，後者は新たな配列値を返すことを期待します。利用シーンで言いますと、例えばデータベースに一つずつ要素を登録していくような処理には、前者の方法のほうが適していると思います。一方で、あるベクトル（vector）に線形（せんけい）写像を適用するようなケースでは、Array.map()を使うのが適切だと思います。

## **9. host objects と native objectsの違いを述べよ。**

Both are concepts in the global scope, but a host object is an object provided by the execution environment, such as a window or document, while a native object is an object that JavaScript has built-in, such as an Array.

どちらもグローバルスコープ（global scope）にある概念だが、host objectはwindowやdocumentのように実行環境によって提供されるオブジェクトでございます。一方[いっぽう]native objectはJavaScriptが組み込みとして持っているオブジェクトです。たとえばArrayはそうです。

## 10. new演算子がなんですか

The new operator is a built-in mechanism in JavaScript used to create object instances. For example, when you use new Person(), a new object that inherits from Person.prototype is first created. Next, the constructor function is called, and if any arguments are specified, they are used in the internal processing. Finally, the fully constructed object is returned — that’s the overall flow.

new 演算子は、JavaScript に組み込まれている仕組みで、オブジェクトのインスタンスを作成するために使われます。たとえば new Person() のように使うと、まず Person.prototype を継承した新しいオブジェクトが生成されます。次に、**コンストラクタ（constructor）**関数が呼び出され、もし引数が指定されていれば、それを使って内部で処理が行われます。そして最後に、完成したオブジェクトが返（かえ）される、という流れになります。

## **11. event capturingについて書け。**

For example, when an event is fired at an HTML element, how does the event propagate? Bubbling and capturing are the methods of propagation.

Bubbling is when an event flows from child to parent.

Capturing is when an event flows from parent to child.

例えば、HTML要素について、ある要素においてイベントが発火したとき、そのイベントはどのように伝搬[でんぱん]するだろうか。bubblingとcapturingはその伝搬方法です。

bubblingというは子から親にイベントが流れていることです。

capturingというは親から子にイベントが流れていることです。

## **12. attributeとプロパティの違いを説明せよ。**

In short, an attribute is an argument passed to a DOM element, and a property is a member variable of an object. For example, if you want to get information about a p tag, you can use Element.attributes(). A property (particularly in JavaScript) refers to a set of information associated with a data structure.

簡単に言うとattributeはDOM要素に渡す引数で、propertyはオブジェクトのメンバ変数です。例えばp タグの情報を取得したい場合は、Element.attributes() を使用できます。propertyは(特にJavaScriptにおいて)あるデータ構造と関連された情報群を指(さ)す。

## **13.== と ===（**三重イコール**） の違いを説明せよ。**

== は「等価演算子，equality operator」、=== は「厳密等価演算子:strict equality operator」。

The new operator is a built-in mechanism in JavaScript used to create object instances. For example, when you use new Person(), a new object that inherits from Person.prototype is first created. Next, the constructor function is called, and if any arguments are specified, they are used in the internal processing. Finally, the fully constructed object is returned — that’s the overall flow.

同値判定をする場合には基本的に == よりも === が使います。その大きな違いは、== の場合、暗黙[あんもく]的に型変換 [かたへんかん]が行われる点です。しかし、「===」の場合、データ型が異なると左側(ひだりがわ)と右側（みぎがわ）が同じになりません。そのため、結果は false になります。

## **14. Function.call および Function.applyの働きを説明せよ。またその顕著な違いについても説明せよ。**

Function.call is a function that assigns a property belonging to one object to another object and calls it. .apply is almost the same thing, but differs in that the second args take an array as the argument type.

Function.callはあるオブジェクトに所属するプロパティを、別なオブジェクトに割り当て（わりあて）て呼ぶ関数である。

.applyはほとんど同じものであるが、2つめ以降のargsにおいて、配列を引数の型として取るようになっている点で異なる（ことなる）。

## **15. Function.prototype.bind について説明せよ。**

Imagine you have a function that depends on ‘this’ operator , and when you call it, the ‘this’ points to the wrong object. You can use bind() to lock this to the right object so that your function works correctly.

this に依存する関数があると想像してください。それを呼び出すとthis が間違ったオブジェクトを指（さ）しているます。bind() を使用すると、関数が正しく動作するようにこれを適切なオブジェクトにロックできます。

const person = {

name: 'Alice',

greet: function() {

console.log('Hi, I am ' + this.name);

}

};

const greetFunc = person.greet;

greetFunc(); // ❌ "Hi, I am undefined" — because `this` is lost!

const greetFunc = person.greet.bind(person);

greetFunc(); // ✅ "Hi, I am Alice"

## **16. Explain hoisting. 巻き上げ（まきあげ：hoisting）について説明せよ。**

Hoisting is a mechanism related to the execution order of code specific to JavaScript.Hoisting is JavaScript's behavior of moving declarations to the top of the current scope (either global or function scope) before code is executed. For example, var and function declarations are treated as being at the top of the code. However, although the declaration is processed first, the actual execution or initialization does not occur there. On the other hand, with let and const, which are commonly used these days, hoisting does not generally occur, allowing you to write code more intuitively.

巻き上げ（ホイスティング）は、JavaScript特有のコードの実行順(じゅん)に関する仕組みです。たとえば、varや関数の宣言（せんげん）は、コードの先頭（せんとう）にあるものとして扱われます。ただし、宣言は先に処理されますが、実際の実行や初期化はその場所で行われるわけではありません。一方、最近よく使われるletやconstでは、基本的に巻き上げは起（お）こらないため、より直感的にコードを書くことができます。

## **17.extending JavaScript built-in object. JavaScriptの組み込みオブジェクトを拡張する利点と欠点[けってん]について述べよ。**

Although extending JavaScript's built-in objects has its benefits, it is generally considered best to avoid it.

For example, you can add useful functionality to your own projects, but built-in objects are globally shared, and your changes to them can negatively impact other people's code and libraries.

For this reason, it is best to avoid directly extending built-in objects, especially when working in a team or using shared libraries.

JavaScriptの組み込みオブジェクトを拡張することには、メリット(merit)もありますが、基本的には避け(さけ)た方が良いとされています。たとえば、自分のプロジェクトで便利な機能を追加できるという利点はありますが、組み込みオブジェクトはグローバルに共有されているため、そこに手を加える（てをくわえる）と、他の人のコードやライブラリに悪影響（あくえいぎょう）を与（あた）えてしまう可能性があります。

そのため、特にチーム開発や共有ライブラリを使う場合には、組み込みオブジェクトを直接拡張するのはやめておいたほうがいいです。

## **18.tenary operator. 三項演算子について説明せよ。**

A ternary operator is an operator that uses three terms to switch values depending on conditions. It is used in JavaScript, PHP, Python, etc.

三項演算子とは、\*\*3つの項（こう）\*\*を使って、条件によって値(あたい)を切り替える演算子のことです。JavaScriptやPHP、Pythonなどで使われます。

基本の形：条件式 ? trueのときの値 : falseときの値

## **19. Languages that compile to JavaScript. JavaScriptへとコンパイルする言語について、利用することのメリットとデメリットを説明せよ。**

**JavaScriptへコンパイルされる代表的な言語はTypeScriptです。JavaScriptにMicrosoftが開発して型（Type）を追加した言語です。コードが型を使ってエラーを実行前に発見できる。**

**A typical language that is compiled to JavaScript is TypeScript. It is a language developed by Microsoft that adds types to JavaScript. Code uses types to detect errors before execution.**

# **React編**

## **What is React? React.js とは何ですか？**

React is an open-source front-end JavaScript library that is used for building user interfaces, especially for single-page applications. It is used for handling view layer for web and mobile apps. The main concept is to break down the UI into small independent components that can be combined and reused.

Reactは、オープンソースのフロントエンドJavaScriptライブラリで、特にSPA（single page application）のユーザーインターフェイスを構築することができ、WebアプリケーションやMobileアプリケーションのViewレイヤーを処理するために使用されます。主要（しゅよう）な概念は、UI を小（ちい）さな独立したコンポーネントに分割し、それらを組み合わせて再利用できるようにすることです。

## **What are the major features of React?　React.js の主な機能は何ですか？**

* **JSX** (JavaScript XML): JSX allows you to write HTML-like syntax in JavaScript, making your code easier to understand and debug.
* **Components**: In React, everything is represented as a component. These components are reusable, independent pieces of UI.
* **Virtual DOM**: React optimizes UI updates using a Virtual DOM, minimizing direct DOM manipulation for faster rendering.
* **unidirectional data binding**: React follows a unidirectional data flow, making your code predictable and easier to debug.
* **Declarative**: React allows you to declaratively describe how your UI should look based on state, abstracting away manual DOM manipulation.
* **JSX (JavaScript XML):** JSX を使用することで、HTML のような構文[こうぶん]を JavaScript 内で記述できます。コードの理解やデバッグが容易になります。
* **コンポーネント:** React では、すべてがコンポーネントで表（あらわ）されます。これらのコンポーネントは再利用可能で、独立した UI の部分です。
* **仮想**（かそう） DOM: React は、仮想 DOM を使用して UI 更新を最適化し、直接的な DOM 操作を最小限に抑える[おさえる]ことで、レンダリングを高速化します。
* **一方向データバインディング**: React は一方向のデータフローに従う(したがう)ため、コードが予測（よそく）可能でデバッグが容易です。
* **宣言的**: React を使用することで、状態に基づいて UI がどのように見えるべきかを宣言的に記述でき、手動での DOM 操作を抽象（ちゅうしょう）化します。

## What is virtual DOM? How does it work? **仮想 DOM とは何か、React でどのように機能しますか？**

The virtual DOM is a lightweight copy of the real DOM. When a component's state or properties change, React creates a new virtual DOM and compares it with the previous one. This process is called "diffing". React identifies the differences and efficiently updates the real DOM only with what is needed. This makes UI updates fast and smooth.

仮想 DOM は、実際の DOM の軽量(けいりょう)なコピーです。コンポーネントの状態やプロパティが変わる（かわる）と、React は新しい仮想 DOM を作成し、以前のものと比較（ひかく）します。このプロセスは「diffing」と呼ばれます。React は差分（さぶん）を特定し、必要な部分のみを効率的に実際の DOM に反映（はんえい）します。これにより、UI の更新が高速でスムーズになります。

## **What is the difference between Class component and Function component? クラスコンポーネントと関数コンポーネントの違いは何ですか？**

* **Class components:** Introduced in early React, they inherit from React.Component, use lifecycle methods (e.g. componentDidMount and render()), and use a constructor to manage state and event bindings.
* **Function Components:** Simple JavaScript functions that take props and return JSX. With the introduction of React hooks (useState, useEffect), they have become popular instead of class components. Their code is more concise and easier to test.
* **クラスコンポーネント:**初期の React で導入され、React.Componentを継承し、ライフサイクルメソッド（例：componentDidMount や render()）を使用します。constructorを使用して状態やイベントバインディングを管理します。
* **関数コンポーネント:**シンプルな JavaScript 関数で、プロパティを受け取り JSX を返します。React hooks (useState, useEffect) の導入により、クラスコンポーネントに代（か）わって広く使われるようになりました。コードが簡潔(かんけつ)で、テストがしやすいです。

## What is the hook and its important? **React フックとは何ですか？また、その重要性は何ですか？**

Hooks are functions that allow you to "hook into" React state and lifecycle features from your function components. Common hooks include:

useState: Allows you to add state to your function components.

useEffect: Used to perform side effects, such as fetching data or manipulating the DOM. Hooks allow you to simplify your code by not using class components and lifecycle methods.

フックは、関数コンポーネントから React の状態やライフサイクル機能に「フック」できるようにする関数です。代表的なフックには以下があります：

useState: 関数コンポーネントに状態を追加することができます。

useEffect: サイドエフェクトを実行するために使用され、データの取得や DOM 操作に利用されます。 フックにより、クラスコンポーネントやライフサイクルメソッドを使わずにコードを簡素化できます。

## Explain useEffect hook. useEffect フックを説明してください。

useEffect is a hook to run side effects in a component. It is similar to componentDidMount, componentDidUpdate, and componentWillUnmount in class components. It is used to fetch data, set up subscriptions, or manually manipulate the DOM. By default it runs after every render, but you can control this by passing a dependency array as the second argument.

useEffectは、コンポーネント内でサイドエフェクトを実行するためのフックです。これはクラスコンポーネントのcomponentDidMount、componentDidUpdate、およびcomponentWillUnmountに似ています。データ取得、サブスクリプションの設定、または DOM の手動操作に使用されます。デフォルトでは、すべてのレンダー後に実行されますが、依存配列を第二引数として渡すことで制御（せいぎょ）できます。

## **what is the purpose of the Key in the React? React で key プロパティの目的は何ですか？**

The key property is a unique identifier used to identify an item in the list, allowing React to efficiently track changes, additions, or removals in the list. The key allows you to update only what's needed, without having to re-render the entire list.

keyプロパティは、リスト内のアイテムを識別するための一意の識別子です。これにより、React はリスト内の変更、追加、または削除を効率的に追跡[ついせき]できます。keyがあることで、必要な部分のみを更新することができ、リスト全体を再レンダリングする必要がなくなります。

## **What is the Context API of the React? React の Context API とは何ですか？**

The Context API is a feature for sharing data between components, eliminating the need to manually pass props at every level. It is used to manage global state such as themes or user information. By using the createContext and useContext functions, you can avoid prop drilling and keep your code cleaner.

Context API は、コンポーネント間でデータを共有するための機能で、すべてのレベルでプロパティを手動で渡す必要がなくなります。テーマやユーザー情報などのグローバルな状態を管理するために使用されます。createContext と useContext 関数を使用して、プロパティの多段階伝達(でんたつ)（プロップドリリング：prop drilling）を回避(かいひ)し、コードをよりクリーン（clean）に保つ（たもつ）ことができます。

## What is the advantage of the Redux in React? **Redux を React で使用する利点は何ですか？**

Redux is a library used for state management in complex applications. It is often used in combination with React and provides the following benefits: there are four main factors in Redux: action, reducer, store, UI

Centralized state: All application state is stored in one place and is easy to manage.

Predictable state updates: Redux actions and reducers make state transitions predictable.

Easy to debug: With Redux DevTools you can track state updates per action.

Improved performance: Reduce unnecessary re-renders and optimize performance

Redux は、複雑なアプリケーションの状態管理に使用されるライブラリです。React と組み合わせて使用されることが多く、以下の利点があります：

* **状態の集中管理:** すべてのアプリケーション状態が 1 つの場所に保存され、管理が容易です。
* **予測可能な状態更新:** Redux のアクションとリデューサーにより、状態遷移が予測可能になります。
* **デバッグの容易さ:** Redux DevTools を使用することで、アクションごとの状態更新を追跡できます。
* **パフォーマンス向上:** 不要な再レンダリングを減らし、パフォーマンスを最適化します。

## **React の Higher-Order Components (HOCs) とは何ですか？**

A Higher-Order Component (HOC) is a function that takes a component as an argument and returns a new component. HOCs are used to give existing components additional functionality or logic, and are useful for concerns that are shared across multiple components, such as authentication or logging.

Higher-Order Component (HOC) は、コンポーネントを引数として受け取り、新しいコンポーネントを返す関数です。HOC は既存のコンポーネントに追加の機能やロジックを付与するために使用され、認証やロギングなど、複数のコンポーネント間で共有される関心事に役立ちます。

## **What are React Fragments and When would I use them? React フラグメントとは何ですか？また、いつ使用しますか？**

React fragments allow you to return multiple elements from a component without adding extra nodes to the DOM. Fragments are useful when enclosing them in a parent element would unnecessarily increase the depth of the DOM tree. You can define a fragment using <React.Fragment> or the shorthand <>.

In a short, in React, a **Fragment** (written as <React.Fragment>...</React.Fragment> or its shorthand <>...</>) lets you group a list of children **without** adding an extra DOM node.

React フラグメントは、DOM に余分なノードを追加することなく、コンポーネントから複数の要素を返すことができます。フラグメントは、親要素で囲むと DOM ツリーの深さが不必要に増える場合に役立ちます。<React.Fragment> または省略形の <> を使用してフラグメントを定義できます。

What is the React Fiber? **React Fiber とは何ですか？**

React Fiber is a new arbitration engine in React 16 and higher that optimizes the prioritization of rendering tasks. This helps React manage updates efficiently in applications with a large number of rendering requests, improving performance during animations and data-intensive operations.

React Fiber は、React 16 以降の新しい調停(ちょうてい)エンジンで、レンダリングタスクの優先順位（ゆうせんじゅんい）を最適化します。これにより、大量のレンダリング要求を持つアプリケーションにおいて、React が効率的に更新を管理できるようになり、アニメーションやデータ集約型操作の際にパフォーマンスが向上します。

# プロジェクトのリリース方法を学ぶ。（how to learn release a project.）

1. Prepare Production-Ready Files（制作準備ファイルを準備する必要がある）

* Frontend(e.g. React)
* you run **npm run dev** to start the local development server while **npm run** **build** for release.
* Backend
* Php artisan optimize for **Laravel**
* Py manage.py runserver for **Djiango**
* Npm install –production for node.js or **Express**

2. Upload Destination（宛先：あてさき） Files to Hosting Server

* For the Frontend (React)

 **Static Hosting Services:**  
If you are hosting a static site（静的サイト）, deploy（デプロイ） your build/ folder to a service such as Netlify, Vercel, GitHub Pages, Firebase Hosting, or through FTP to your hosting provider（ホスティング プロバイダ）.

 **Custom Hosting:**  
If you’re uploading to a personal server, use tools like SCP, SFTP, or Git-based deployments to transfer（転送） your files to the correct directory (e.g., public\_html).

* **Backend Deployment:**
* Upload your server code to the **hosting provider** or cloud service.

3. Configure the Web Server (e.g., NGINX)

* For Frontend Assets:
* Configure your NGINX server block to point（指す：さす） to your deployment directory:

server {

listen 80;

server\_name yourdomain.com;

root /path/to/your/build;

index index.html;

location / {

try\_files $uri /index.html;

}

}

* For Backend Applications:

Use NGINX as a reverse proxy to forward client requests to your backend service (for example, a Node.js application running on a specific port):

**Q: Can you describe the challenges you've faced when implementing scalable web applications using React or Vue.js, and how do you optimize the performance of these applications?**

仕入れ管理システムを開発する際、フロントエンドにはReact.jsを、バックエンドにはLaravelを使用しました。

# React（リアクト）は、**ユーザーインターフェース（UI）を構築するためのJavaScriptライブラリ**です。主に**シングルページアプリケーション（SPA）の開発に使用されます。**

私は、コード**分割**（ぶんかつ）、遅延読み込み、ピュアコンポーネント（PureComponent）、Vue.js の仮想（かそう） DOM などのテクニックを使用してパフォーマンスを最適化することにできました。

Reactでのコード**分割**（[ぶんかつ] Code Splitting）とは、大きなJavaScriptファイルを複数 [ふくすう]の小さな（ちいさな）ファイルに分割して、必要なときに必要な部分だけをロードする技術です。具体的には、Reactのアプリケーションでは以下のような方法でコード分割を実現します：

**動的 [どうてき]インポート（Dynamic Imports）**

**React Routerとの組み合わせ**

**普通のコンポーネントとピュアコンポーネント**（PureComponent）**の違い：**

* **通常のコンポーネント**（React.Component）は、propsやstateが変更されるたびに再レンダリング（rendering）されます。
* **ピュアコンポーネント**（React.PureComponent）は、propsやstateが変更されない場合に再レンダリングをスキップします。

**A:** In my experience, one of the challenges with React and Vue.js is ensuring smooth performance in large-scale applications with frequent updates. I focus on optimizing performance using techniques like:

* **Code splitting** to load only the necessary chunks of JavaScript.
* **Lazy loading** of components and routes to improve initial load time.
* **Memoization** and **PureComponent** (for React) to avoid unnecessary re-renders.
* **Virtual DOM diffing** for Vue.js to efficiently manage updates. By using these techniques, I’ve been able to maintain good performance, even in complex applications.

**Q: How do you approach integrating backend technologies (PHP, Node.js, Java, C#) with modern frontend frameworks like React or Vue.js?(** **バックエンド テクノロジー (PHP、Node.js、Java、C#) を React や Vue.js などの最新のフロントエンド フレームワークと統合するにはどうすればよいでしょうか?)**

フロントエンドとバックエンド間の通信には、RESTful API または GraphQL アプローチを採用することで、スムーズな統合を実現します. バックエンドロジックを処理するために Laravel を実装しています。また、ユーザー認証と承認に JWT を使用した安全なデータ転送にも重点を置いています。バックエンドへのリクエストにはAxiosまたはFetch APIを使用し、レスポンスはReactのReduxやVue.jsのVuexなどの状態管理ライブラリで処理します。

**A:** I ensure smooth integration by adhering to a **RESTful API** or **GraphQL** approach for communication between the frontend and backend. With **Node.js**, I use Express to create APIs, while for **PHP**, I implement Laravel or Symfony to handle the backend logic. I also focus on secure data transmission using **JWT** or **OAuth** for user authentication and authorization. For frontend integration, I use Axios or Fetch API for making requests to the backend and handle the responses with state management libraries such as Redux (for React) or Vuex (for Vue.js).

**Q: How do you ensure code quality and maintainability in projects where multiple engineers are involved?(** **複数のエンジニアが関与するプロジェクトで、コードの品質と保守性をどのように確保しますか?)**

私は、コードの品質(hinshitsu)を高く保つ(tamotsu)ために、コードレビュー、ユニットテスト、統合テストの使用を重視しています。ESLint や Prettier などのツールを開発パイプラインに統合して、一貫したコーディング標準(hyōjun)を適用しています。さらに、文書化は保守性にとって重要であるため、すべてのモジュールとコンポーネントが適切に文書化されていることを確認します。

**A:** I emphasize the use of **code reviews**, **unit tests**, and **integration tests** to ensure high code quality. Tools like **ESLint** and **Prettier** are integrated into the development pipeline to enforce consistent coding standards. I also encourage the use of **CI/CD pipelines** for automated testing and deployment, ensuring that changes don't break existing functionality. In addition, documentation is key for maintainability, so I ensure that all modules and components are well-documented.

**Q: What’s your experience with DevOps or infrastructure, and how do you ensure your web applications are scalable and highly available?(** **DevOpsやインフラストラクチャに関する経験はどのようなものですか? また、Web アプリケーションのスケーラビリティと高可用性をどのように確保していますか?)**

Web アプリケーションの開発では、それを理解することが非常に重要だと思います。

私はコンテナ化のためにDockerを使ってきました。 さらに、クラウドインフラストラクチャのAWSとGCPの経験もあります。

**A:** I’ve worked with **Docker** for containerization and **Kubernetes** for orchestration to ensure scalability and fault tolerance in production environments. I also use **CI/CD pipelines** with tools like **Jenkins** or **GitLab CI** for automated testing and deployment. For monitoring and alerting, I’ve integrated **Prometheus** and **Grafana** to ensure high availability. Additionally, I have experience with **AWS** and **GCP** for cloud infrastructure, particularly with load balancing, auto-scaling, and managed services like RDS and CloudFront.

# PHPとLaravel**編**

## **MVCアーキテクチャ**

* **モデル（Model）は**：  
  データのロジックや、データベースとのやり取りを担当する部分でございます。
* **ビュー（View）は**：  
  ユーザーに表示されるUI（ユーザーインターフェース）を担当する部分でございます。
* **コントローラー（Controller）は**：  
  ユーザーからのリクエストや入力を受け取り、適切に処理する部分でございます。

## ルーティング

URL をコントローラー（controller）/メソッド(method)にマップする方法を決定します。

## データベースインタラクション(Database intro)

Eloquent (Laravel) などの ORM (オブジェクトリレーショナルマッピング) の使用

CRUD 操作 (作成、読み取り、更新（kōshin）、削除（sakujo）)

## MVC architecture

**小規模教育管理システム**において、PHPとLaravel使ってMVC　architectureを実施しておりました。具体的にはModelでEloquent(エロクアント) ORM、ViewでBlade templates、ControllerでController classesを活用して、monolithic systemを構築いたしました。

**中規模銀行向けサビスシステム**

# 会社要件

## MVC:View: React,

Model: Mongoose, Sequelize(シークゥェライズ)ー＞Data design, handling

Controller:Express route controllers ー＞Handles logic between routes and models

## 顧客管理（CRM機能）

Salesforce、Zoho、HubSpot

最新は、zendeskサービス（SDK api）を利用し、顧客からのリクエスト・チケトを管理する顧客管理システム開発に関わりました。