



# DXを加速させる "Ruby人材育成の鍵"

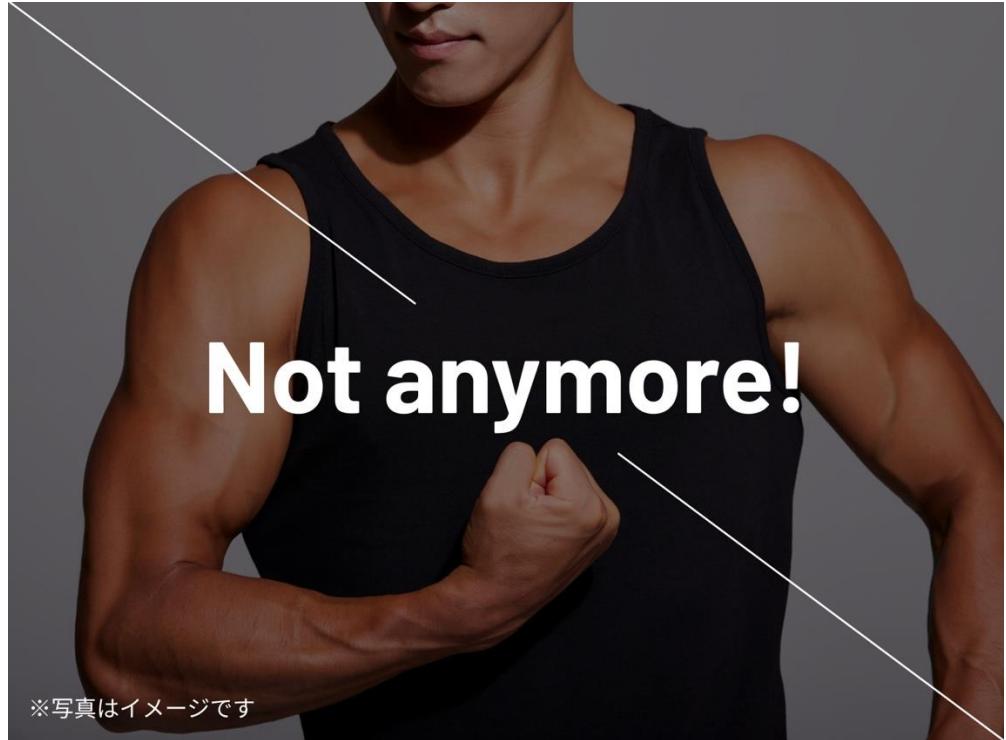
新卒9割越えの開発組織を支える、育成ノウハウの裏側を徹底解説

---

RIZAP TECHNOLOGIES

Nov.6-7, 2025

# DX推進の決意と現実の壁



- DXを本格推進することを決意
- しかしテックカンパニーとしての  
知名度が低い
- 即戦力となる人材の確保に苦戦

▼

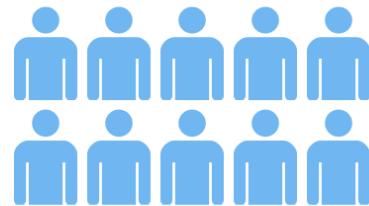
ポテンシャル人材を採用し、  
育成で伸ばす方針へ

# わずか3年での急成長



24  
人

2022年7月



140  
人

2025年9月1日時点  
(※パート社員含む)

- ポテンシャル採用と育成を軸に組織が拡大
- 新卒エンジニアを中心に、  
若手が急速に成長しながら活躍
- Rubyのバックエンド開発エンジニアは、  
現在、9割以上が新卒採用

# 急成長を支えたのは"独自の育成メソッド"



ポテンシャル人材を、  
実務で活躍するエンジニアへと導く、  
育成プログラムの全容をお伝えします！

1

# 自己紹介



RIZAPテクノロジーズ株式会社

バックエンドエンジニア

梅田 智大

Umeda Tomohiro

もともとは商品開発を担当、ITとは無縁の世界に。

プログラミング未経験、PC作業はメールとブラウザだけ。

そんな私が育成プログラムを通じてエンジニアへ転身。

そしてわずか3年で、RubyWorld Conferenceや

Kaigi on Railsに登壇するまでに成長しました！

15分講演-2

DXを加速させる“Ruby人材育成の鍵” — 新卒9割超の開発組織を支える、育成ノウハウの裏側を徹底解説



梅田 智大  
RIZAPテクノロジーズ株式会社

RubyWorld Conference



Hall Blue

Range on Rails — 「多重範囲型」という新たな選択肢が、複雑ロジックを劇的にシンプルにしたワケ

梅田智大

Kaigi on Rails

2

# 育成プログラムの流れ

# 1,000時間で幅広い領域を網羅



3

ここからが本題！  
**育成の鍵、その裏側に迫る**

皆さんなら、この1行の教育に何分かけますか？

```
# config/routes.rb
resources :users
```

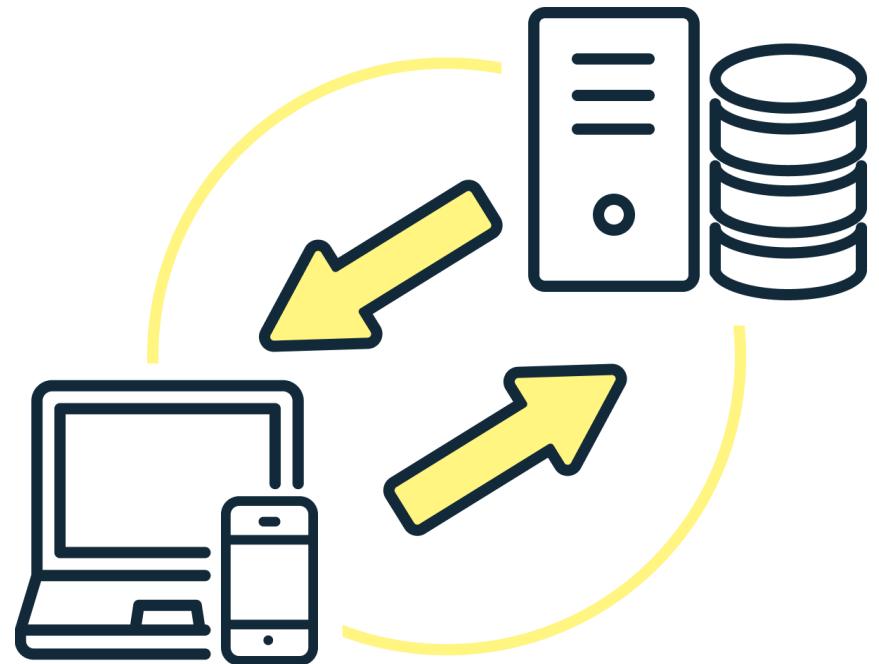
RIZAPテクノロジーズでは、  
この1行の教育に16時間を費やします

# まずは機能的な仕組みを理解

```
# resources :users は、  
# 下記8行の省略形であることを理解する  
get "/users", to: "users#index"  
get "/users/:id", to: "users#show"  
get "/users/new", to: "users#new"  
post "/users", to: "users#create"  
get "/users/:id/edit" to: "users#edit"  
patch "/users/:id" to: "users#update"  
put "/users/:id" to: "users#update"  
delete "/users/:id" to: "users#destroy"
```

まず理解すべきは、  
どのリクエストが、  
どのコントローラーの  
どのアクションに割り当てられるのか  
という、  
ルーティングの機能的な仕組みです

# 次に理解すべきはWebの仕組み



- Webページはどのように表示されるのか？
- HTTP通信とは何か？

こうした「Webを支える技術」を理解することで、ルーティングの本質が見えてくる

# なぜ「resources」なのか？

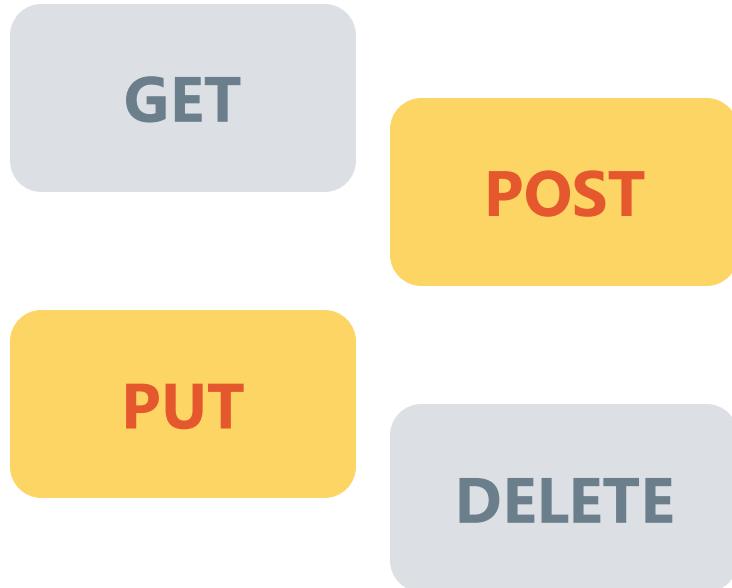


- さらに問う
- なぜDHHはこの機能を  
「resources」と名付けたのか？



その背景にはRailsの根底にある、  
**REST**という思想がある

# RESTとは何か？

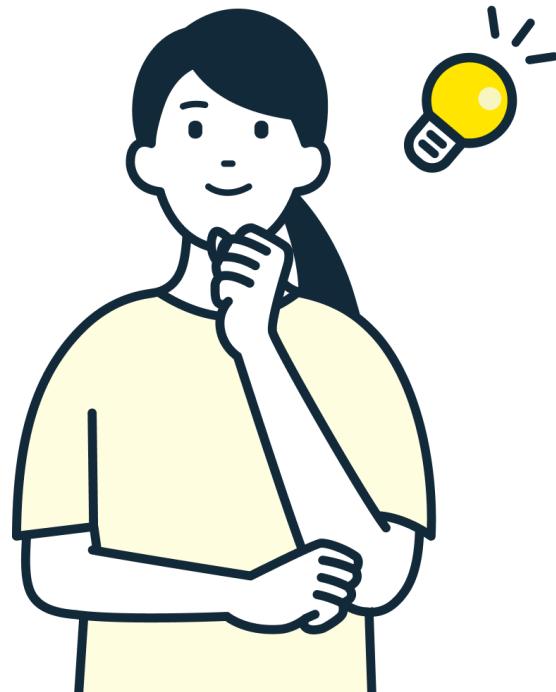


- 全ての情報はリソース
- リソースは識別子（URI）を持つ
- Webがやっていることはリソースに対する操作であり、**取得・作成・更新・削除** の4つだけ
- これを表現するのが、HTTPメソッド



ここまで理解すると、「**resources**」という名前の必然性が見えてくる

# 名前重要



- ここで押さえておきたい、  
プログラマが知るべきことのひとつ
- 作者がどんな意図で命名したのかを探ることで、たった1つのメソッドにも理解すべき真髓が見えてくる

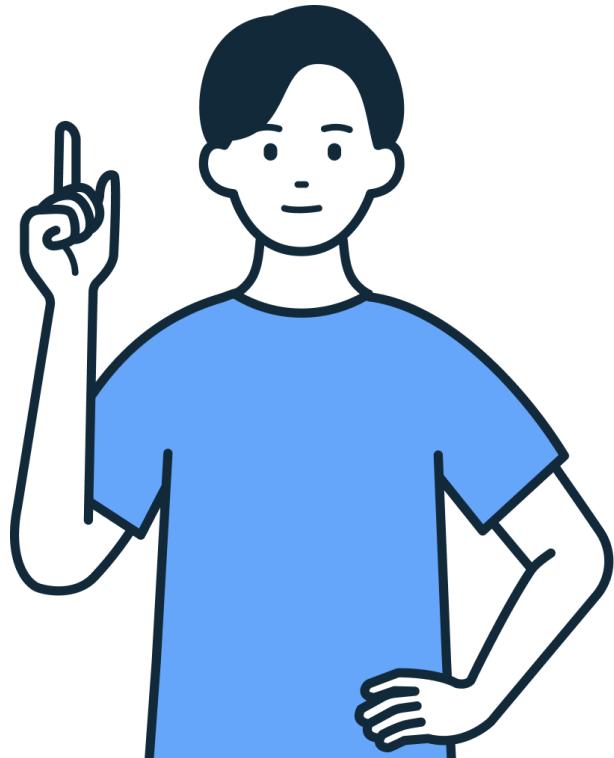
## AI時代だからこそ、1行・1メソッドの理解にこだわり抜く

- たった1行のコードも「理解したつもり」で終わらない
- 背景や思想まで掘り下げ、名前の意味や仕組みに至るまで  
**徹底的に理解する**
- この積み重ねが、現場で自走できるエンジニアを育てる土台  
となる

4

このカリキュラムには  
「台本」がある

# カリキュラムを完全内製



- 各章ごとに「何を」「どこまで理解するか」を明文化
- メンターはこのカリキュラムを基に指導
- 教え方のばらつきをなくし、学びを標準化

# 研修は"台本通り"に進んでいく

## パスワード認証を実装する研修

メンター 「パスワード認証を自力で実装してください。」

受講生 「えっ、フレームワークで簡単にできるのに、なぜわざわざ？」

メンター 「いいから、まずは自力でやってみましょう。」

受講生 「……なるほど、こんなに複雑で危険なんですね。」

メンター 「そうです。認証・認可は難しく、自作すべきではない。  
それが大事な学びです。そこで、おすすめの本があります。  
徳丸 浩さん著書の『安全なWebアプリケーションの作り方』です。」

# "決め台詞"までカリキュラム化する

## routes.rb

- そもそもルーティングとは何か

```
1 resources :users
```

- この1行で起こること、役割を説明できるようにすること
  - どんなルーティングが生成されるか
  - resourcesを使用しない場合にどうなるかをコメントアウトでコード内に記載してもらう

例:

```
1 # get 'users', to: 'users#index'  
2 # post 'users/create', to: 'users#create'  
3 # get 'users/new', to: 'users#new'  
4 # get 'users/:id/edit', to: 'users#edit'  
5 # get 'users/:id', to: 'users#show'  
6 # patch 'users/:id', to: 'users#update'  
7 # put 'users/:id', to: 'users#update'  
8 # delete 'users/:id', to: 'users#destroy'
```

- URLパスとHTTPメソッドの対応関係を理解すること
  - そもそもHTTPメソッドとは何か
- なぜ、DHHがresourcesというメソッド名にしたのか
  - RESTとはなにか
  - RESTの概念に関して次の記事を一読する → [yohei-y:weblog: REST 入門](#)
  - 全ての情報はリソースであり、Webの仕組みを理解することは、リソースに対する操作を理解するということ

● ここでMatzの格言である『名前重要』を紹介する！

■ [名前重要 | プログラマが知るべき97のこと](#)

■ 先ほど紹介した「resources」の解説

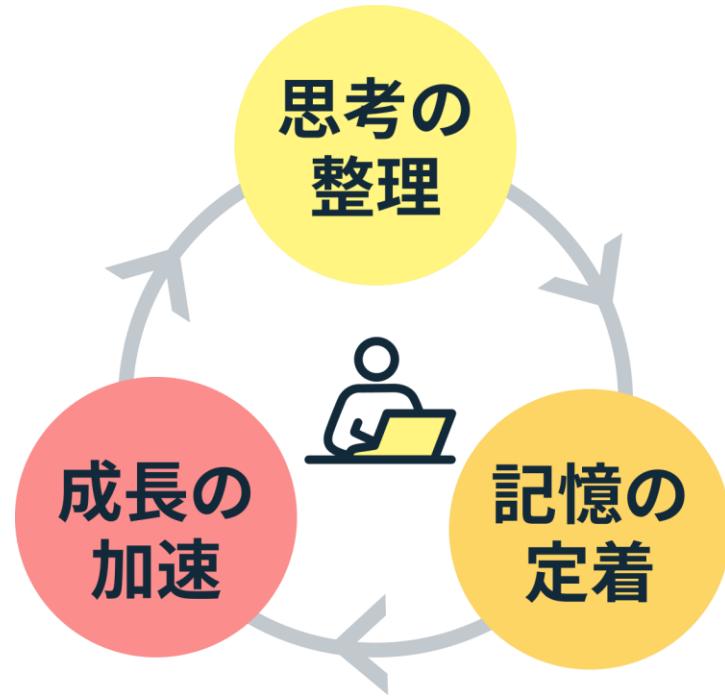
も、事細かにカリキュラム化

■ どこで何を話すかまで決められた  
「台本」が存在

5

# 成長の秘訣 「日報」

# 令和の時代に「日報」!?



- 毎日30分以上かけて日報を作成
- 何をやったか、気づき、分からぬこと等を、  
言語化することでアウトプット力を磨く
- さらに思考を整理し文章に起こすことが、  
エンジニアの基礎スキルへ直結

# 1日分の日報は約3,000文字

5月29日(木)

## やったこと

- Lesson11(コントローラーテスト、PR作成、その他修正)

## 学び

### <%= paginate @users %>の挙動

paginate は、Kaminariのヘルパーメソッドで「ページ送りのリンク (UI)」をHTMLとして表示してくれるもの

#### 具体的には

Kaminariが @users オブジェクトから以下の情報を読み取ってHTMLを生成している

- 今何ページ目か → @users.current\_page
- 全部で何ページあるか → @users.total\_pages
- 次のページはあるか → @users.next\_page
- 前のページはあるか → @users.prev\_page

#### なぜそれが可能なのか

@users = User.page(params[:page]) することで、@users は普通の配列ではなく、Kaminariが拡張した「ページネーション可能なオブジェクト (ActiveRecord::Relation + Kaminariの機能)」になっている。

そしてこの特別なオブジェクトには次のようなメソッドが使える。

```
1 @users.total_pages      # 全ページ数
2 @users.current_page    # 今のページ番号
3 @users.next_page        # 次のページ番号
4 @users.prev_page        # 前のページ番号
```

そして paginate @users は、これらの情報をもとに「HTMLのリンクリスト (ページネーションUI)」を自動で組み立てて表示してくれている。

## SQLインジェクションについて

[Rails セキュリティガイド - Railsガイド](#)

### SQLインジェクションとは...

外部からのユーザー入力が、意図しないSQL構文として解釈・実行されてしまう脆弱性のこと。

#### 例：危険な書き方

```
1 User.where("name = '#{params[:name]}'"')
```

params[:name] に次のような値が渡されたとする

```
1 ' OR 1=1 --
```

→ 生成されるSQL

```
1 SELECT * FROM users WHERE name = '' OR 1=1 --'
```

- name = '' ... 空文字を検索 (正常)
- OR 1=1 ... 常に true の条件 (すべてのレコードが一致!)
- ... コメント扱いにして後の条件を無効化

#### ✗ つまり、この WHERE 条件は

「名前が空文字列のユーザー」または「1=1 (常に真)」に該当するユーザーを全て出して」  
→ 全てのユーザーが該当してしまう

#### 特に危険なケース：ログイン処理

```
1 User.where("email = '#{params[:email]}' AND password = '#{par...
```

入力された値

```
1 params[:email] = ' OR 1=1 --
2 params[:password] = ''
```

→ 生成されるSQL

```
1 SELECT * FROM users WHERE email = '' OR 1=1 --' AND password = ''
```

- OR 1=1 によって認証条件が常に true

- 実際にパスワードのチェックが無効化される

→ 不正ログイン成立してしまう

### Railsの対策：構文と値を分離 (Prepared Statement)

#### 基本方針: 構文と値を分離する

##### 1. プリペアドステートメント (Prepared Statement)

- Railsはまず、SQL構文だけをテンプレートとして作る

```
1 SELECT * FROM users WHERE name = ?
```

このとき、? は単なる「値をあとから埋め込むためのプレースホルダ」この構文を、データベースに「構文部分だけ」送る。

##### 2. ユーザー入力を「値」として別送する (バインド変数)

- 次に、Railsは params[:name] の値 (例: ' OR 1=1 --') を、構文とは無関係な「ただのデータ」としてデータベースに送る。

→ たとえばこのようなユーザー入力

```
1 params[:name] = '' OR 1=1 --'
```

これがバインドされると、最終的に実行されるSQLは

```
1 SELECT * FROM users WHERE name = '\' OR 1=1 --'
```

- ? に値がバインドされるとき、構文にはならず「クオートされた文字列」として扱われる
- OR 1=1 は文字列の一部であり、構文ではない
- 結果としてインジェクションが成立しない

##### 3. LIKEをつかうときの注意点

```
1 User.where("name LIKE ? ", "%#{params[:name]}%")
```

この書き方は一見安全に見えるが、実は params[:name] に含まれる特殊文字 (%, \_, \) によって、意図しない検索結果になってしまうことがある。

たとえば次のような入力

```
1 params[:name] = "%"
```

→ 生成されるSQLは

```
1 SELECT * FROM users WHERE name LIKE '%%%'
```

→ % は「任意の文字列」のワイルドカードとして解釈され、全件ヒットしてしまう可能性がある

#### 対策： sanitize\_sql\_like を使って特殊文字をエスケープする

Railsには、 ActiveRecord::Base.sanitize\_sql\_like というヘルパーメソッドが用意されており、これを使うと % や \_ などを「ただの文字列」として扱えるようになる。

実際のコードでは、以下のように書くことで安全な LIKE 検索が可能になる。

```
1 User.where("name LIKE ? ", "%#{User.sanitize_sql_like(params[:...
```

たとえば...

```
1 params[:name] = "%"
```

とユーザーが入力した場合、上記のコードによりエスケープされたSQLは

```
1 SELECT * FROM users WHERE name LIKE '%\%%'
```

となり、「名前に % という文字を含むユーザー」だけを対象に検索する。

Railsが自動的に \% のようにエスケープしてくれることで、% をワイルドカードではなくリテラル (文字としての %) として扱うことができる。

#### エスケープされる文字と理由

文字	意味	なぜエスケープが必要？
%	任意の文字列にマッチ	意図せず全件一致になることがある
_	任意の1文字にマッチ	意図しない部分一致が起きることがある
\	エスケープ文字	上記の % や _ を文字列にする際に必要

# 日報を書くまで帰れません



日報を書き終えるまで  
打刻できないルールとしています

何十人のも研修を通じて、  
日報を丁寧に書く人ほど、成長が早いことを確信！

6

# 育成プログラムのまとめ

## "急成長"の裏にあるのは、地道な基礎の積み重ね

- 1行の理解にこだわり抜き、日々、ひたむきに日報を書き続ける
- これを研修全体を通して徹底的にやり続けるからこそ、研修時間はトータルで約1,000時間に及ぶ
- 内定者インターンとして入社前から研修に取り組むことで、入社後すぐに実務へ

その結果、ポテンシャル採用中心でも  
生産性が高く、スピード感を持った開発が可能に

7

ここで終わりじゃない。  
入社後も成長を後押しする

いろいろありますが、時間の都合で  
本日は1つだけ紹介をさせていただきます・・・

# 急成長を支える仕組みのひとつが「分報」



これから店舗検索APIの  
実装に入れます！



店舗温度管理システムの  
要件確認MTGに参加する



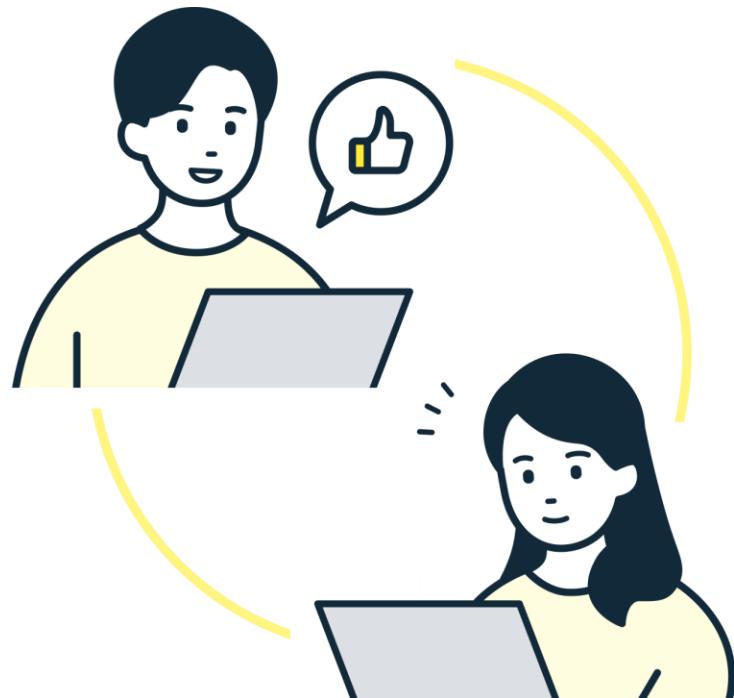
お腹すいた～  
昼休憩行ってきます！



自分もお腹すいてきたな…  
昼ごはん牛丼にしようかな

- タスク切り替え時に、  
「これやります」「終わりました」  
と随時投稿
- 雑談もOKで、気軽につぶやける  
チャンネル

# なぜ分報をやるのか？



- 細かい業務報告だけが目的ではない
- 日常的なコミュニケーションが、  
**「心理的安全性」**をつくる
- 悩んだときや躓いたときに、  
**気軽に相談できる環境**に
- お互いにフォローし合う**関係性**が築ける

## コミュニケーションが、成長を加速させる

- RIZAPテクノロジーズは100%フルリモートだが、どこよりもコミュニケーションにこだわっている
- 日常的なコミュニケーションが、チームの関係性を深め、仕事のしやすい環境をつくる
- お互いにフォローし合うからこそ、生産性が高く、開発はスピーディーに進む
- だからこそ、RIZAPテクノロジーズは急成長を遂げている！

# RIZAPテクノロジーズで、ともに成長を。

RIZAPテクノロジーズには、急成長を続ける仕組みがあります。

その背景には、**お互いを寄り添う文化**が根づいています。

ここで**共に成長を遂げたい**——