実務で学んだあれこれ

実務に取り掛かるにあたっての心構え

1人で悩みすぎない

- 実務は基本的に期日があるので、悩みすぎてスケジュールが遅れるのが1番迷惑となります。
    - 報告は細かくしていき、また遅れているときは遅れている原因も一緒に報告できたら〇
    - エラーで煮詰まった時や実装が全くわからない時は、一旦休憩しましょう！！
    - 辛い時は溜め込まずに誰かに相談しよう！！
    - 仕様については考えてもわからないのでどんどん質問しましょう

急いでる時こそ慌てずに確認を

- 経験上焦って実装や連絡しても余計に遠回りになる可能性の方が高いです
- 中途半端な実装をすると逆に先方の方の信用を落としかねないので、気をつけましょう

人に質問するときはわかりやすく簡潔に

- 先輩たちの時間を奪わないように分かりやすく！！
    - ポイントとしては結論から話す事や太字・句読点・改行を使って文字が詰め詰めにならないように！！
    - エラーの場合はエラーログを調べてから質問をする

先方へのPRを作成するときこれだけは確認しよう!!

- コミットメッセージは綺麗で適切か？
- そもそも仕様をもれなく満たしているか？
- ブランチ名に問題はないか？
- PRを作成する先を間違ってはいないか？
- ソースコードの書き方(詳細下記)に記載のあるものは全部問題ないか？

チケットに取り掛かる前に

- 【新人プログラマ応援】開発タスクをアサインされたらどういう手順で進めるべきか
- Railsの案件にジョインしたときに、何を足掛かりにすべきか

便利なメソッド等のまとめ(Ruby,Rails)

モデル・DB（リレーション）周りはここら辺押さえておくと良さそう

- scope
- Railsで別名の外部キーを設定する方法⇨詳細
- polymorphic⇨詳細
- inverse_of
- accepts_nested_attributes_for
- delegateの使い方⇨詳細
- バリデーションまとめ
    - バリデーションでifを使って制御する方法

SQLを扱うならここら辺押さえておくと良さそう

- 基本的にRailsに既存であるメソッドでSQLを組むことを推奨
    - どうしても厳しい場合や複雑なSQLを組むときは、 プレースホルダー等を用いて動的に変更できるようなSQLを書いていく
- SQLの基本を覚える
- N+1対策はここら辺を使えば問題ない（includes・preload・eager_load）
- ActiveRecordのjoinsとpreloadとincludesとeager_loadの違い
- joins⇒詳細
- ActiveRecordにおけるGROUP BYの使い方
- orderで関連テーブルのカラムで並び替え
- where⇒詳細
- 論理削除したデータを扱う
- 関連テーブルの条件をON句に書きたい！
- present?とexists?のSQLでデータ取得する際のパフォーマンスの違い
- SQLをRailsで扱う時の色々な便利なもの
- find_by_sqlでRailsから生SQLクエリを直接実行する
- GROUP BY

その他の便利なメソッド

- pluck
- all?

- map
    - mapは体感(&:)の書き方が多い
- Railsで変数の中身が空か確認する空チェックメソッド4種
- to_sql→ActiveRecordのオブジェクトに対して使うことでSQLを見ることができる
    - 主にデバックで使用
- inspect→ActiveRecordのオブジェクトに対して使うことでオブジェクトの中身を確認できる
    - 主にデバックで使用する・めっちゃおすすめ 例：`User.find(1).inspect`
- 配列の要素と何番目にあるかを一緒に取り出す_each_with_indexメソッド
- ぼっち演算子でNilエラーを回避する(割と使う)
- merge
- html_safe
- eachは空の配列[]でも対応可能だから、条件分岐等で場合分けは不要

その他実務で見たあれこれ

- バッチの実行方法などで使う(rails-runner)
- rails-runnerに引数を持たせる方法
- CanCanCanで特定のユーザーがアクセスできるリソースを制限する
- decoratorの使い方
    - （例）Viewに関係するメソッドでかつModelに関係するメソッドの記述 ユーザーの名前とかのフォーマットetc
- Serializers※重要JSONとしてデータを返す時によく使う
- attr_accessor
    - Railsでのマイグレーションファイルを作成する際にも自動で使われているはず `User.name`で値を取得したり書き換えたりできるのは`attr_accessor`のおかげ
- I18nで日本語化まとめ
- i18nで引数を渡して動的にする
- ラッパークラスの概念
- 早期returnを使って無駄な処理をしない※記事はJSだけどRubyでもある
- breakで処理を途中で抜ける
- RubyのModuleの使い方

    - モジュール内で名前空間やインスタンスの生成で`initialize`をよく使う
- 名前空間
    - 同じ名前であることによるコンフリクト(衝突)が起きないように 「どこに属するメソッド、変数ですか？」というのを明記する仕組み 例：`puts Math::PI`でMathモジュールの定数PIを呼び出す
- initialize
    - 個人的にはモジュールとセットで使う印象 また、インスタンスを作成する時に使う印象が強い
- class << self
- concern
- ActiveSupport::Concern
- include ※モジュール内で使うことが多い
- includeとextendsの違い
- Transaction
- Redis
- sidekiq
- 例外処理の書き方
- Rubyで独自例外を定義するときはStandardErrorを継承する
- Cookieとセッションをちゃんと理解する
- Web Storage
- update_attributes
- freeze
- private と protected の使い方まとめ
- alias
- respond_to
- インスタンスメソッドとクラスメソッドの使い分け

プログラミングの考え方

Railsの設計

- 中／大規模開発のためのRails設計パターン

ソースコードの書き方

- 変数名は既存と命名規則等合わせる
- unlessを多用しないようにする
    - unlessは読みにくいです
- 条件分岐をネストしすぎない
- 自分の書いたコードが影響範囲はないかを確認する
    - 既存の機能が動かなくなることがたまにあります
- 配列等の変数名は複数形を使う
- boolean値を変数として保存する際は、変数名の先頭にis_をつける
- N+1が起こらないようにする
- 同じようなソースコードを何個も書いていないか(重複してる内容はメソッド化する)
- 基本的に1つのファイルに責務は1つにする
- ディレクトリを作成する際は関連する処理をまとめるようにすると良い(errors・scope・jobs等)
- 基本的に既存のコードに合わせる形で実装しましょう

ソースコードの読み方

- メソッドジャンプを使用して処理の流れを1つづつ理解
- Rails.logger.debugを使って処理の流れとは格納されている値を確認していく
- あるページに対応するファイルを探すときは、日本語・特別な処理など、なるべく固有なもので全体検索かける

エラーの対処法

- エラーが出たときはバックエンドorフロントエンドどちらでエラーが出ているかを確認する
- フロント側ならコンソールを確認・バックエンド側ならログを確認
- ログを確認してどこまで処理が通って、どこでコケているのか確認する
- APIを使ってフロントと繋いでいる場合はディベロッパーツールのネットワークタブも確認してAPIの返してる値や、エラー出ていないかチェックする
- Rails.logger.debugを使って格納されている値を確認していく
- 返り値を意識すると尚いい
    - ActiveRecordのオブジェクトが返される等、どのような値が返されているのかを確認
- エラーに限らずだが、処理の内容を1行づつ理解することが大事
    - エラー周りは特にソースコードを日本語に翻訳できるくらい理解しよう

最後に

どんなに辛くても諦めずに、決して思考を停止しないでください