

メルカリITPチームが開発速度とソフトウェア品質を高めるためにやっていること

Hiroaki KARASAWA 🐯 @karszawa

Mercari Engineer's meetup for students vol.3 (2019/11/25)

自己紹介

-  Hiroaki KARASAWA
-  @karszawa
-  Mercari, **Web Frontend Engineer** (2019新卒)
-  React, TypeScript, GraphQL, ReactNative(Expo)

学生時代

- 明石高専 → 東京大学EEIC（編入）→ 学士（工学）
- メルカリ / LINE / クックパッド など5社でインターン
- JPHacksなどのハッカソンに参加
- 卒業研究で管理栄養士とアスリート向けのサービスを開発

仕事

- 2019年4月 株式会社メルカリに新卒入社
- 同6月 **Logistics and Transaction Team**のフロントエンドエンジニア

今日話すこと

1. 自己紹介
2. ITPチームについて ← イマココ
3. ITPのコードベース上での工夫
4. ITPの開発体制に関する工夫
5. まとめ

Web Frontend teams in Mercari

mercari 新規会員 ログイン

何をお探しですか?

カテゴリーから探す ブランドから探す

キャンプ用品 大特集!! Let's go camping!

人気のカテゴリ

レディース メンズ 家電・スマホ・カメ

レディース新着アイテム もっと見る >

¥3,100 ¥888 ¥1,000

らくらくメルカリ便で発送する

商品が購入され支払いされました。発送してください。らくらくメルカリ便では、コンビニ・宅配便口ッカーからの発送、ヤマト営業所への持ち込み、またはご自宅への集荷依頼を選択できます。

スマホが売れた後の流れ >
あんしんスマホサポートとは? >
出品者のよくある質問 >
らくらくメルカリ便とは? >

スマホ発送前に下記項目をご確認ください。

[スマホ発送前の最終確認をしました](#)

[コンビニから発送](#)

[ヤマトの営業所へ持ち込んで発送](#)

※あんしんスマホサポートは配送方法の変更ができません。配送のお問い合わせはアプリよりお願いします

Confidential

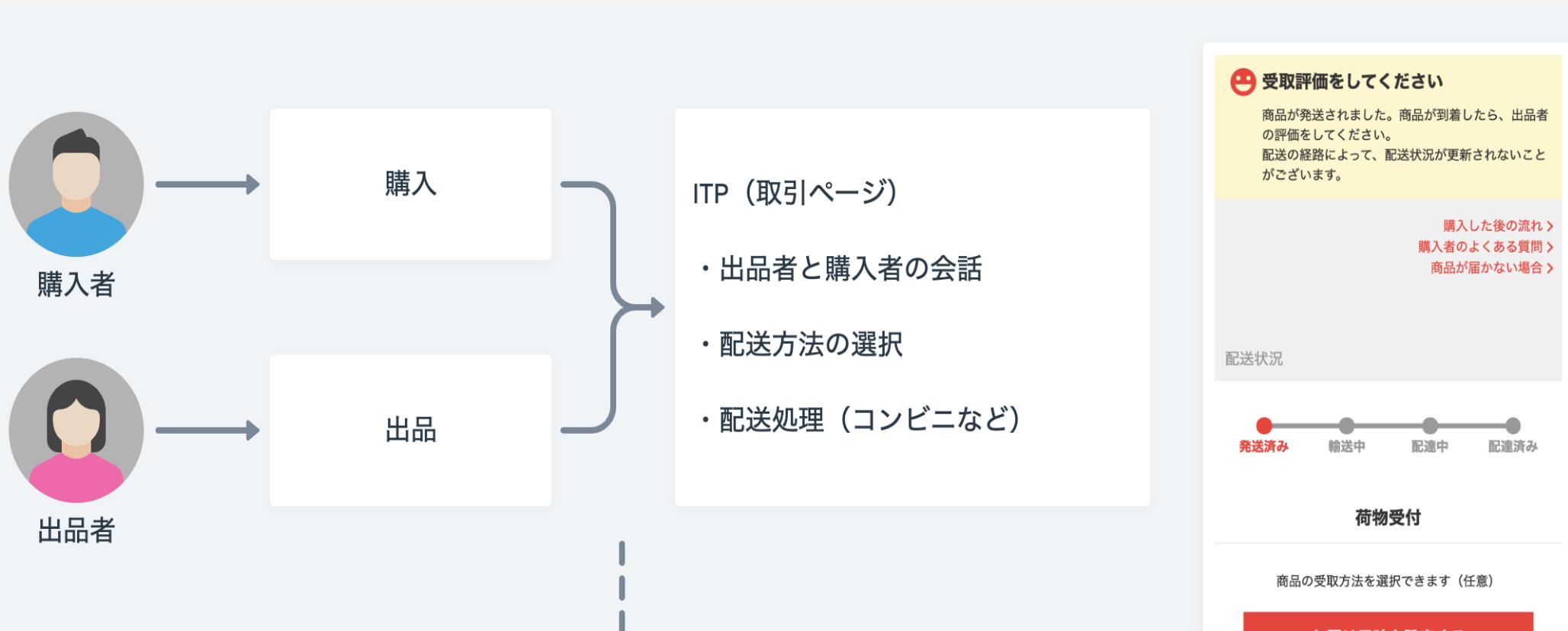
WebアプリケーションとしてのITP

- **React / Redux / Redux-Saga** による SPA
- WebView上でのみ表示される特殊なアプリケーション
 - 開発はシミュレータではなくPCのブラウザで

(最終確認はシミュレータだが、シミュレータは操作しにくくデバッグしにくいので使わない)

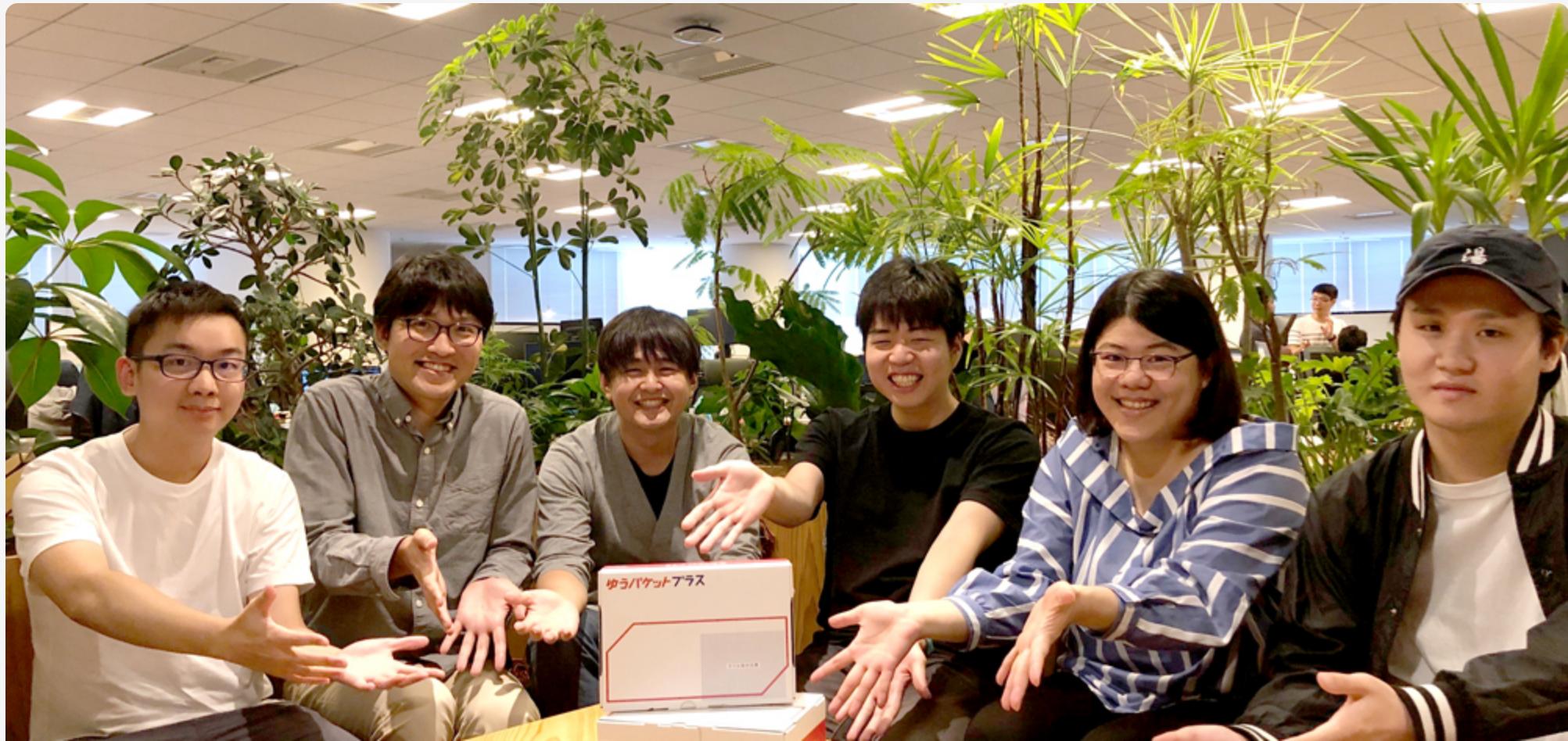
ITPの主な機能

- 購入者と出品者がテキストで会話できる 💬
- 出品者が出品方法を選択して発送できる 📦
- ヤマトや日本郵政のAPIとのやり取りが多い 🐾 💕 🎉



ITPチームの最近の活動

- らくらくメルカリ便で「受け取り場所の変更」ができるようになった
- ゆうゆうメルカリ便で「ゆうパケットプラス」が選択できるようになった



今日のトピック

開発速度と品質を両立しながら開発を行う
ためにITPチームが何を行っているか

今日のまとめ

- 巨大なビジネスロジックの制御方法（何を作っているか）
- スクラムの実践方法（どうやって作っているか）

ITPは単一のアプリケーションだが巨大で複雑なビジネスロジックを持っている

ITP（取引画面）は次の条件で振る舞いを変える

- **購入者** か **出品者**
- **取引状態** (WaitShipping / WaitReview / Done, ...)
- **発送方法** (らくらくメルカリ便, ゆうゆうメルカリ便, 大型らくらくメルカリ便, ...)
- **商品カテゴリ** (あんしんスマホサポート, あんしん自動車保証, ...)

ITP（取引画面）の例

受取評価をしてください

商品が発送されました。商品が到着したら、出品者の評価をしてください。
配送の経路によって、配送状況が更新されないことがあります。

購入した後の流れ >
購入者のよくある質問 >
商品が届かない場合 >

配送状況

発送済み 輸送中 配達中 配達済み

荷物受付

商品の受取方法を選択できます（任意）

お届け日時を設定する

受取場所を変更する
※変更したい場合アプリのアップデートが必要です

送り状番号 768530004863

※ヤマト運輸HPより荷物お問い合わせ状況を確認いただけます
※送り状番号が正しく表示されない場合はしばらく待ってから

支払いをしてください

2月7日(水) 23時59分までに必ずお支払ください。

購入した後の流れ >
あんしんスマホサポートとは？>

支払い情報

商品代金	¥88,998
ポイント利用	P0
売上金利用	¥0
メルペイ残高利用	¥1
コンビニ/ATM手数料	¥0
支払い金額	¥88,998

支払い方法 支払い方法を確認する >

支払いをするコンビニ/ATMを選択してください

セブン-イレブン ローソン
 ファミリーマート ミニストップ
 デイリーヤマザキ セイコーマート
 銀行ATM

支払い方法を設定する

自動車と書類を受け取ってください

自動車と書類が発送されました。自動車と書類を受け取ったら名義変更の手続きを行ってください。
名義変更手続き完了後、新しい車検証の交付年月日を入力し、最後に出品者の評価をしてください。

自動車取引の全体の流れガイド >
購入者のよくある質問 >

自動車と書類を名義変更手続きと評価をする 出品者の評価を待つ 取引完了

名義変更手続きと評価

1. 名義変更手続きをしましょう

受け取った自動車と名義変更に必要な書類を用意し、管轄の登録窓口で手続きしてください。
※手続きは必須です。手続きを行わなかった場合、法律の罰金対象になります。速やかにお手続きください

[名義変更手続きについて詳しくみる](#)

2. 車検証の交付年月日を入力しましょう

新しく発行された車検証に記載されている交付年月日を入力してください。

交付年月日

出品者情報

iov >

The Guardian
Mock API does not know user id, so if you send a message the client will probably fail with exception
① 2月5日 16:35

A peasant
I know, but i will still do it! 🚦
① 2月5日 16:39

なにか分からないことがあれば質問してみましょう。

取引メッセージを送る

※取引メッセージの内容は、必要に応じて事務局で確認しています
※お困りの際はよくある質問をご確認ください

取引情報

事務局メモ 事務局メモ 事務局メモ

商品代金 **¥88,998** >

メッセージ画面

ポイント

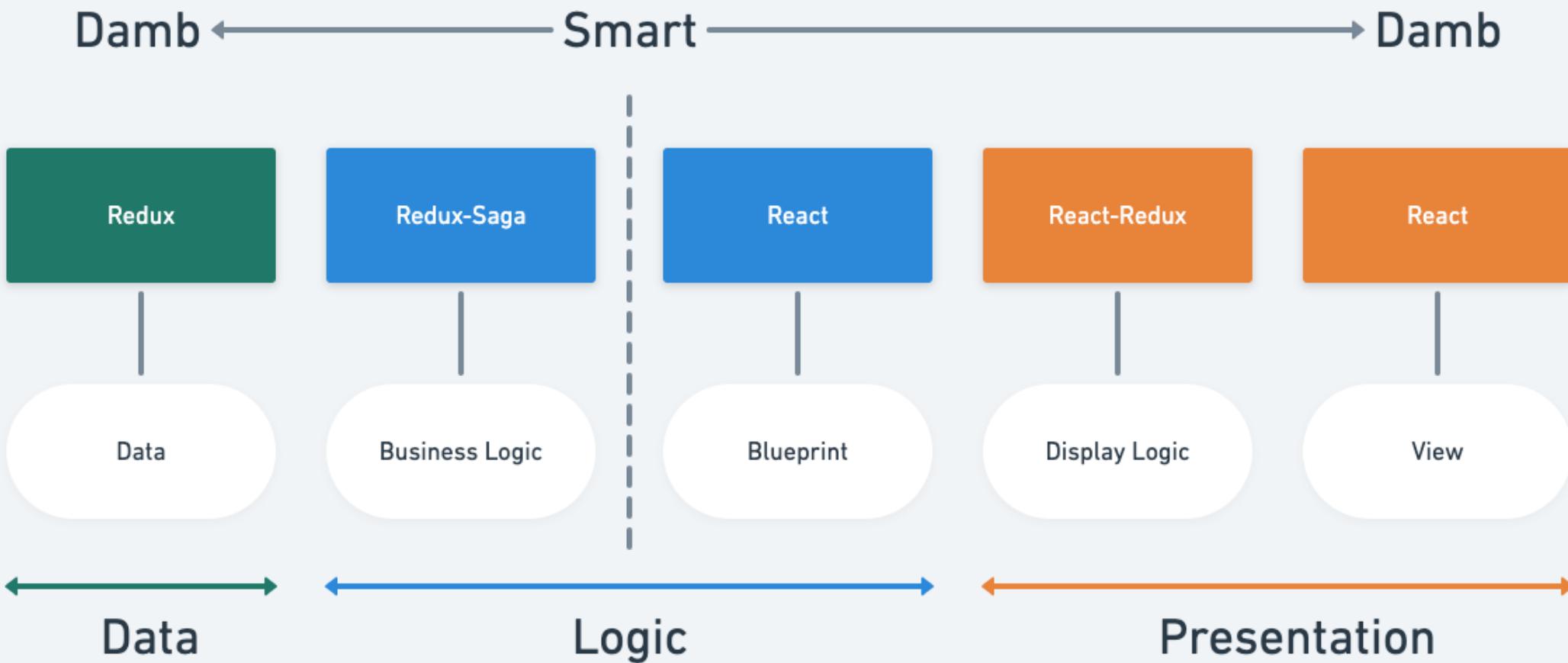
- ビジネスロジックは画面ごとに異なる
- 共有できるコンポーネントが多くある
- ☞ ビジネスロジックとUIがきれいに分割できるかどうかが重要

ビジネスロジックとUIがきれいに分割できると

- 再利用がしやすい **(開発速度の向上)**
- テストしやすく変更にも強くなる **(品質の向上)**

具体的にはどうやって分ける？

データ層・ロジック層・プレゼンテーション層などのレイヤーに分離



Presentation Layer (React)

- マークアップやスタイルリング
- 責務が明確なのでデザインシステムに移行しやすい（移行したとは言っていない）

```
const CommentBoxTypeOne: React.FC<Props> = ({  
  messages,  
  handleMessageChange,  
  handleSubmit,  
  shouldDisableSendButton,  
) => (  
  <div class="block">  
    <textarea  
      value={message}  
      onChange={handleMessageChange}>  
    />  
    {!shouldDisableSendButton && <PrimaryButton onClick={handleSubmit} />}  
  </div>  
) ;
```

Presentaion Layer (React-Redux)

- コンポーネント・データを選択し紐付ける

```
const CommentBox: React.FC<Props> = ({  
  shouldDisableSentButton,  
  messages,  
) => {  
  if (someBranchingLogic) {  
    return <CommentBoxTypeOne />;  
  }  
  return <CommentBoxTypeTwo />;  
}  
  
const mapStateToProps = ({ transaction }: Store) => ({  
  // ...some display logic  
  shouldDisableSentButton,  
  messages: someCalculation(transaction.messages),  
});;  
  
const mapDispatchToProps = () => ({  
  handleMessageChange: updateMessage  
});
```

Business Logic Layer (Redux-Saga)

- 振る舞いを定義する層

例：メッセージを送信するときのビジネスロジック

1. コメントボックスに入力されたメッセージを取り出す
2. 本当に送信するかの確認のためにダイアログを表示する
3. 確認ボタンが押されたらメッセージをAPIに送信する
4. APIにエラーが発生したらエラーメッセージを表示する
5. 送信に成功したらメッセージのリストを更新する

Business Logic layer (Redux-Saga)

- Sagaを起動

```
function createRootSaga(api: APIClient) {
  return function *rootSaga() {
    yield fork(function*() {
      while(true) {
        const action = yield take(POST_CHAT_MESSAGE);
        yield call(postMessageSaga, apiClient, action);
      }
    });
  }
}
```

Business Logic layer (Redux-Saga)

- Saga本体

```
export function* postMessageSaga(api: APIClient, action: ReduxAction<{ message: string }>) {
  // 1. コメントボックスに入力されたメッセージを取り出す
  const { message } = action.payload;

  // 2. 本当に送信するかの確認のためにダイアログを表示する
  const confirmParams: ConfirmParams = {
    message: "本当にメッセージを送信しますか？",
    confirmText: "はい",
    dismissText: "いいえ"
  };
  if (!(yield call(nativeConfirm, confirmParams))) {
    return;
  }

  const evidenceId = yield select(transactionEvidenceSelector);

  try {
    // 3. 確認ボタンが押されたらメッセージをAPIに送信する
    yield call([api, api.postTransactionmessage], { evidenceId, message });
  } catch(e) {
    console.error(`Error occurred while sending message: ${e.message}`);
  }
}
```

Data Layer (Redux Store)

- Reducerにロジックを書くことは可能
- 絶対にロジックを持たせない強い意思が重要

```
function comment(state: string = "", action: ReduxAction<Comment>) {  
  switch (action.type) {  
    case INPUT_COMMENT:  
      return action.payload;  
    case CLEAR_COMMENT:  
      return "";  
    default:  
      return state;  
  }  
}
```

- 以上がフロントエンドアプリケーションの見通しを良くする設計論
- とはいえどこに何を書くべきか混乱することもある

ディレクトリベースでレイヤを分割して分割の間違いが起こりにくくしている

```
src
└── common
    └── components → Common Presentation Layer (Dumb Components)
        ├── Button.tsx
        ├── Panel.tsx
        └── Row.tsx
    └── transactions → 取引方法ごとにディレクトリを分割
        ├── car
        ├── oogata
        └── rakuraku
            ├── Buyer.tsx
            ├── Seller.tsx
            └── components → Specific Presentation Layer (Dumb Components)
                ├── ProductDetail.tsx
                └── TransactionInfo.tsx
        └── containers → Presentation Layer (Smart Containers)
            ├── Done.tsx
            ├── WaitReview.tsx
            └── WaitShipping.tsx
    └── stores
        ├── reducers.ts → Data Layer
        └── sagas.ts → Logic Layer
```

- 取引画面ごとのコンポーネントなどドメイン知識も結構必要
- 知識の共有には組織体制としての工夫も必要

1. 自己紹介
2. ITPチームについて
3. ITPのコードベース上の工夫
4. **ITPの開発体制に関する工夫** ← 次ここ
5. まとめ

What is SCRUM

持続可能なチームを作るための組織開発方法のフレームワーク

スクラムセレモニー（会議）

1スプリント（2週間）に一度

- Sprint Review (成果物の共有)
- Sprint Retrospective (全般的な振り返り)
- Product Backlog Refinement (タスクの確認・選択)
- Sprint Planning (スプリントで実行するタスクの選択)

毎日

- Standup (タスクの確認)

スクラムの長所・短所

長所

- 個人に依存しない
 - チームへのアサインがあるのみで個人へのアサインがない
 - (Daily standupで毎日担当を変更することも可能)

短所

- 仕様の共有のための会議に時間を使う (業務時間の10%ほど)

個人に依存しないことの利点

- 全員がほぼ完全なドメイン知識を持っている
 - 「OOはXXが詳しいから彼が担当」ということが起きない
- レビューのクオリティが高くなる
 - ソフトウェアデザイン的な視点だけでなくビジネス的な視点からも仕様をきちんと把握した上で高度なレビューができる
- モブプログラミングを実施できる
 - コンテキストが共有されているので
- 業務量の平準化
 - 休みやすい → 有給もSick Leaveも

ITPチームでのスクラム

Sprint Review (30min)

- ユーザーストーリーに沿って機能のデモを行う
- ユーザーの視点からステークホルダーからレビューをもらう
- 特に問題なければリリースできる



Sprint Retrospective (30min)

- そのスプリントでの行動をKPT(Keep/Problem/Try)で分類
- 開発の効率・品質を上げるために何ができるかを考えて次回のスプリントに活かす

Product Backlog Refinement (30min)

- チケットの内容を精査し、内容が不十分であれば明確にする
- メンバーがチケットの内容を理解しているか確認
 - 全員が機能を理解するまでとことんやる

Sprint Planning (1h)

- タスクを**S**・**M**・**L**でざっくり推定
 - 正確な推定は不可能なのでやるだけ無駄
- 推定が難しい調査系のタスクは「**2人で1日**」等の人月型タイムボックスで実施
 - その期間で終わらなければ、他のタスクとの優先度を再度調整
 - 優先度が下がって次のスプリントになることも

Daily standup (10min)

進捗状況の確認のために次の3つを各人がチームメンバーに説明

- 昨日やったこと
- 今日やること
- ブロック

スプリントの実施

- Sprint Planningで定義したタスクリストを優先度順に処理していく
- 特定の誰かが決まったアサインであることは少なく、Daily standupでその日の担当を決めるだけ

スクラムコーチ

- 今年に入ってからメルカリは全社的にスクラムの実施を開始した
- 外部の会社からスクラムコーチを招待して指導を受けている

Scrum so far

- ITPチームではスクラムを開始して半年ほど
- 開始直後はメンバー間でドメイン知識が共有されておらず、知識共有に時間がかかることも
- 最近はメンバー間で知識が平準化されてきたので効率が上がってきた 😎

スクラムまとめ

- チームメンバーの変動に強い
 - メンバーの参加・離脱の両方で
- 個人としての生産性よりもチームとしての生産性を重視する
 - 会社としての成果が出やすい

まとめ | 話したこと

- コードベース上の工夫
- 組織体制上の工夫（スクラム）

まとめ | 話したこと

- Re-architecture プロジェクト（古いページから新しいページにマイグレートする方法）
- レイヤごとに選択したテストツール（Jest,Enzyme,Storybook,Storyshots,Cypress）
- Redux-Sagaのテストが書きやすいという話ともっと書きやすくするための工夫
- **コードオーナー**による品質の保証とメンバーの教育
- QAとの協力的E2Eテスト
- WebViewの**デザインシステム**が難しいという話
- 最近筋トレを始めた話
- 開発時におけるバックエンドへの依存を減らすための**モックサーバー**
- 最近そのモックサーバーをon-browserにしようとしている話
- WebView特有のクライアント**ネイティブAPIのモッキング**