
Mobile Hands-on
Service Dev Engineer TAIKEN

Work

ハンズオン資料のダウンロードのお願い

モバイルハンズオンに参加頂き、ありがとうございます。

本日使用するハンズオン資料のダウンロードをお願いします。
ダウンロードURLはZoomのチャットで連絡します。

ダウンロード後、任意の場所でzipファイルの解凍をお願いします。

ハンズオン資料を準備する

Work

このスライドをPDFにしたファイルを提供しています。

PDFファイルを開いて見れる状態にしてください。

mobile-handson.pdf

サービス開発エンジニア体験

サービス/プロダクトの開発に欠かせない
アプリ開発とDevOpsを体験してみませんか？

10月　SPAハンズオン
11月　APIハンズオン
12月　モバイルハンズオン
1月　DevOpsハンズオン
2月　チームで腕試しハッカソン

スタッフ紹介

TIS株式会社

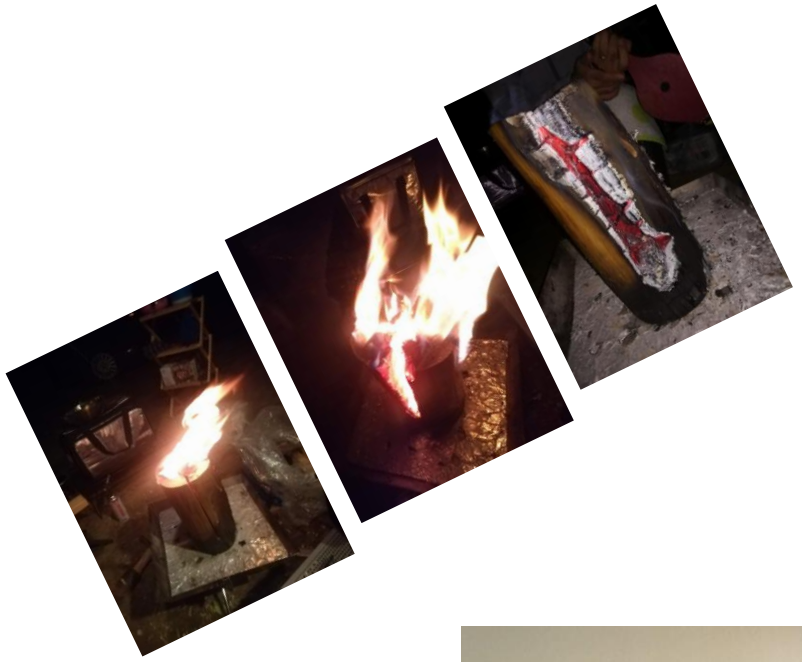
テクノロジー＆イノベーション本部

テクノロジー＆エンジニアセンター

伊藤 清人@会津若松

世古 雅也@会津若松

井上 拓門@東京



お願い

Zoomで名前（ニックネームも可）を分かるようにしてください。
オンライン開催なのでリアクションは大きな動きでお願いします！
周りの音が入り込まないようにお願いします。

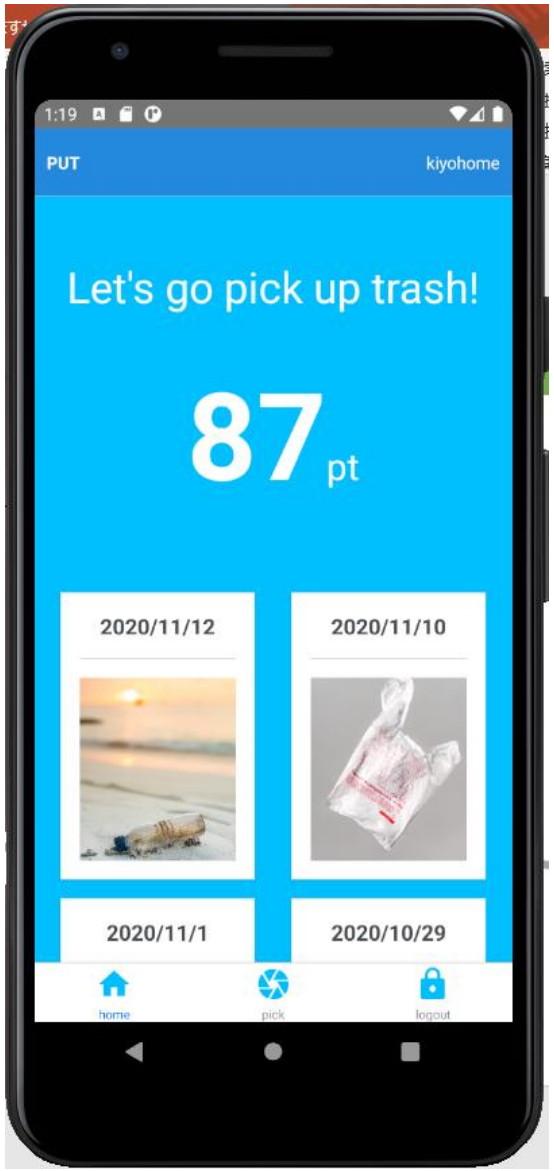
今後の改善等に活用したいので
ハンズオン終了後のアンケートにご協力をお願いします。

React Nativeを使った
モバイルアプリの作り方を学ぶハンズオン

ハンズオンの題材

ゴミ拾いアプリ「PUT」を作ります。

拾ったゴミの写真を送るとポイントが付与されるアプリです。

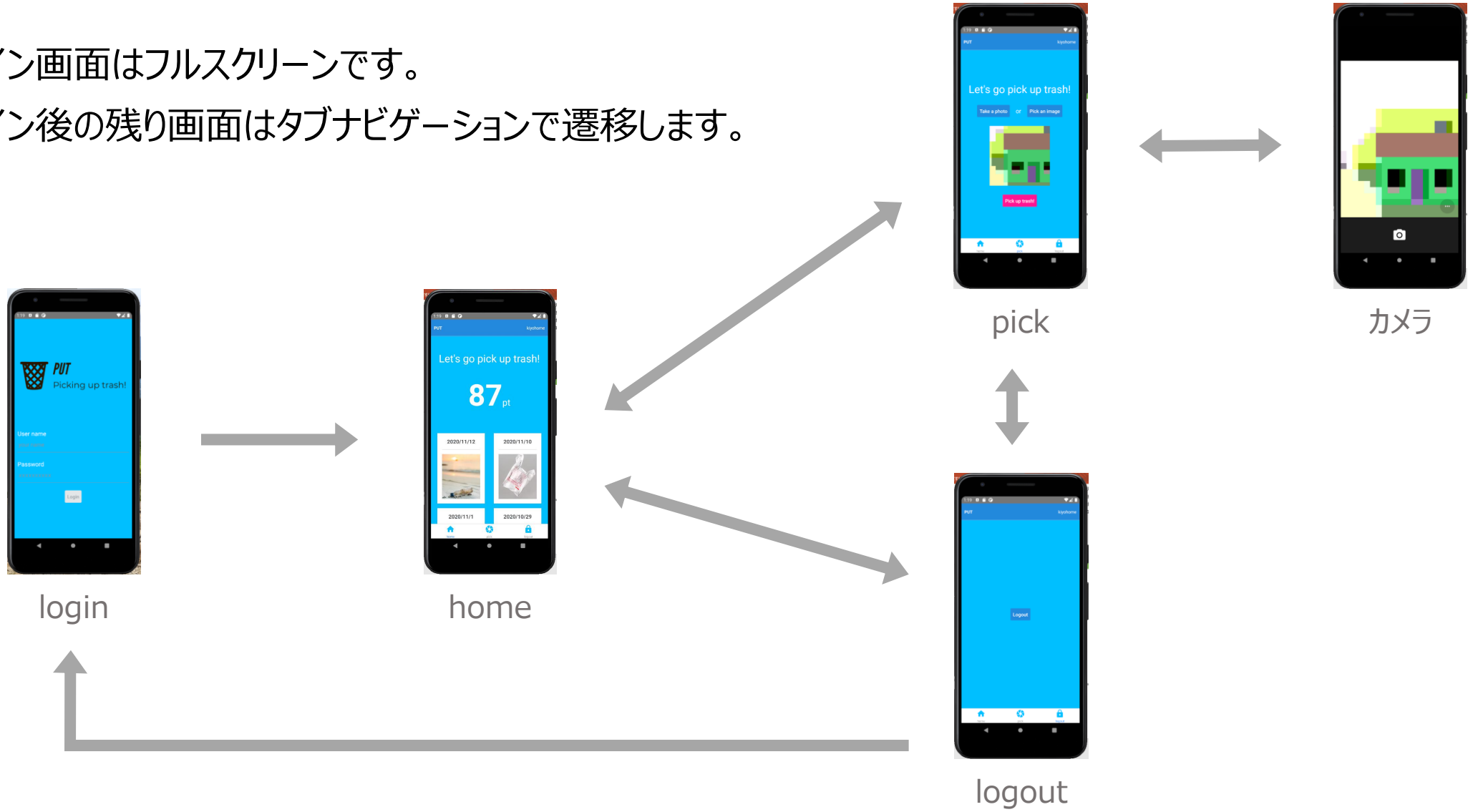


ハンズオンの題材

Demo

ログイン画面はフルスクリーンです。

ログイン後の残り画面はタブナビゲーションで遷移します。



ハンズオンのゴール

モバイルアプリの作り方を体験するがゴールです。
React NativeやTypeScriptの仕様や使い方は細かく説明しないです。（質問はしても大丈夫です！）
モバイルアプリの開発に必要となる技術要素をできるだけ多く体験できるように構成しています。
そのため、じっくりコーディングするより、ショートカットしてどんどん進めていくハンズオンです。
皆さんが作業しただけにならず、皆さんにモバイルアプリの作り方を持ち帰ってもらえるように頑張ります！

React Native
TypeScript ＜ モバイルアプリの作り方　⇒　体験

ハンズオンの進め方

スタッフが説明しながら作業→参加者も作業・・・といったかたちでStep by Stepで進行します。
皆さんの作業状況を確認しながら進めますのでリアクションをお願いします。

つまった場合は声をかけてください。画面共有して問題解消にあたります。

質問は進行中いつでも大丈夫です。
質問タイム、休憩中も受け付けますので遠慮なく聞いてください！

ハンズオンのスケジュール（全体240分）

オープニング（20分）

開発準備（70分）
モバイルアプリ入門
プロジェクトの準備
Open APIの活用
アプリの起動
休憩（10分）

開発（120分）
画面遷移
休憩（10分）
Login
Logout
休憩（10分）
Home
Pick
休憩（10分）

クロージング（20分）
アンケート（10分）

モバイルアプリ入門

モバイルアプリ

モバイルアプリが画面遷移をコントロールします。

操作に応じた
処理を呼び出す

処理結果に応じた
画面を表示する

スマホ
（モバイルアプリ）

リクエスト

データ

サーバ
（API）

何か処理する

アイコンはいらすとやを使用しています。
https://www.irasutoya.com/

クロスプラットフォーム開発

スマホアプリはiOSとAndroidへの対応が求められます。
コードを共有しないネイティブ実装（Swift、Kotlin）での開発は大変です。

1つのコードで複数のプラットフォーム向けの開発を行うことをクロスプラットフォーム開発と呼びます。
クロスプラットフォーム開発向けのフレームワークとしてFlutter、React Native等があります。
同じReactでSPAの開発にも活かせるのでReact Nativeを使います。

React Native
https://reactnative.dev/

モバイルアプリのクロスプラットフォーム開発：Flutter vs. React Nativeを徹底比較
https://kaopiz.com/ja-news-cross-platform-framework-flutter-vs-react-native/

React

SPA（Single Page Application）を作るためのフレームワークです。
画面部品(＝コンポーネント)、イベントハンドリング、
状態保持、画面遷移(＝ルーティング)など、
SPA作成に必要な仕組みを提供してくれます。

JavaScriptでコーディングします。

React
https://ja.reactjs.org/

```
import React from 'react';

function App() {
  return (
    <h1>Hello, world</h1>
  );
}

export default App;
```

「Hello, world」と出すReactのコード

ソースコードはcarbonを使用しています。
https://carbon.now.sh/

React Native

Reactでモバイルアプリを作れるにようしたフレームワークです。
クロスプラットフォーム開発ができます。

画面部品(＝コンポーネント)、イベントハンドリング、
状態保持、画面遷移(＝ルーティング)など、
モバイルアプリ作成に必要な仕組みを提供してくれます。

JavaScriptでコーディングします。

React Native
https://reactnative.dev/

```
import React from 'react';
import { Text, View } from 'react-native';

const HelloWorldApp = () => {
  return (
    <View style={{
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center'
      }}>
      <Text>Hello, world!</Text>
    </View>
  );
}

export default HelloWorldApp;
```

「Hello, world」と出すReact Nativeのコード

TypeScript

Reactの大部分はJavaScriptで作ります。
JavaScriptは型がないので苦労します。
動かすまで間違いに気づけないです。

少しでも苦労を和らげたいのでTypeScriptを使い、
JavaScriptに型を導入します。

学習コストに見合うだけの恩恵を受けられます。

```
type Todo = {
  id: number
  text: string
  completed: boolean
}

export const TodoBoard: React.FC = () => {
  const [todos] = useState<Todo[]>([
    { id: 2001, text: '洗い物をする', completed: true },
    { id: 2002, text: '洗濯物を干す', completed: false },
    { id: 2003, text: '買い物へ行く', completed: false }
  ]);

  return (
    <div className="TodoBoard_content">
      <TodoForm />
      <TodoFilter />
      <TodoList todos={todos}/>
    </div>
  );
};
```

TypeScriptで型を定義しているコード（Todo部分）

Visual Studio Code

TypeScriptの恩恵を受けるためVSCodeを使います。
型に基づいてコード補完や
コードの間違いを教えてくれます。

```
4
5   type Todo = {
6       id: number,
7       text: string,
8       completed: boolean
9   };
10
11  type Props = {
12      todos: Todo[],
13      toggleTodoCompletion: (id: number) => void
14  };
15
16  export const TodoList: React.FC<Props> = ({todos, toggleTodoCompletion}) => {
17      return (
18          <ul className="TodoList_list">
19              {todos.map(todo =>
20                  <TodoItem key={todo.id}
21                            id={todo.}
22                            text={tod  completed
23                            completed  id                    (property) id: number
24                            toggleTod  text
25              )}
26          </ul>
27      );
```

ToDoのプロパティがコード補完されます。number型も分かってます

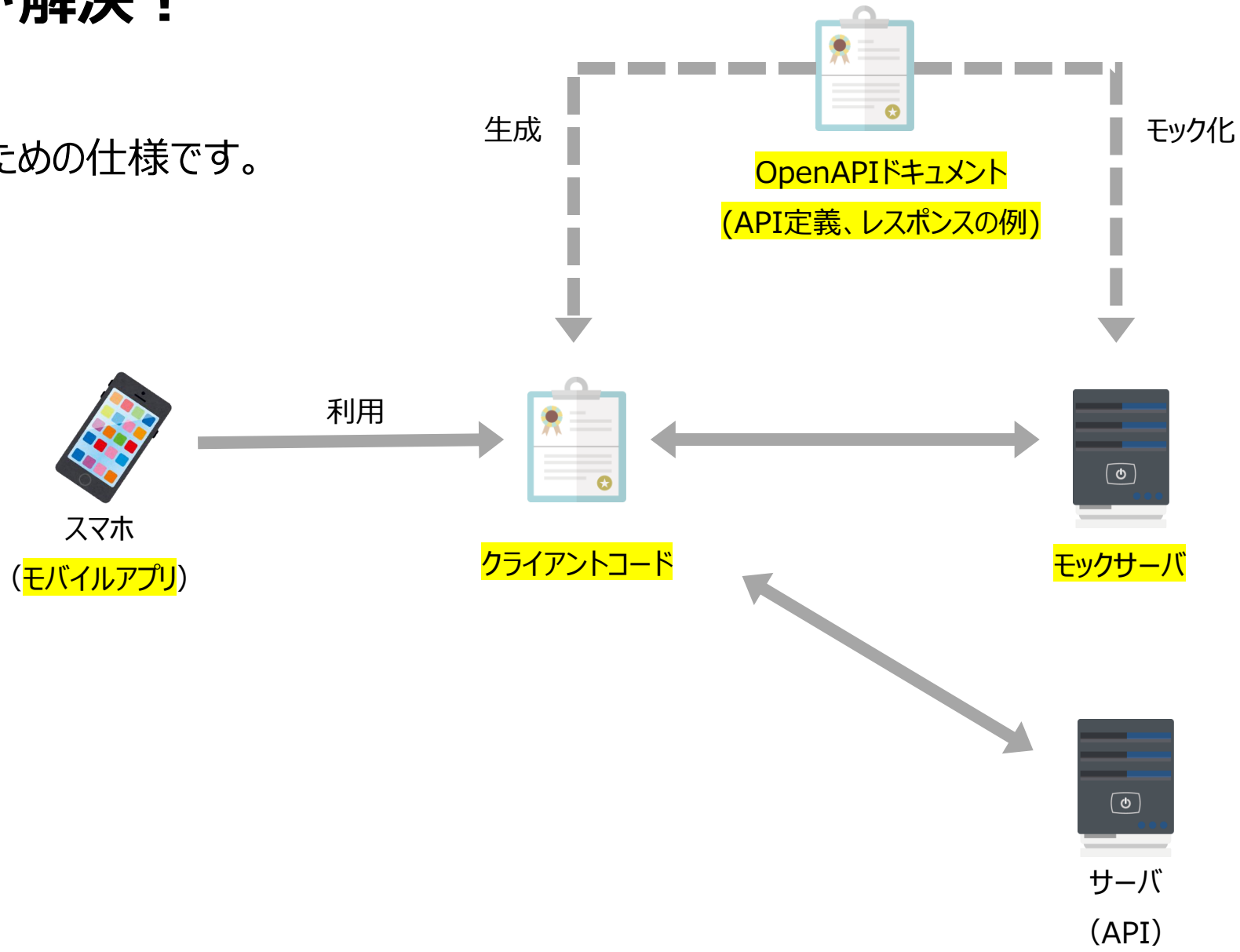
モバイルアプリとREST APIの並行開発？

モバイルアプリとAPIが独立しているのでうれしい面が多々ありますが、並行して開発するのが難しくなります。

APIがないとモバイルアプリが動かせない？

OpenAPIが解決！

APIを定義するための仕様です。



プロジェクトの準備

プロジェクトの作成

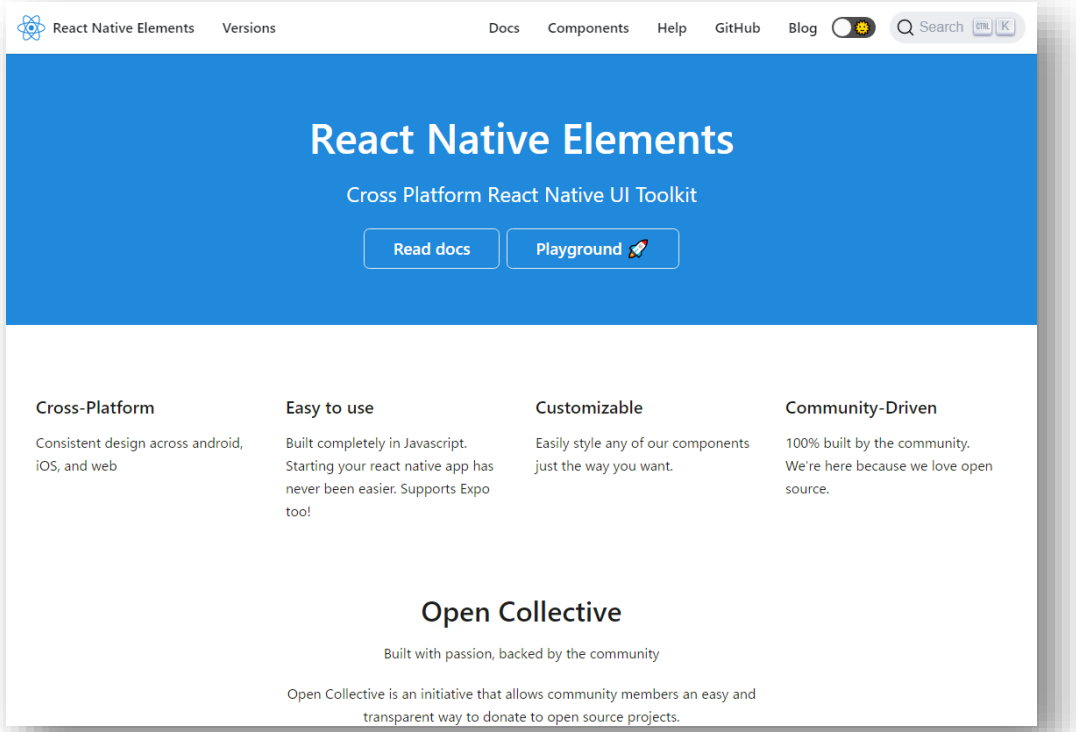
プロジェクトを作成します。

mobile-handson/put2-starterディレクトリを
任意の場所にコピーします。

ディレクトリ名をput2に変更します。
put2ディレクトリをVSCodeで開きます。

React Native Elementsを少し見てみましょう。

Expoのexpo-template-bare-typescriptから作成しています。
UIライブラリとしてReact Native Elementsを追加しています。



プロジェクトの作成

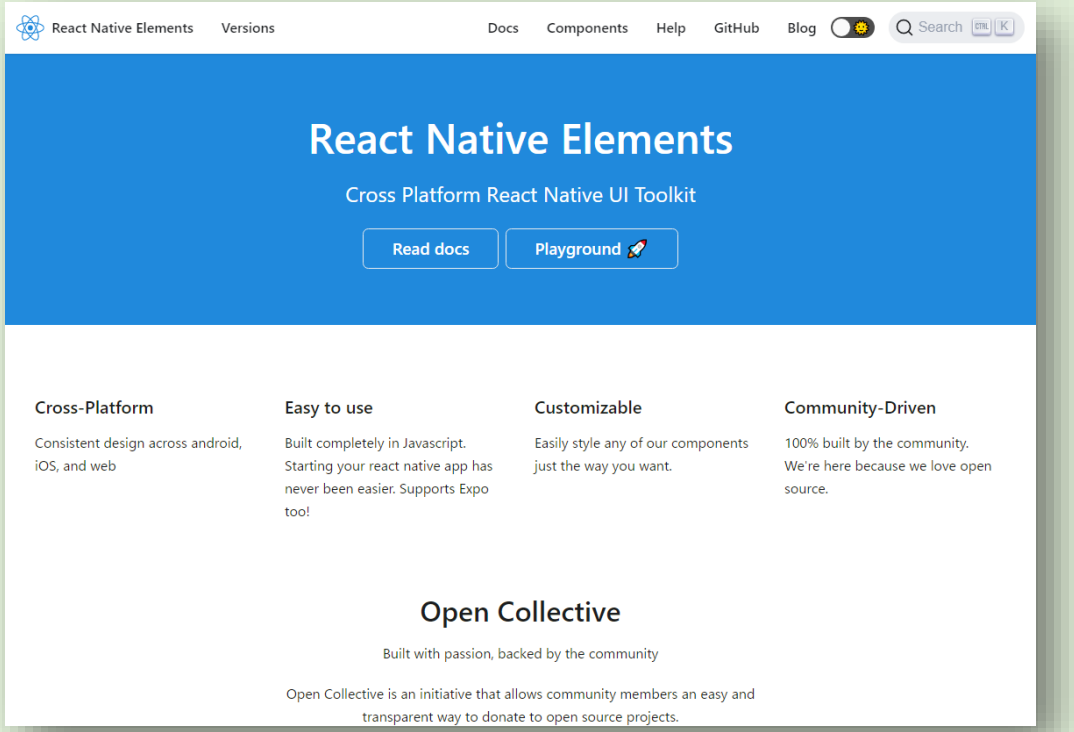
Work

プロジェクトを作成します。

mobile-handson/put2-starterディレクトリを
任意の場所にコピーします。

ディレクトリ名をput2に変更します。
put2ディレクトリをVSCodeで開きます。

React Native Elementsを少し見てみましょう。



Expoのexpo-template-bare-typescriptから作成しています。
UIライブラリとしてReact Native Elementsを追加しています。

パッケージのインストール

put2ディレクトリで次のコマンドを実行します。

```
$ npm install
```

時間がかかるので先に進みましょう。

パッケージのインストール

Work

put2ディレクトリで次のコマンドを実行します。

```
$ npm install
```

時間がかかるので先に進みましょう。

質問タイム

Open APIの活用

クライアントコードの生成

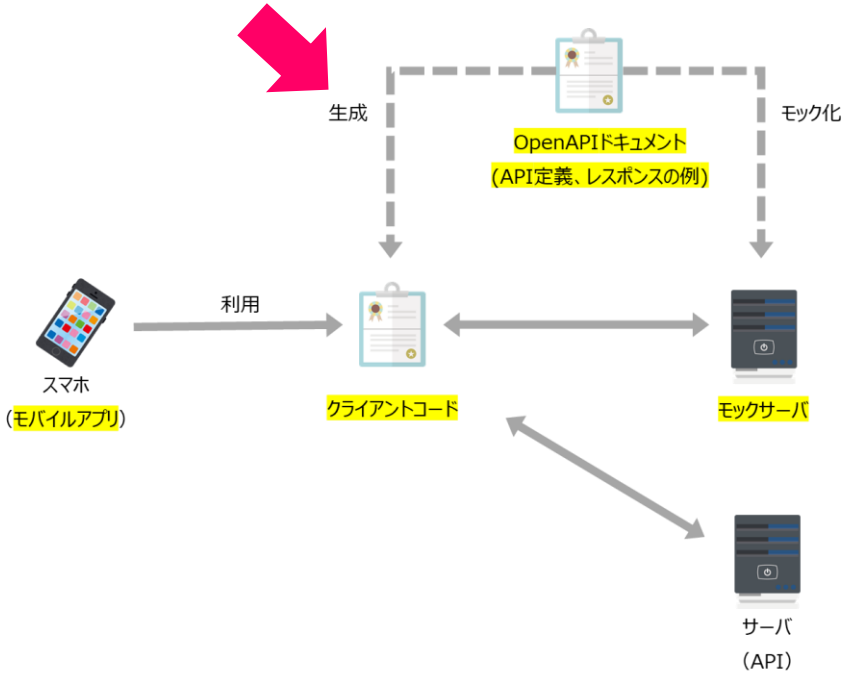
クライアントコードは既に生成済みなので作業しなくてよいです。

openapiディレクトリで次のコマンドで生成します。

$ docker-compose -f api-gen.yml up

次のディレクトリに生成されます。

/backend/generated-rest-client/



クライアントコードの生成にはOpenAPI Generatorというツールを使用しています。
Dockerコンテナで実行します。

OpenAPIドキュメントとクライアントコードの対応

OpenAPIドキュメント
(API定義、レスポンスの例)

```yaml
openapi: 3.0.3
info:
  title: PUT REST API
  version: '1.0.0'
  description: モバイルアプリハンズオンで作成するPUTアプリの
tags:
  - name: users
    description: ユーザ管理
  - name: trash
    description: ゴミ拾い管理
servers:
  - url: 'http://localhost:9080'
paths:
  /api/trash:
    get:
      summary: ゴミ拾い一覧の取得
      description: >
        登録しているゴミ拾いを全て取得する。
      tags:
        - trash
      operationId: getTrashList
      responses:
        '200':
          description: OK
          content:
            'application/json':
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Trash'
              examples:
                example:
                  value:
                    - id: 1001
                      imageUrl: https://images.unsplash.com/・・・
                      date: 2020/11/12
                      point: 34
                        :
```

クライアントコード

```typescript
export class TrashApi extends runtime.BaseAPI {

    :

    /**
     * 登録しているゴミ拾いを全て取得する。
     * ゴミ拾い一覧の取得
     */
    async getTrashList(): Promise<Array<Trash>> {
        const response = await this.getTrashListRaw();
        return await response.value();
    }

    :
```

tagsでグルーピングします。
コード生成するとグループごとにクラスが作成されます。
operationIdでREST APIを識別するIDを指定します。
コード生成すると関数名に使われます。

モックサーバが返すレスポンス

OpenAPIドキュメント
(API定義、レスポンスの例)

```
          :
        examples:
          example:
            value:
              - id: 1001
                imageUrl: https://images.unsplash.com/・・・
                date: 2020/11/12
                point: 34
              - id: 1002
                imageUrl: https://images.unsplash.com/・・・
                date: 2020/11/10
                point: 17
              - id: 1003
                imageUrl: https://images.unsplash.com/・・・
                date: 2020/11/1
                point: 8
              - id: 1004
                imageUrl: https://images.unsplash.com/・・・
                date: 2020/10/29
                point: 17
              - id: 1005
                imageUrl: https://images.unsplash.com/・・・
                date: 2020/10/20
                point: 11
          :
```

レスポンス
（JSON）

```
[
  {
    "id": 1001,
    "imageUrl": "https://images.unsplash.com/・・・",
    "date": "2020/11/12",
    "point": 34
  },
  {
    "id": 1002,
    "imageUrl": "https://images.unsplash.com/・・・",
    "date": "2020/11/10",
    "point": 17
  },
  {
    "id": 1003,
    "imageUrl": "https://images.unsplash.com/・・・",
    "date": "2020/11/1",
    "point": 8
  },
  {
    "id": 1004,
    "imageUrl": "https://images.unsplash.com/・・・",
    "date": "2020/10/29",
    "point": 17
  },
  {
    "id": 1005,
    "imageUrl": "https://images.unsplash.com/・・・",
    "date": "2020/10/20",
    "point": 11
  }
]
```

examplesに実際に返却される例を定義します。
モックサーバが返却するデータになります。

モックサーバの起動

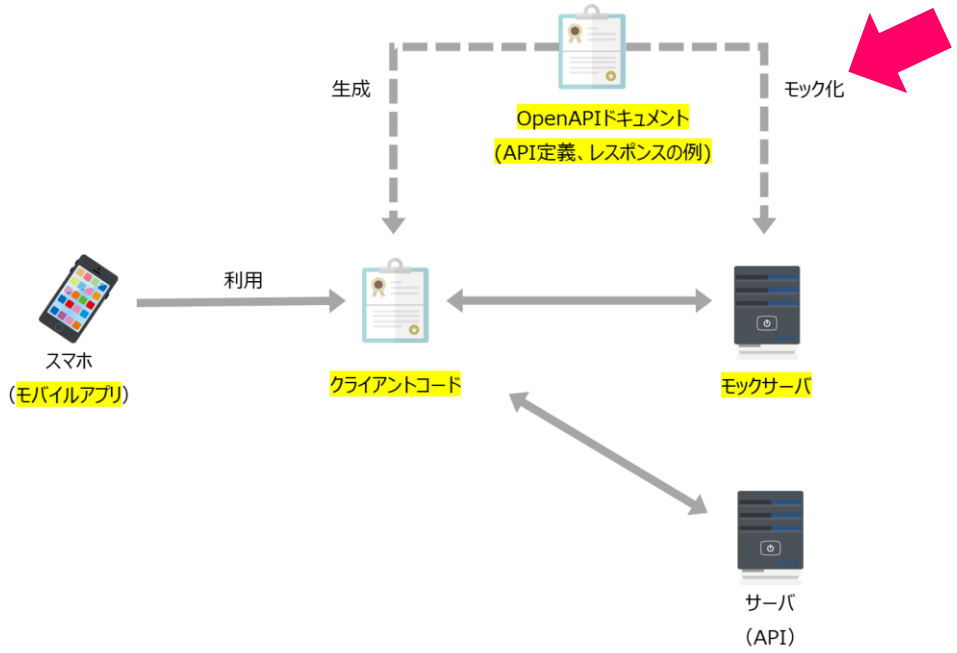
REST APIを呼び出せるようにします。

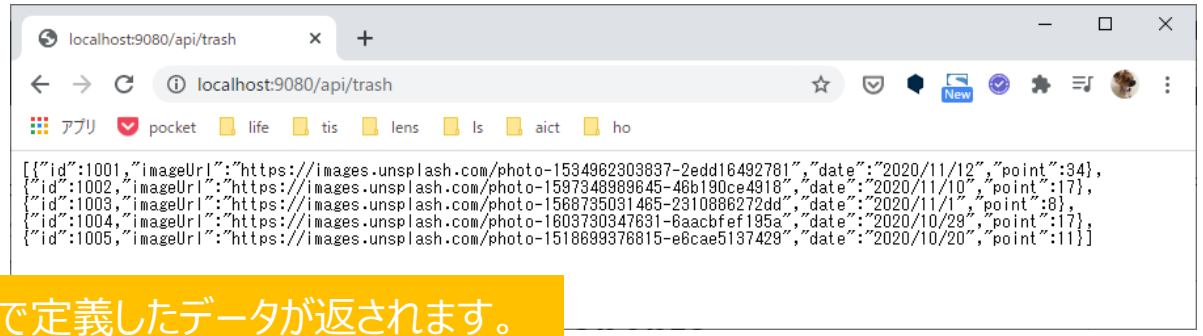
openapiディレクトリで次のコマンドで起動します。

$ docker-compose -f api-mock.yml up

ブラウザで次のURLにアクセスします。

http://localhost:9080/api/trash





examplesで定義したデータが返されます。

モックサーバの起動

Work

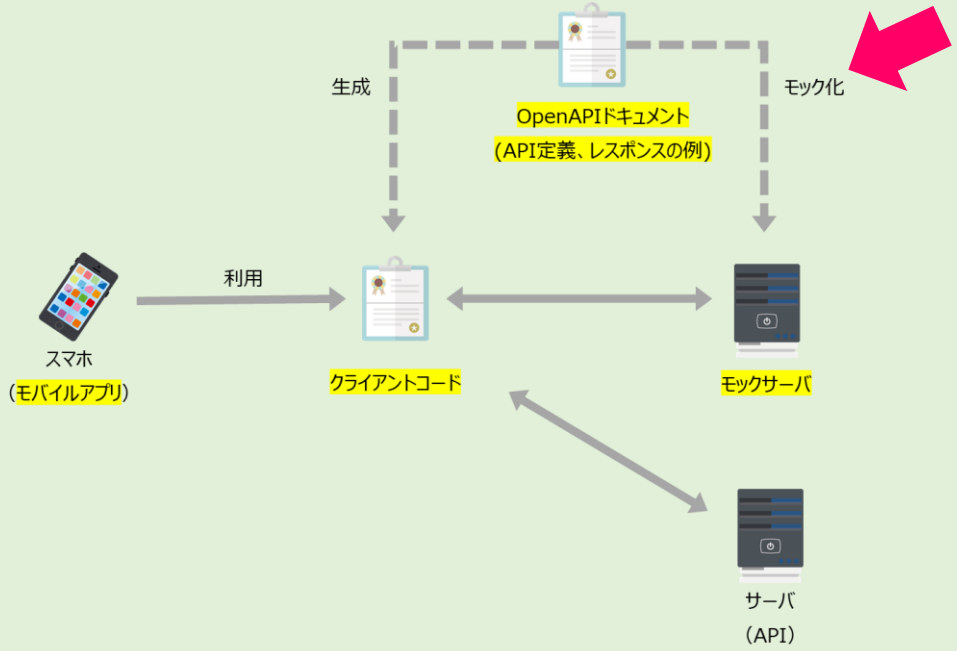
REST APIを呼び出せるようにします。

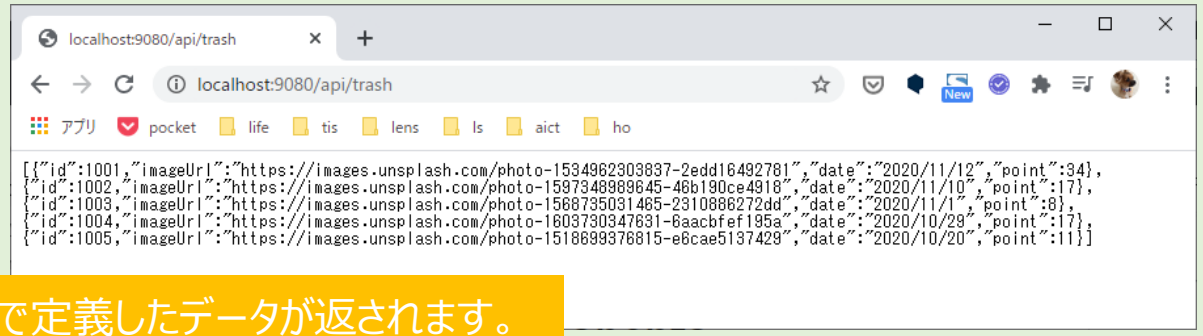
openapiディレクトリで次のコマンドで起動します。

```
$ docker-compose -f api-mock.yml up
```

ブラウザで次のURLにアクセスします。

http://localhost:9080/api/trash





examplesで定義したデータが返されます。

APIクライアント

生成したクライアントコードをラッピングしたBackendSerivceを作成しています。

BackendServiceにより、各機能を作る時のREST APIの呼び出しを実装しやすくし、
API呼び出し時の共通処理を埋め込むことが可能になります。

backendディレクトリのBackendService.tsファイルを確認しましょう。

BackendService

Reading

```
import { Configuration, Middleware, TrashApi, UsersApi } from './generated-rest-client';

const logger: Middleware = {
  pre: async (context) => {
    console.log(`>> ${context.init.method} ${context.url}`, context.init);
  },
  post: async (context) => {
    console.log(`<< ${context.response.status} ${context.url}`, context.response);
  },
};

const config = new Configuration({
  middleware: [logger],
});

const trashApi = new TrashApi(config);

const usersApi = new UsersApi(config);

const login = async (userName: string, password: string) => {
  return usersApi.login({ inlineObject: { userName, password } });
};

const logout = async () => {
  return usersApi.logout();
};

const getTrashList = async () => {
  return trashApi.getTrashList();
};

const postTrash = async (trash: Blob) => {
  return trashApi.postTrash({ body: trash });
};

export const BackendService = {
  login,
  logout,
  getTrashList,
  postTrash,
};
```

Middlewareと呼ばれる部品を作成して、
リクエストやレスポンスに対する共通的な処理を実装できます。
開発時にREST APIの呼び出しを確認しやすいように、
リクエストとレスポンスをコンソールにログ出力する
Middlewareを作成しています。

アプリの起動

アプリの起動

事前準備で実施して頂いた方法で
ブラウザorスマホでアプリを起動します。

put2ディレクトリで次のコマンドを実行します。
$ npm run start

ブラウザで次のURLにアクセスします。
http://localhost:19002

画面にHello World!が表示されれば成功です。

Work

アプリの起動

事前準備で実施して頂いた方法で
ブラウザorスマホでアプリを起動します。

put2ディレクトリで次のコマンドを実行します。
$ npm run start

ブラウザで次のURLにアクセスします。
http://localhost:19002

画面にHello World!が表示されれば成功です。

5分休憩

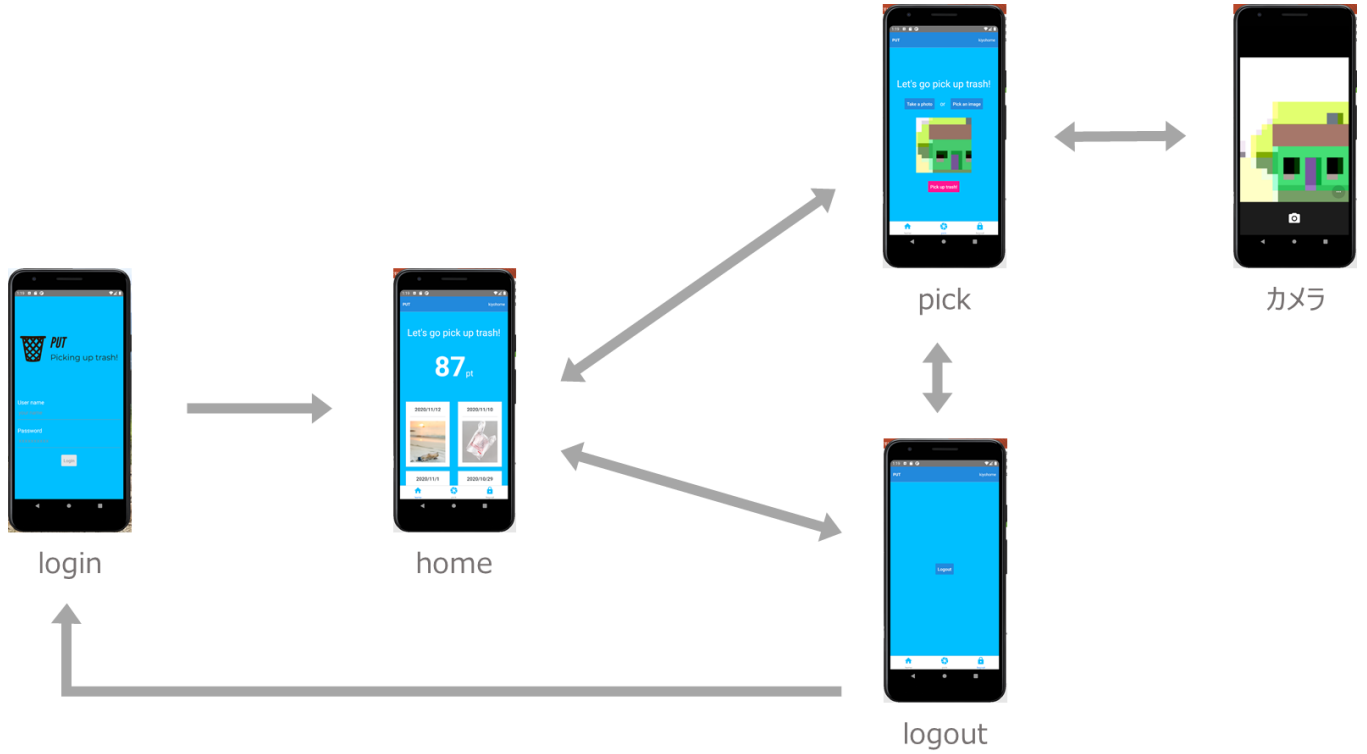
休憩中は質問OKです！

開発の進め方

開発の進め方

次の順で作成していきます。

- 画面遷移
- Login、Logout
- Home
- Pick

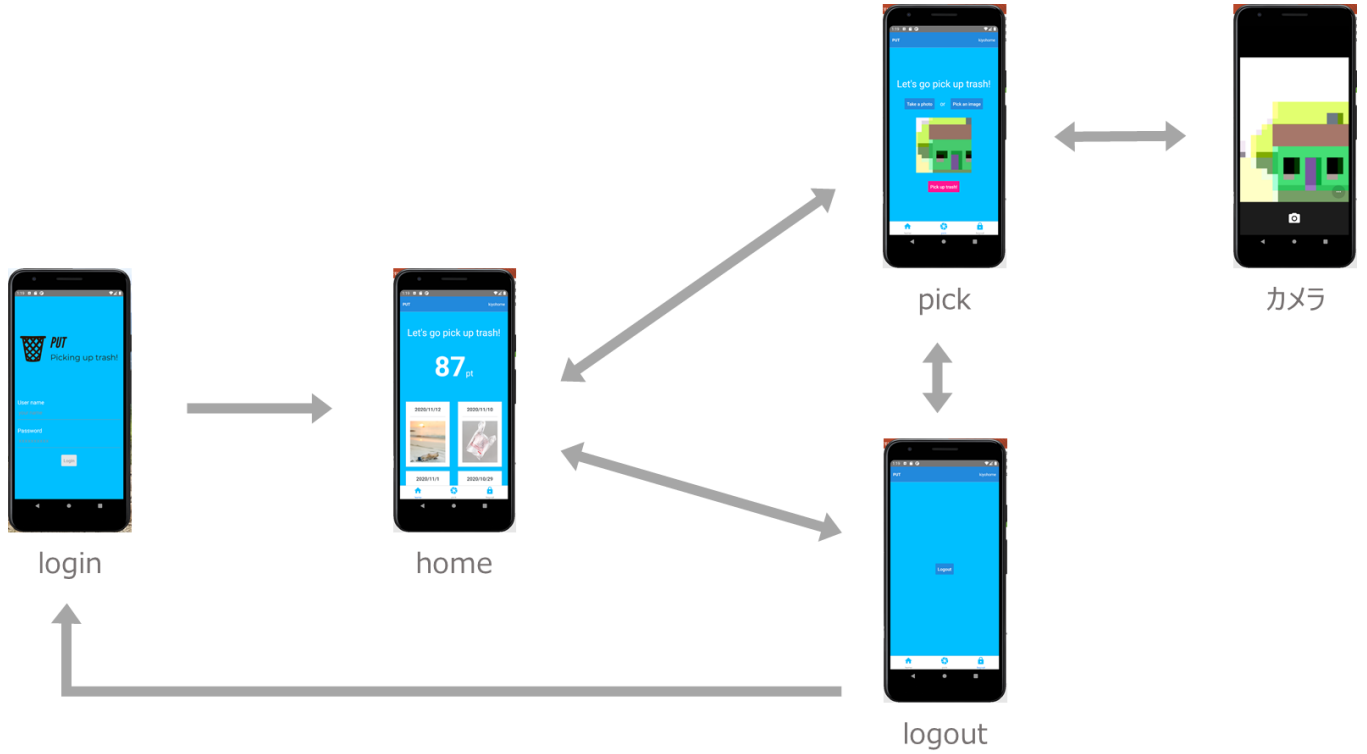


各画面のソースコード

Reading

初期状態で各画面のデザインのみ実装してあります。

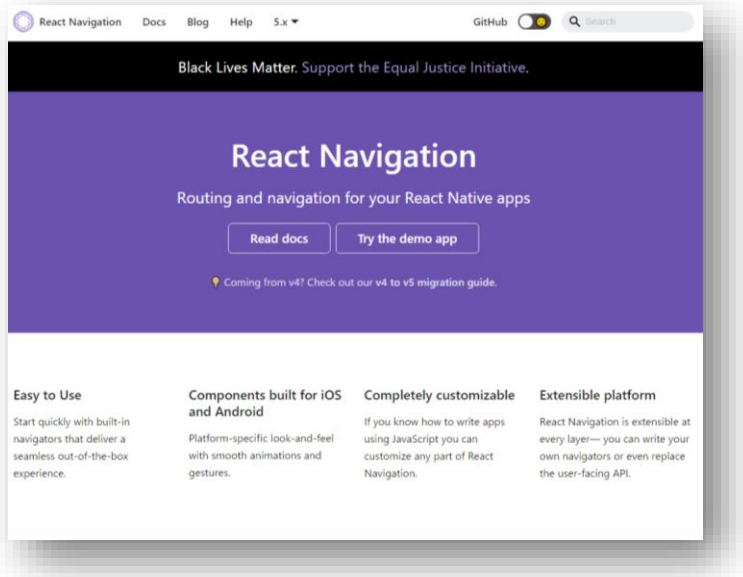
各画面のソースを見てみましょう。



画面遷移

React Navigation

3つのナビゲーションが提供されています。
これらを組み合わせてモバイルアプリの画面遷移を作成します。



Stack

https://reactnavigation.org/docs/stack-navigator

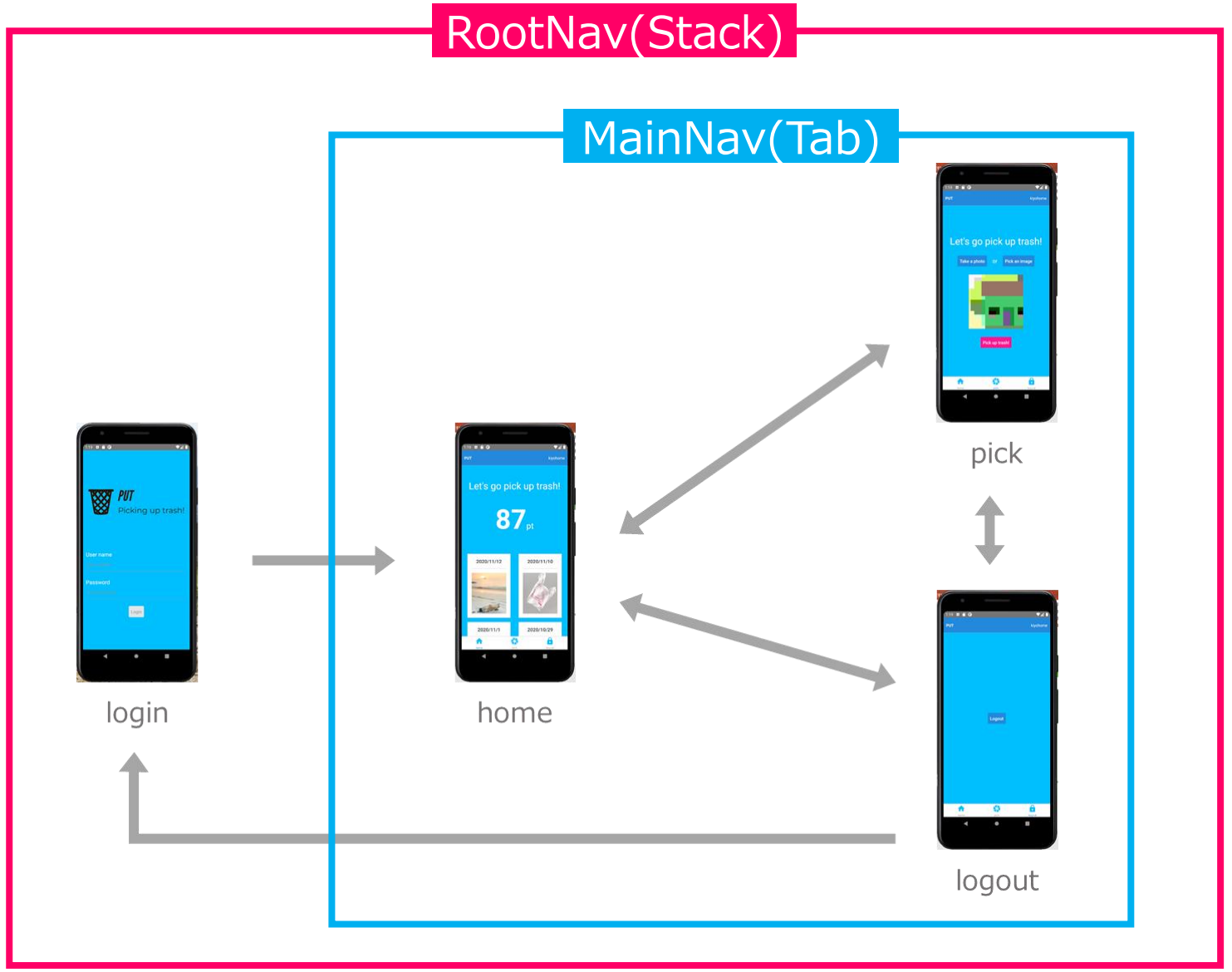
Tab

https://reactnavigation.org/docs/bottom-tab-navigator

Drawer

https://reactnavigation.org/docs/drawer-navigator

画面遷移とナビゲーション



RootNav(Stack)

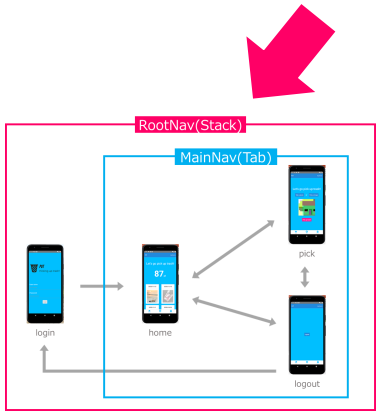
```
:

const Stack = createStackNavigator();

const RootNav: React.FC = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator headerMode="none">
        <Stack.Screen {...MainNav} />
        <Stack.Screen {...Login} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default RootNav;
```

MainNav(Tab)

```
:

const Tab = createBottomTabNavigator();

const MainNav: React.FC = () => {
  return (
    <Tab.Navigator initialRouteName="home">
      <Tab.Screen {...Home} />
      <Tab.Screen {...Picking} />
      <Tab.Screen {...Logout} />
    </Tab.Navigator>
  );
};

export default { name: 'main', component: MainNav };
```

RootNavの表示

```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import { StyleSheet, View } from 'react-native';
import { Text } from 'react-native-elements';

const styles = StyleSheet.create({
  container: {
    flexGrow: 1,
    backgroundColor: '#00bfff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  hello: {
    fontSize: 50
  },
});

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <View style={styles.container}>
      <Text style={styles.hello}>Hello, World!</Text>
    </View>
  );
};

export default App;
```



App

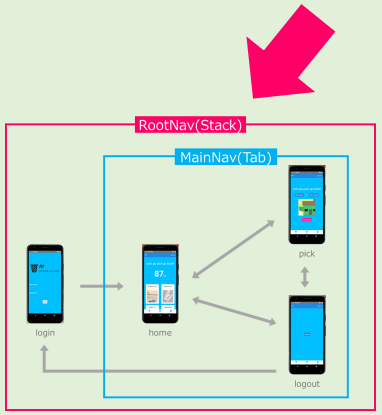
```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import RootNav from './components/pages/RootNav';

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <RootNav />
  );
};

export default App;
```

RootNavを表示します。

RootNavの表示

Work

```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import { StyleSheet, View } from 'react-native';
import { Text } from 'react-native-elements';

const styles = StyleSheet.create({
  container: {
    flexGrow: 1,
    backgroundColor: '#00bfff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  hello: {
    fontSize: 50
  },
});

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <View style={styles.container}>
      <Text style={styles.hello}>Hello, World!</Text>
    </View>
  );
};

export default App;
```



App

```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import RootNav from './components/pages/RootNav';

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <RootNav />
  );
};

export default App;
```

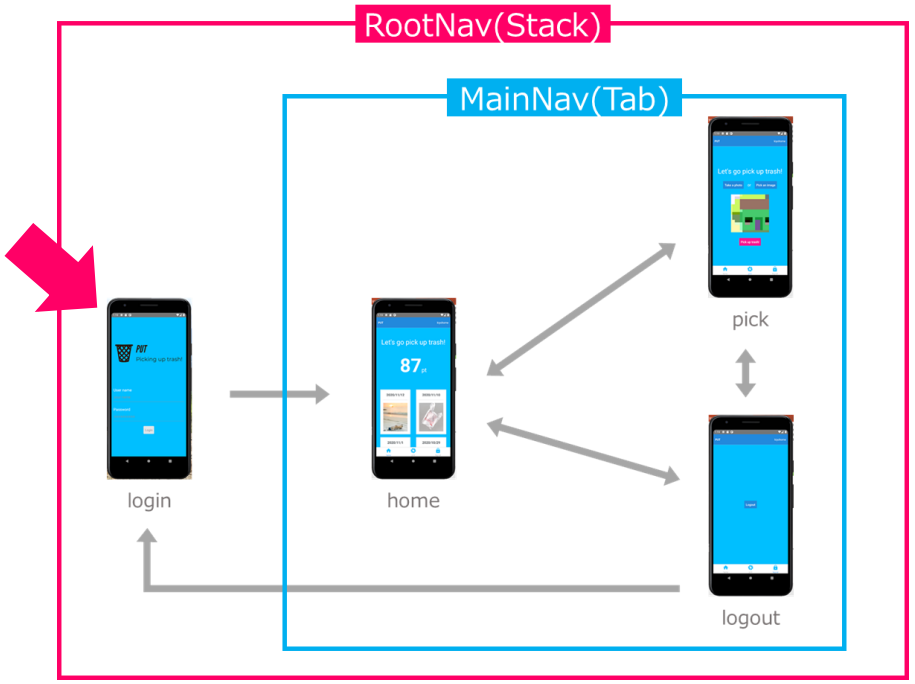
RootNavを表示します。

Loginの表示

RootNav

```
import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationContainer } from '@react-navigation/native';
import Login from './Login';

const Stack = createStackNavigator();

const RootNav: React.FC = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator headerMode="none">
        <Stack.Screen {...Login} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default RootNav;
```

Stackを作成し、
StackにLoginを追加します。




```
<Stack.Screen {...Login} />
```

```
<Stack.Screen name={Login.name} component={Login.component} />
```

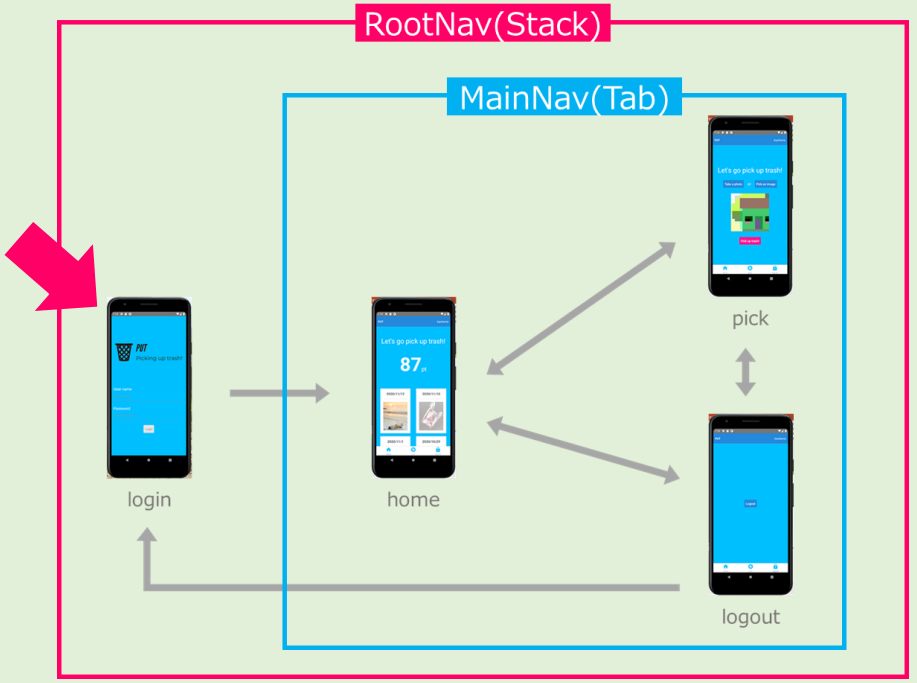
Loginの表示

Work

RootNav

```
import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationContainer } from '@react-navigation/native';
import Login from './Login';

const Stack = createStackNavigator();

const RootNav: React.FC = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator headerMode="none">
        <Stack.Screen {...Login} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default RootNav;
```

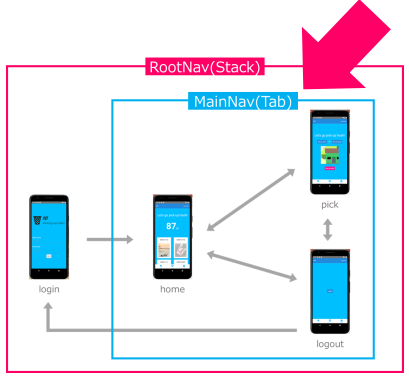
Stackを作成し、
StackにLoginを追加します。




```
<Stack.Screen {...Login} />
```

```
<Stack.Screen name={Login.name} component={Login.component} />
```

MainNavの表示



RootNav

```
:
import Login from './Login';
import MainNav from './MainNav';

const Stack = createStackNavigator();

const RootNav: React.FC = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator headerMode="none" initialRouteName="login">
        <Stack.Screen {...MainNav} />
        <Stack.Screen {...Login} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default RootNav;
```

StackにMainNavを追加します。

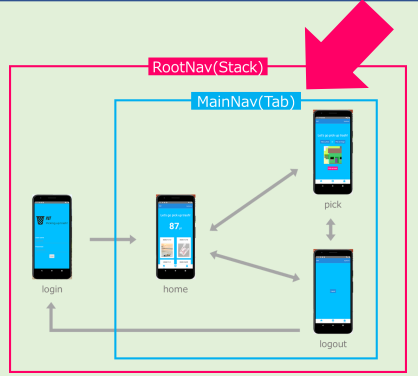
Login

```
import { useNavigation } from '@react-navigation/native';
import React from 'react';
:

const styles = StyleSheet.create({
  :
});

const Component: React.FC = () => {

  const navigation = useNavigation();

  const login = () => {
    navigation.navigate('main');
  };

  return (
    <Page>
      <Image containerStyle={styles.image} source={require('../../assets/logo.png')}
        style={{ height: 200, width: 500 }} />
      <Text style={styles.label}>User name</Text>
      <Input placeholder="your name" />
      <Text style={styles.label}>Password</Text>
      <Input placeholder="xxxxxxxxxx" secureTextEntry />
      <Button title="Login" onPress={login} />
    </Page>
  );
};

export default rootPage('login', Component);
```

Loginボタンが押されたら
MainNavに遷移させます。

MainNavの表示

Work



RootNav

```
 :
import Login from './Login';
import MainNav from './MainNav';

const Stack = createStackNavigator();

const RootNav: React.FC = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator headerMode="none" initialRouteName="login">
        <Stack.Screen {...MainNav} />
        <Stack.Screen {...Login} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default RootNav;
```

StackにMainNavを追加します。

Login

```
import { useNavigation } from '@react-navigation/native';
import React from 'react';
 :

const styles = StyleSheet.create({
  :
});

const Component: React.FC = () => {

  const navigation = useNavigation();

  const login = () => {
    navigation.navigate('main');
  };

  return (
    <Page>
      <Image containerStyle={styles.image} source={require('../../assets/logo.png')}
        style={{ height: 200, width: 500 }} />
      <Text style={styles.label}>User name</Text>
      <Input placeholder="your name" />
      <Text style={styles.label}>Password</Text>
      <Input placeholder="xxxxxxxxxx" secureTextEntry />
      <Button title="Login" onPress={login} />
    </Page>
  );
};

export default rootPage('login', Component);
```

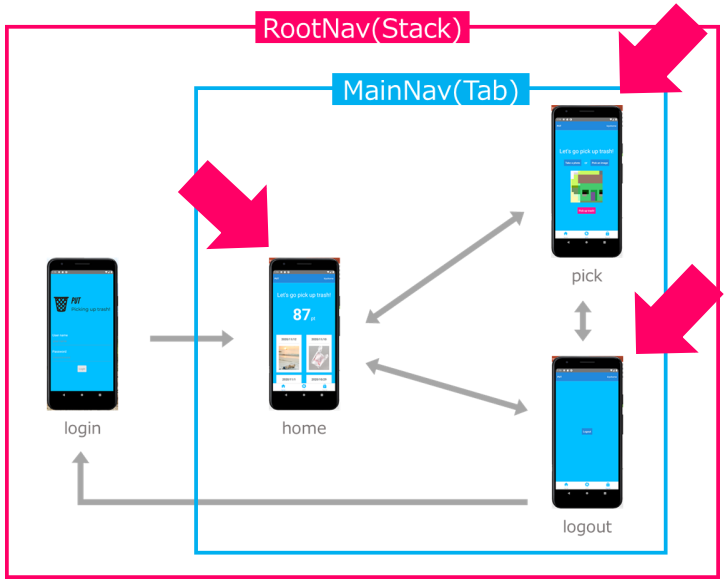
Loginボタンが押されたら
MainNavに遷移させます。

Home、Pick、Logoutの表示

MainNav

```
import React from 'react';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import Home from './Home';
import Picking from './Picking';
import Logout from './Logout';

const Tab = createBottomTabNavigator();

const MainNav: React.FC = () => {
  return (
    <Tab.Navigator initialRouteName="home">
      <Tab.Screen {...Home} />
      <Tab.Screen {...Picking} />
      <Tab.Screen {...Logout} />
    </Tab.Navigator>
  );
};

export default { name: 'main', component: MainNav };
```

Tabを作成し、
Tabに各画面を追加します。



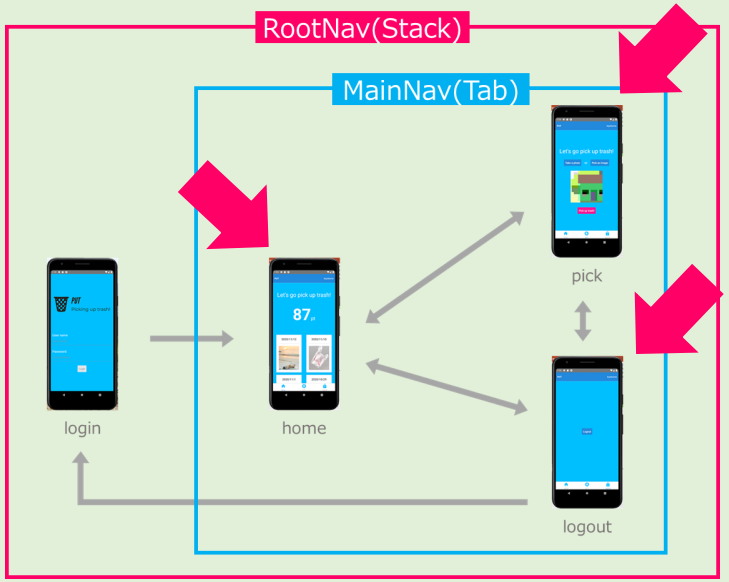
Home、Pick、Logoutの表示

Work

MainNav

```
import React from 'react';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import Home from './Home';
import Picking from './Picking';
import Logout from './Logout';

const Tab = createBottomTabNavigator();

const MainNav: React.FC = () => {
  return (
    <Tab.Navigator initialRouteName="home">
      <Tab.Screen {...Home} />
      <Tab.Screen {...Picking} />
      <Tab.Screen {...Logout} />
    </Tab.Navigator>
  );
};

export default { name: 'main', component: MainNav };
```

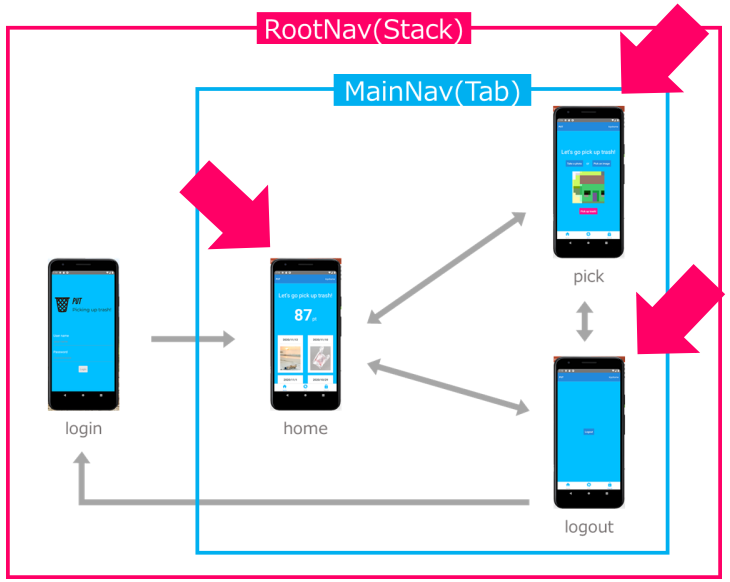
Tabを作成し、
Tabに各画面を追加します。



Pick、Logoutのボタンによる遷移

Picking

```
import { useNavigation } from '@react-navigation/native';
import React from 'react‘;
:

const styles = StyleSheet.create({
  :
});

const Component: React.FC = () => {

  const navigation = useNavigation();

  const pickUpTrash = () => {
    navigation.navigate('home');
  }

  return (
    <Page>
      <Text style={styles.lead}>Let's go pick up trash!</Text>
      <View style={styles.select}>
        <Button title="Take a photo" />
        <Text style={styles.or}>or</Text>
        <Button title="Pick an image" />
      </View>
      <Image source={{ uri: ‘https://images.unsplash.com/・・・' }}
        style={styles.image} />
      <Button title="Pick up trash!" buttonStyle={styles.upload}
        onPress={pickUpTrash} />
    </Page>
  );
};

export default mainPage('pick', Component, 'camera');
```



Logout

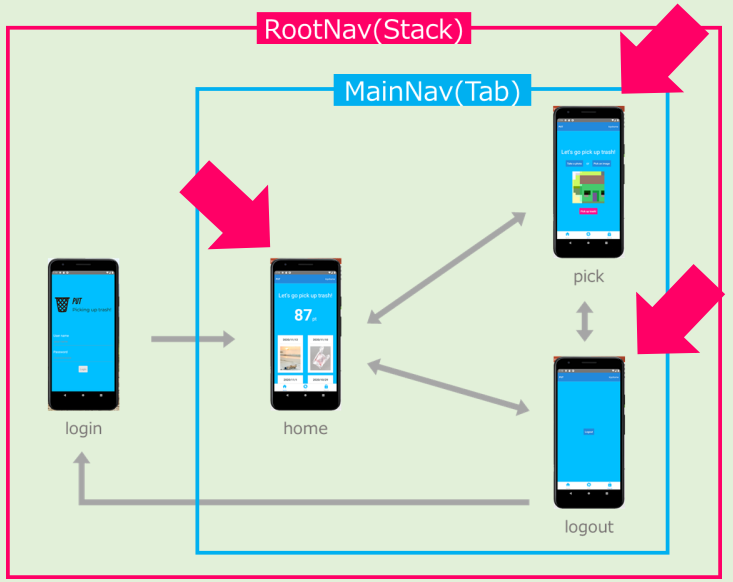
```
import { useNavigation } from '@react-navigation/native';
import React from 'react';
import { Button } from 'react-native-elements';
import { Page, mainPage } from './MainPage';

const Component: React.FC = () => {

  const navigation = useNavigation();

  const logout = () => {
    navigation.navigate('login');
  }

  return (
    <Page>
      <Button title="Logout" onPress={logout} />
    </Page>
  );
};

export default mainPage('logout', Component, 'lock');
```

Pick、Logoutのボタンによる遷移

Work

Picking

```
import { useNavigation } from '@react-navigation/native';
import React from 'react‘;
:

const styles = StyleSheet.create({
  :
});

const Component: React.FC = () => {

  const navigation = useNavigation();

  const pickUpTrash = () => {
    navigation.navigate('home');
  }

  return (
    <Page>
      <Text style={styles.lead}>Let's go pick up trash!</Text>
      <View style={styles.select}>
        <Button title="Take a photo" />
        <Text style={styles.or}>or</Text>
        <Button title="Pick an image" />
      </View>
      <Image source={{ uri: ‘https://images.unsplash.com/・・・' }}
        style={styles.image} />
      <Button title="Pick up trash!" buttonStyle={styles.upload}
        onPress={pickUpTrash} />
    </Page>
  );
};

export default mainPage('pick', Component, 'camera');
```



Logout

```
import { useNavigation } from '@react-navigation/native';
import React from 'react';
import { Button } from 'react-native-elements';
import { Page, mainPage } from './MainPage';

const Component: React.FC = () => {

  const navigation = useNavigation();

  const logout = () => {
    navigation.navigate('login');
  }

  return (
    <Page>
      <Button title="Logout" onPress={logout} />
    </Page>
  );
};

export default mainPage('logout', Component, 'lock');
```

5分休憩

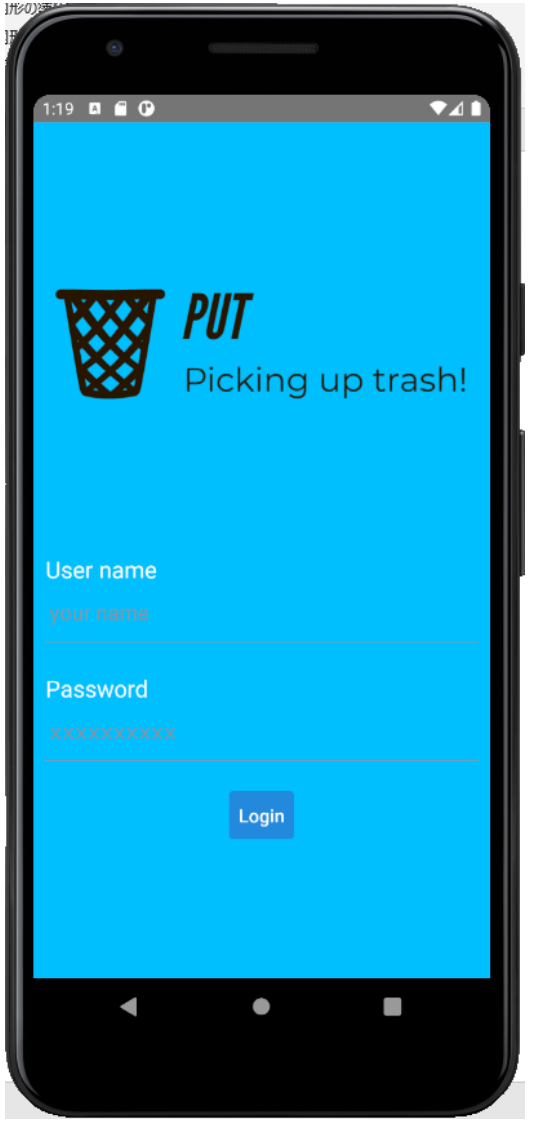
休憩中は質問OKです！

Login

Login

次の順で作成していきます。

- 入力データを受け取れるようにします。
- 入力されたらボタンを押せるようにします。
- REST APIを呼び出し
  ログイン状態に応じた表示切替をします。

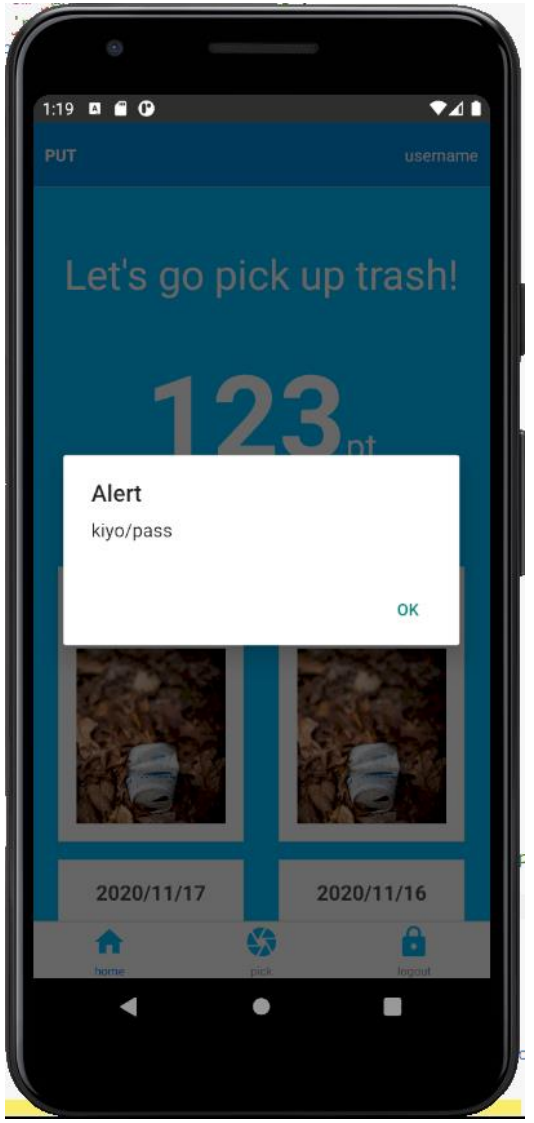


入力データを受け取る

Login

```
import { useNavigation } from '@react-navigation/native';
import React, { useState } from 'react';
import { StyleSheet } from 'react-native‘;
:

const Component: React.FC = () => {

  const navigation = useNavigation();
  const [userName, setUserName] = useState<string>('');
  const [password, setPassword] = useState<string>('');

  const login = () => {
    alert(userName + '/' + password);
    navigation.navigate('main');
  };

  return (
    <Page>
      <Image containerStyle={styles.image} source={require('../assets/logo.png')}
        style={styles.image} />
      <Text style={styles.label}>User name</Text>
      <Input placeholder=“Your name" value={userName}
        onChangeText={setUserName} />
      <Text style={styles.label}>Password</Text>
      <Input placeholder=“Password" secureTextEntry
        value={password} onChangeText={setPassword} />
      <Button title="Login" onPress={login} />
    </Page>
  );
};

export default rootPage('login',
```

入力データやユーザ操作で変化する値は状態として保持します。Reactが提供するstateフックと呼ばれるuseState関数を使用して実装します。




```
const [値, 値を更新する関数] = useState<型>(初期値);
```

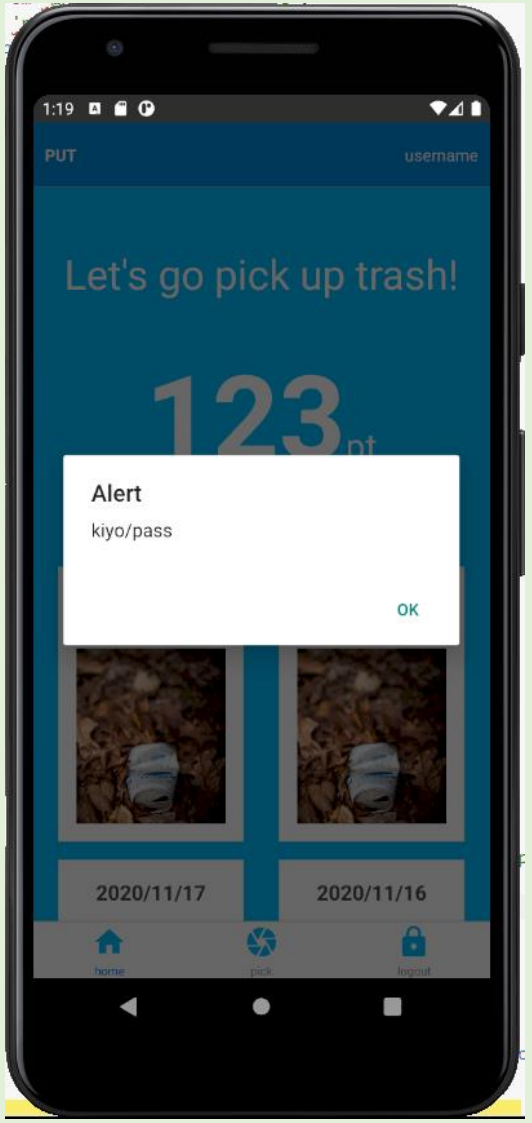
入力データを受け取る

Work

Login

```
import { useNavigation } from '@react-navigation/native';
import React, { useState } from 'react';
import { StyleSheet } from 'react-native‘;
:

const Component: React.FC = () => {

  const navigation = useNavigation();
  const [userName, setUserName] = useState<string>('');
  const [password, setPassword] = useState<string>('');

  const login = () => {
    alert(userName + '/' + password);
    navigation.navigate('main');
  };

  return (
    <Page>
      <Image containerStyle={styles.image} source={require('../assets/logo.png')}
        style={styles.image} />
      <Text style={styles.label}>User name</Text>
      <Input placeholder=“Your name" value={userName}
        onChangeText={setUserName} />
      <Text style={styles.label}>Password</Text>
      <Input placeholder=“Password" secureTextEntry
        value={password} onChangeText={setPassword} />
      <Button title="Login" onPress={login} />
    </Page>
  );
};

export default rootPage('login',
```

入力データやユーザ操作で変化する値は状態として保持します。Reactが提供するstateフックと呼ばれるuseState関数を使用して実装します。




```
const [値, 値を更新する関数] = useState<型>(初期値);
```

入力されたらボタンを押せるようにする

Login

```
import React, { useMemo, useState } from 'react';
 :

const Component: React.FC = () => {

   :

  const invalid = useMemo(() => userName === '' || password === '', [userName, password]);

  return (
    <Page>
       :
      <Button title="Login" onPress={login} disabled={invalid} />
    </Page>
  );
};

export default rootPage('login', Component);
```

memoフックを使用します。
第二引数で渡した値が変更されたら第一引数の式が再評価されます。

入力されたらボタンを押せるようにする

Work

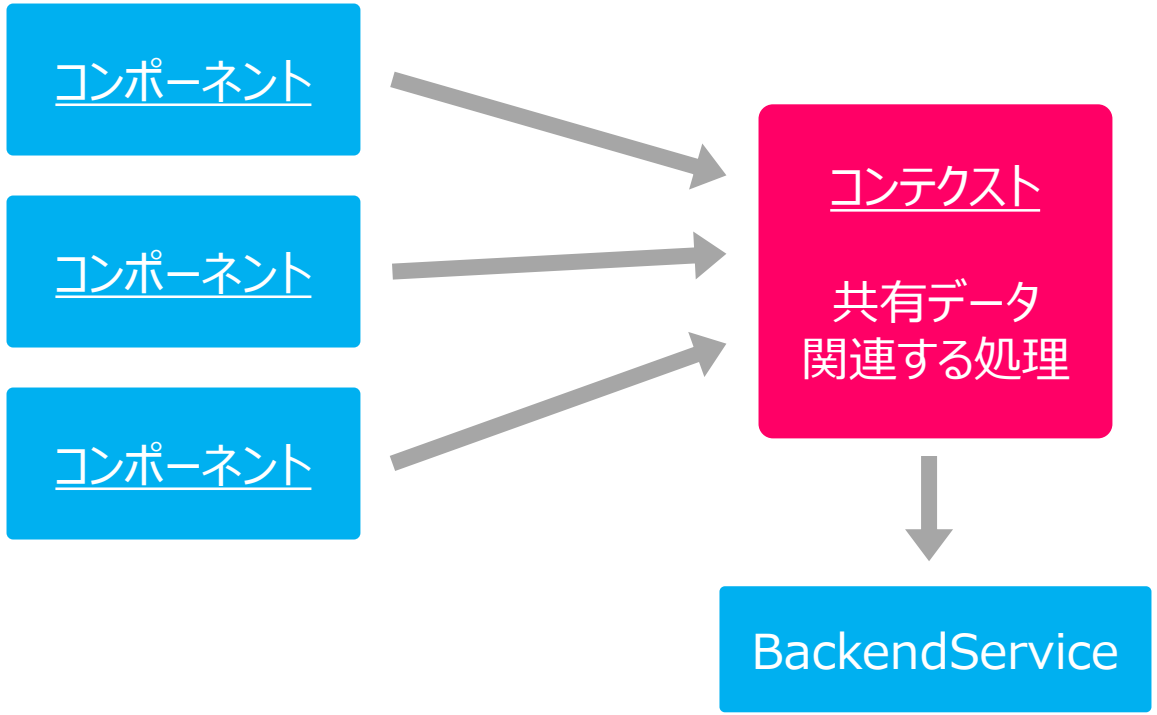
Login

```
import React, { useMemo, useState } from 'react';
 :

const Component: React.FC = () => {

   :

  const invalid = useMemo(() => userName === '' || password === '', [userName, password]);

  return (
    <Page>
       :
      <Button title="Login" onPress={login} disabled={invalid} />
    </Page>
  );
};

export default rootPage('login', Component);
```

memoフックを使用します。
第二引数で渡した値が変更されたら第一引数の式が再評価されます。

コンテクスト

ログイン状態のように複数のコンポーネントで共有したいデータは
コンテクストと呼ばれるReactが提供する仕組みを使用して保持します。
コンテクストには共有したいデータだけでなく、データに関連する処理を持たせることもできます。




```
const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <UserContextProvider>
      <Component />
    </UserContextProvider>
  );
};

export default App;
```

```
const Component: React.FC = () => {

  const userContext = useUserContext();

};
```

ContextProviderでコンテクストを使える状態にして、
各コンポーネントでフックでコンテクストを取得して使います。

ユーザ管理

ユーザ管理まわりのデータと処理をまとめたUserContextを作成しています。

- ログイン処理
- ログアウト処理
- ユーザ名
- ログイン状態

src/contextsディレクトリのUserContext.tsxファイルを確認しましょう。

UserContext

Reading

```tsx
import React, { useContext, useState } from 'react';
import { BackendService } from '../backend/BackendService';

export class AuthenticationFailedError {}

interface ContextValueType {
  login: (userName: string, password: string) => Promise<void | AuthenticationFailedError>;
  logout: () => Promise<void>;
  userName: string;
  isLoggedIn: boolean;
}

export const UserContext = React.createContext<ContextValueType>({} as ContextValueType);

export const useUserContext = () => useContext(UserContext);

export const UserContextProvider: React.FC = ({ children }) => {

  const [userName, setUserName] = useState<string>('');

  const contextValue: ContextValueType = {
    login: async (userName, password) => {
      try {
        await BackendService.login(userName, password);
        setUserName(userName);
      } catch (error) {
        if (error.status === 401) {
          return new AuthenticationFailedError();
        }
        throw error;
      }
    },
    logout: async () => {
      await BackendService.logout();
      setUserName('');
    },
    userName,
    isLoggedIn: userName !== '',
  };

  return <UserContext.Provider value={contextValue}>{children}</UserContext.Provider>;
};
```

コンテクスト、
コンテクストを取得するフック、
コンテクストプロバイダー
を実装してexportしています。

REST APIの呼び出し先を変更する

アプリからモックサーバのREST APIを呼び出せるようにします。

ローカルPCのIPアドレスを確認し、次のソースのIPアドレスを変更します。

backend/generated-rest-client/runtime.ts

```
 :

//export const BASE_PATH = "http://localhost:9080".replace(/¥/+$/, "");
export const BASE_PATH = "http://172.20.10.2:9080".replace(/¥/+$/, "");

const isBlob = (value: any) => typeof Blob !== 'undefined' && value instanceof Blob;
 :
```

REST APIの呼び出し先を変更する

Work

アプリからモックサーバのREST APIを呼び出せるようにします。

ローカルPCのIPアドレスを確認し、次のソースのIPアドレスを変更します。

backend/generated-rest-client/runtime.ts

```
:

//export const BASE_PATH = "http://localhost:9080".replace(/¥/+$/, "");
export const BASE_PATH = "http://172.20.10.2:9080".replace(/¥/+$/, "");

const isBlob = (value: any) => typeof Blob !== 'undefined' && value instanceof Blob;
:
```

REST APIを呼び出しログイン状態に応じた表示切替をする

UserContextを使えるようにしてログイン処理を実装します。

App

```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import RootNav from './components/pages/RootNav';
import { UserContextProvider } from './contexts/UserContext';

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <UserContextProvider>
      <RootNav />
    </UserContextProvider>
  );
};

export default App;
```

Login

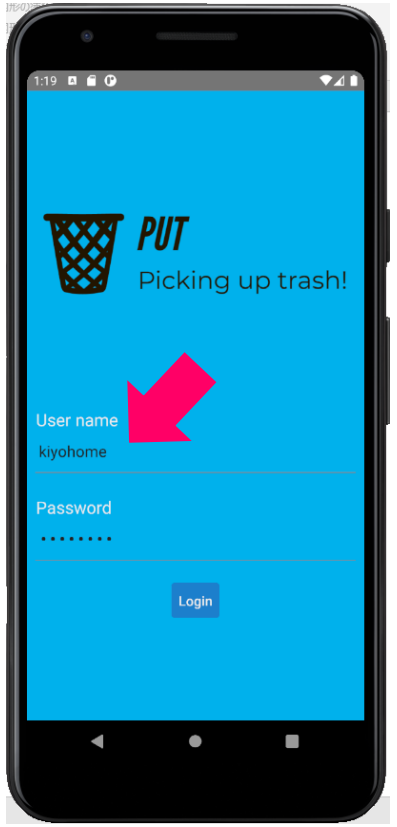
```
:
import { useUserContext } from '../../contexts/UserContext';
:

const Component: React.FC = () => {

  const userContext = useUserContext();
  const navigation = useNavigation();
  const [userName, setUserName] = useState<string>('');
  const [password, setPassword] = useState<string>('');

  const login = async () => {
    await userContext.login(userName, password);
    setUserName('');
    setPassword('');
    navigation.navigate('main');
  };

  return (
    :
  );
};

export default rootPage('login', Component);
```

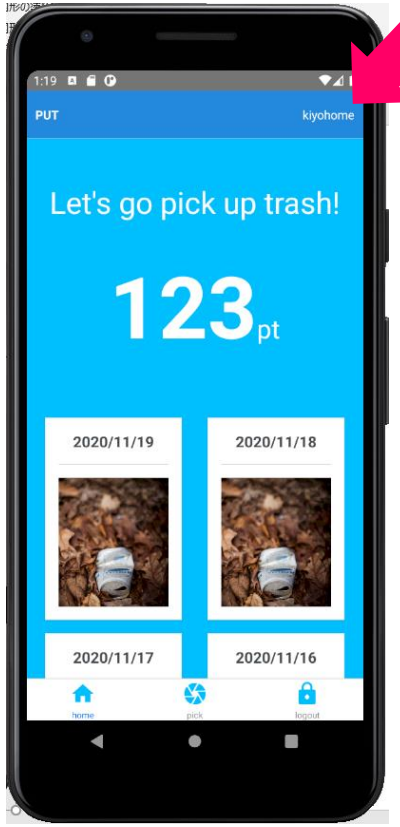
REST APIを呼び出しログイン状態に応じた表示切替をする

ログイン後の画面のヘッダにユーザ名を表示します。

MainPage

```
 :
import { ScrollView } from 'react-native-gesture-handler';
import { useUserContext } from '../../contexts/UserContext';
 :

const Page: React.FC<PropsWithChildren<object>> = ({ children }) => {

  const userContext = useUserContext();
  const name = userContext.isLoggedIn ? userContext.userName : 'guest';

  return (
    <>
      <Header
        leftComponent={{ text: 'PUT', style: { color: '#fff', fontWeight: '700' } }}
        rightComponent={{ text: name, style: { color: '#fff' } }}
      />
      <ScrollView contentContainerStyle={styles.page}>{children}</ScrollView>
    </>
  );
};

 :
```





REST APIを呼び出しログイン状態に応じた表示切替をする

Work

UserContextを使えるようにしてログイン処理を実装します。

App

```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import RootNav from './components/pages/RootNav';
import { UserContextProvider } from './contexts/UserContext';

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <UserContextProvider>
      <RootNav />
    </UserContextProvider>
  );
};

export default App;
```

Login

```
 :
import { useUserContext } from '../../contexts/UserContext';
 :

const Component: React.FC = () => {

  const userContext = useUserContext();
  const navigation = useNavigation();
  const [userName, setUserName] = useState<string>('');
  const [password, setPassword] = useState<string>('');

  const login = async () => {
    await userContext.login(userName, password);
    setUserName('');
    setPassword('');
    navigation.navigate('main');
  };

  return (
    :
  );
};

export default rootPage('login', Component);
```

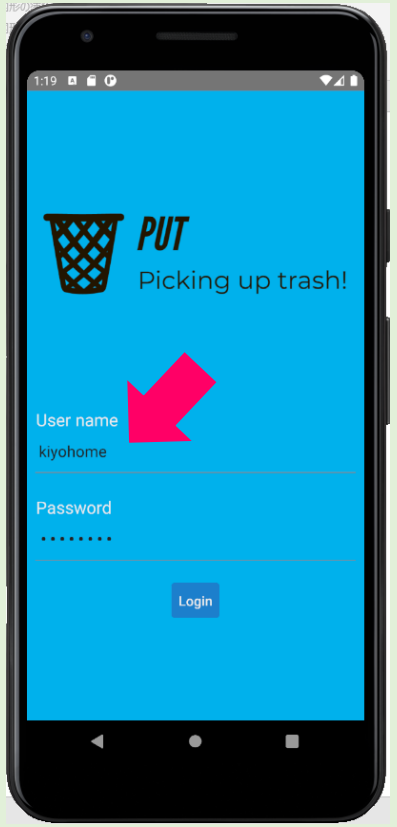
REST APIを呼び出しログイン状態に応じた表示切替をする

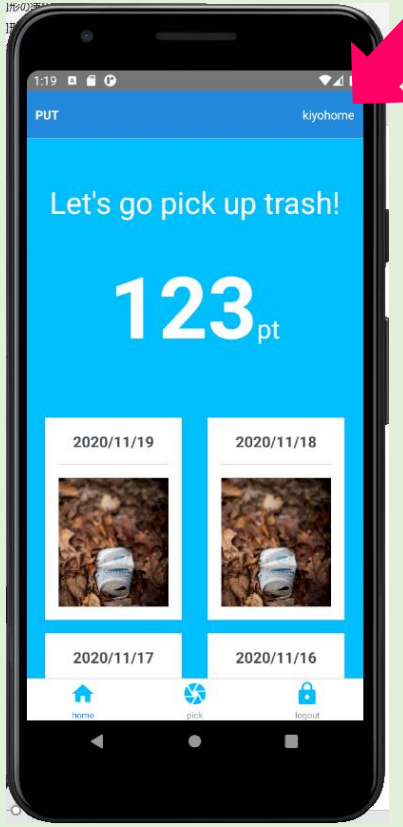
Work

ログイン後の画面のヘッダにユーザ名を表示します。

MainPage

```
 :
import { ScrollView } from 'react-native-gesture-handler';
import { useUserContext } from '../../contexts/UserContext';
 :

const Page: React.FC<PropsWithChildren<object>> = ({ children }) => {

  const userContext = useUserContext();
  const name = userContext.isLoggedIn ? userContext.userName : 'guest';

  return (
    <>
      <Header
        leftComponent={{ text: 'PUT', style: { color: '#fff', fontWeight: '700' } }}
        rightComponent={{ text: name, style: { color: '#fff' } }}
      />
      <ScrollView contentContainerStyle={styles.page}>{children}</ScrollView>
    </>
  );
};

 :
```



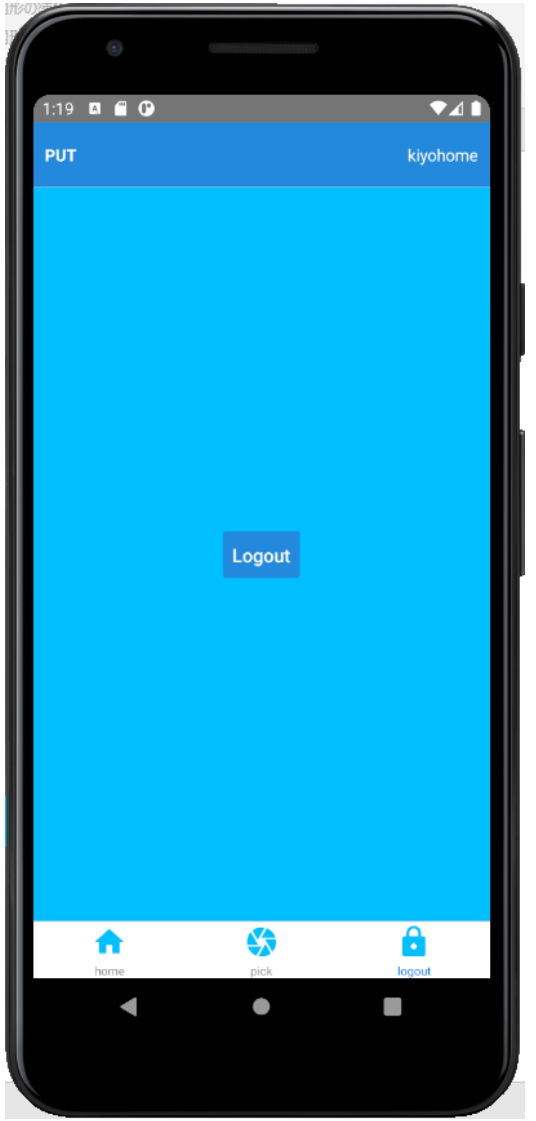


Logout

Logout

Logout

```
import { useNavigation } from '@react-navigation/native';
import React from 'react';
import { Button } from 'react-native-elements';
import { useUserContext } from '../../contexts/UserContext';
import { Page, mainPage } from './MainPage';

const Component: React.FC = () => {

  const userContext = useUserContext();
  const navigation = useNavigation();

  const logout = async () => {
    await userContext.logout();
    navigation.navigate('login');
  }

  return (
    <Page>
      <Button title="Logout" onPress={logout} />
    </Page>
  );
};

export default mainPage('logout', Component, 'lock');
```

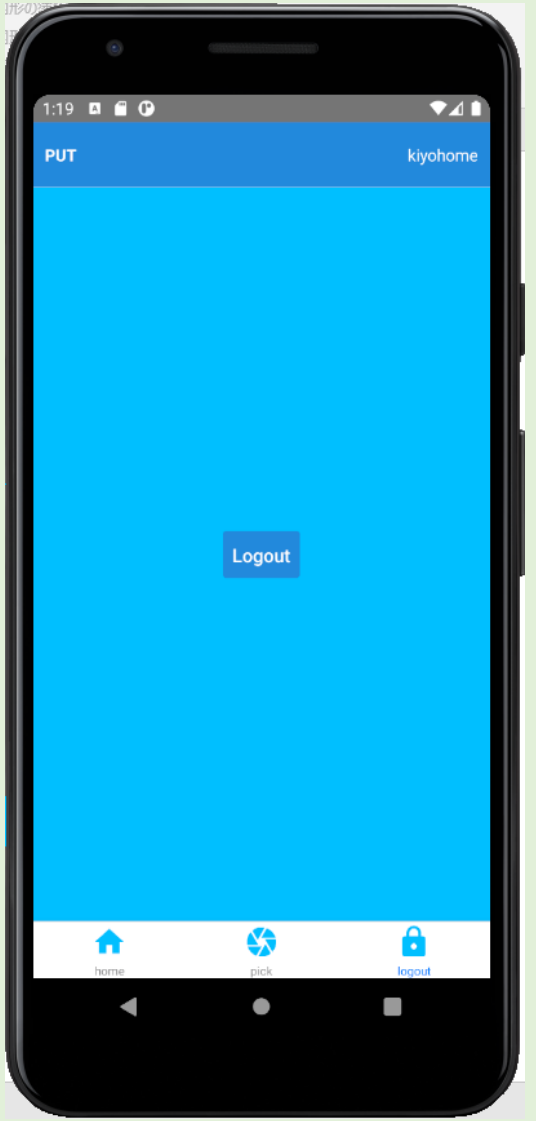


Logout

Work

Logout

```
import { useNavigation } from '@react-navigation/native';
import React from 'react';
import { Button } from 'react-native-elements';
import { useUserContext } from '../../contexts/UserContext';
import { Page, mainPage } from './MainPage';

const Component: React.FC = () => {

  const userContext = useUserContext();
  const navigation = useNavigation();

  const logout = async () => {
    await userContext.logout();
    navigation.navigate('login');
  }

  return (
    <Page>
      <Button title="Logout" onPress={logout} />
    </Page>
  );
};

export default mainPage('logout', Component, 'lock');
```



5分休憩

休憩中は質問OKです！

Home

ゴミ拾い管理

ゴミ拾い管理まわりのデータと処理をまとめたTrashContextを作成しています。

- ゴミ拾い一覧の取得処理
- ゴミ拾いの登録処理
- ゴミ拾い一覧
- ポイント

src/contextsディレクトリのTrashContext.tsxファイルを確認しましょう。

TrashContext

Reading

```tsx
import React, { useContext, useState } from 'react';
import { BackendService } from '../backend/BackendService';
import { Trash } from '../backend/generated-rest-client';

interface ContextValueType {
  getTrashList(): void;
  postTrash(trash: Blob): void;
  trashList: Trash[];
  point: number;
}

export const TrashContext = React.createContext<ContextValueType>({} as ContextValueType);

export const useTrashContext = () => useContext(TrashContext);

export const TrashContextProvider: React.FC = ({ children }) => {

  const [trashList, setTrashList] = useState<Trash[]>([]);

  const contextValue: ContextValueType = {
    getTrashList: () => {
      BackendService.getTrashList()
        .then(response => setTrashList(response));
    },
    postTrash: (trash) => {
      BackendService.postTrash(trash)
        .then(response => setTrashList([response, ...trashList]));
    },
    trashList,
    point: trashList.reduce((total, trash) => total + trash.point, 0),
  };

  return <TrashContext.Provider value={contextValue}>{children}</TrashContext.Provider>;
};
```

コンテクスト、
コンテクストを取得するフック、
コンテクストプロバイダー
を実装してexportしています。

Home

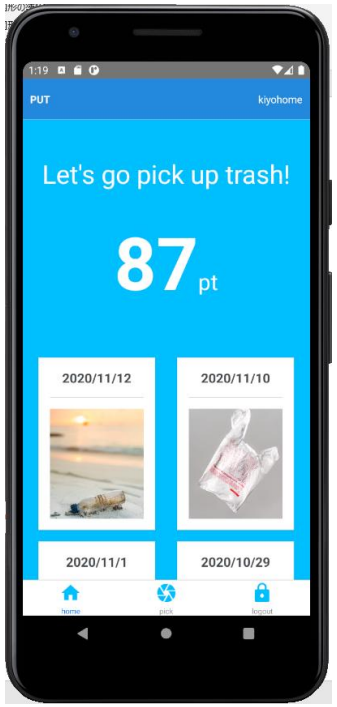
App

```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import RootNav from './components/pages/RootNav';
import { UserContextProvider } from './contexts/UserContext';
import { TrashContextProvider } from './contexts/TrashContext';

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <UserContextProvider>
      <TrashContextProvider>
        <RootNav />
      </TrashContextProvider>
    </UserContextProvider>
  );
};

export default App;
```

初期表示でデータ取得して表示するような場合は
Reactが提供するeffectフックを使用します。
第2引数に空配列を指定することで初期表示の時だ
け第1引数に指定した処理を実行します。

Home

```
import React, { useEffect } from 'react';
import { StyleSheet, View } from 'react-native';
import { Card, Text } from 'react-native-elements';
import { useTrashContext } from '../../contexts/TrashContext';
:

const Component: React.FC = () => {

  const trashContext = useTrashContext();

  useEffect(() => {
    (async () => {
      await trashContext.getTrashList();
    })();
  }, []);

  return (
    <Page>
      <Text style={styles.lead}>Let's go pick up trash!</Text>
      <Text style={styles.point}>
        {trashContext.point}
        <Text style={styles.unit}>pt</Text>
      </Text>
      <View style={styles.trashList}>
        {trashContext.trashList.map((trash, index) => (
          <Card key={index} containerStyle={styles.trash}>
            <Card.Title>{trash.date}</Card.Title>
            <Card.Divider />
            <Card.Image source={{ uri: trash.imageUrl }} />
          </Card>
        ))}
      </View>
    </Page>
  );
};

export default mainPage('home', Component, 'home');
```



Home

Work

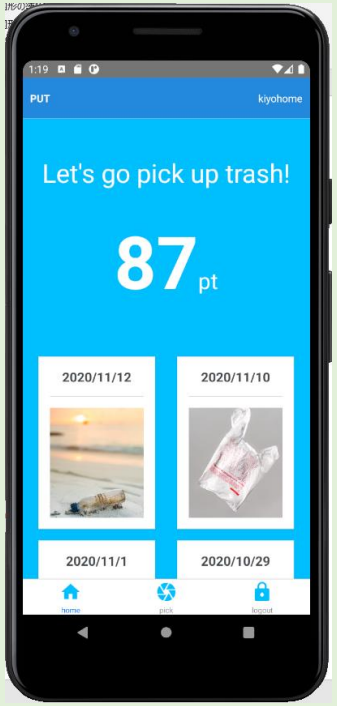
App

```
import React from 'react';
import { activateKeepAwake } from 'expo-keep-awake';
import RootNav from './components/pages/RootNav';
import { UserContextProvider } from './contexts/UserContext';
import { TrashContextProvider } from './contexts/TrashContext';

const App: React.FC = () => {

  if (__DEV__) {
    activateKeepAwake();
  }

  return (
    <UserContextProvider>
      <TrashContextProvider>
        <RootNav />
      </TrashContextProvider>
    </UserContextProvider>
  );
};

export default App;
```

初期表示でデータ取得して表示するような場合は
Reactが提供するeffectフックを使用します。
第2引数に空配列を指定することで初期表示の時だ
け第1引数に指定した処理を実行します。

Home

```
import React, { useEffect } from 'react';
import { StyleSheet, View } from 'react-native';
import { Card, Text } from 'react-native-elements';
import { useTrashContext } from '../../contexts/TrashContext';
:

const Component: React.FC = () => {

  const trashContext = useTrashContext();

  useEffect(() => {
    (async () => {
      await trashContext.getTrashList();
    })();
  }, []);

  return (
    <Page>
      <Text style={styles.lead}>Let's go pick up trash!</Text>
      <Text style={styles.point}>
        {trashContext.point}
        <Text style={styles.unit}>pt</Text>
      </Text>
      <View style={styles.trashList}>
        {trashContext.trashList.map((trash, index) => (
          <Card key={index} containerStyle={styles.trash}>
            <Card.Title>{trash.date}</Card.Title>
            <Card.Divider />
            <Card.Image source={{ uri: trash.imageUrl }} />
          </Card>
        ))}
      </View>
    </Page>
  );
};

export default mainPage('home', Component, 'home');
```



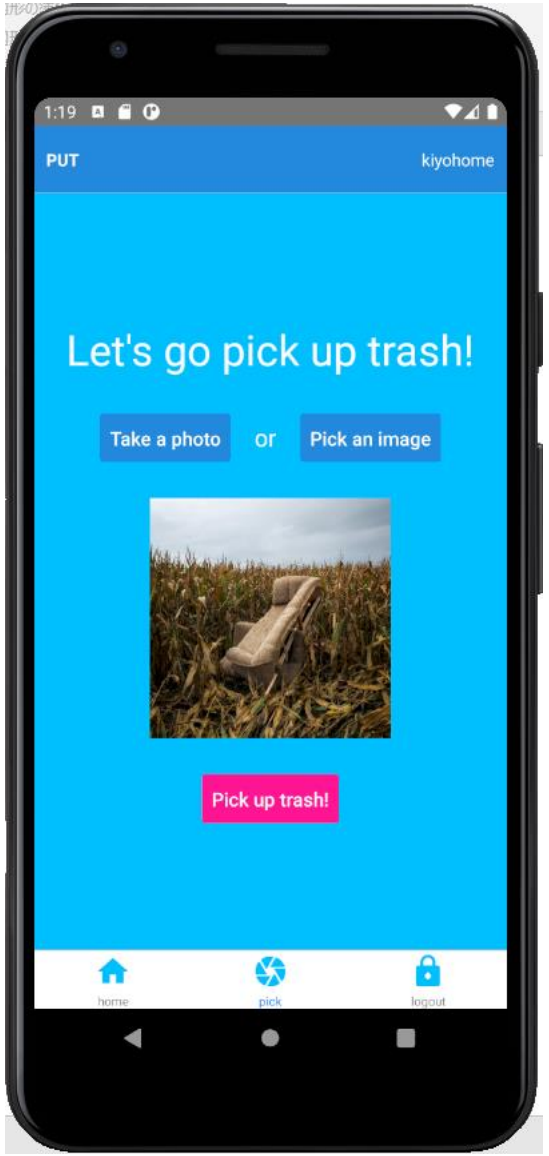
Pick

Pick

スマホのカメラで撮った写真、
またはライブラリにある写真を選択して
ゴミ拾い写真を登録します。

カメラ周りの処理を行う
expo-image-pickerというライブラリを使います。

ImagePicker
https://docs.expo.io/versions/latest/sdk/imagepicker/



ImagePickerを使ったカメラの操作

Picking

```
:
import * as ImagePicker from 'expo-image-picker'
import React, { useEffect, useState } from 'react';
:

const Component: React.FC = () => {

  const navigation = useNavigation();
  const [image, setImage] = useState<string>('');

  useEffect(() => {
    (async () => {
      const { status } = await ImagePicker.requestCameraPermissionsAsync();
      if (status !== 'granted') {
        alert('Sorry, we need camera roll permissions to make this work!');
      }
    })();
    (async () => {
      const { status } = await ImagePicker.requestCameraRollPermissionsAsync();
      if (status !== 'granted') {
        alert('Sorry, we need camera roll permissions to make this work!');
      }
    })();
  }, []);

  const takePhoto = async () => {

    const result = await ImagePicker.launchCameraAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });

    if (!result.cancelled) {
      setImage(result.uri);
    }
  };
```

カメラのアクセス権限チェック

カメラで写真をとる

```
  const pickImage = async () => {
    const result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });

    if (!result.cancelled) {
      setImage(result.uri);
    }
  };

  const pickUpTrash = () => {
    setImage('');
    navigation.navigate('home');
  }

  return (
    <Page>
      <Text style={styles.lead}>Let's go pick up trash!</Text>
      <View style={styles.select}>
        <Button title="Take a photo" onPress={takePhoto} />
        <Text style={styles.or}>or</Text>
        <Button title="Pick an image" onPress={pickImage} />
      </View>
      {image !== '' && <Image source={{ uri: image }} style={styles.image} />}
      <Button title="Pick up trash!" buttonStyle={styles.upload}
        onPress={pickUpTrash} disabled={image === ''} />
    </Page>
  );
};

export default mainPage('pick', Component, 'camera');
```

ライブラリにある写真を選択する

ImagePickerを使ったカメラの操作

Work

Picking

```
:
import * as ImagePicker from 'expo-image-picker'
import React, { useEffect, useState } from 'react';
:

const Component: React.FC = () => {

  const navigation = useNavigation();
  const [image, setImage] = useState<string>('');

  useEffect(() => {
    (async () => {
      const { status } = await ImagePicker.requestCameraPermissionsAsync();
      if (status !== 'granted') {
        alert('Sorry, we need camera roll permissions to make this work!');
      }
    })();
    (async () => {
      const { status } = await ImagePicker.requestCameraRollPermissionsAsync();
      if (status !== 'granted') {
        alert('Sorry, we need camera roll permissions to make this work!');
      }
    })();
  }, []);

  const takePhoto = async () => {

    const result = await ImagePicker.launchCameraAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });

    if (!result.cancelled) {
      setImage(result.uri);
    }
  };
```

カメラのアクセス権限チェック

カメラで写真をとる

```
  const pickImage = async () => {
    const result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });

    if (!result.cancelled) {
      setImage(result.uri);
    }
  };

  const pickUpTrash = () => {
    setImage('');
    navigation.navigate('home');
  }

  return (
    <Page>
      <Text style={styles.lead}>Let's go pick up trash!</Text>
      <View style={styles.select}>
        <Button title="Take a photo" onPress={takePhoto} />
        <Text style={styles.or}>or</Text>
        <Button title="Pick an image" onPress={pickImage} />
      </View>
      {image !== '' && <Image source={{ uri: image }} style={styles.image} />}
      <Button title="Pick up trash!" buttonStyle={styles.upload}
        onPress={pickUpTrash} disabled={image === ''} />
    </Page>
  );
};

export default mainPage('pick', Component, 'camera');
```

ライブラリにある写真を選択する

5分休憩

休憩中は質問OKです！

クロージング

モバイルハンズオンはいかがでしたか？

モバイルの作り方を体験いただけましたでしょうか？

Fintanでモバイルのノウハウを公開しています。

モバイルアプリケーションのセキュリティと認証

https://fintan.jp/?p=5950

他にも現場で活用しているコンテンツやノウハウ、

技術ネタのブログも公開していますのでぜひ覗いてみてください。

友人に紹介いただいたり、勉強会等でご活用ください。

https://fintan.jp/

サービス開発エンジニア体験

サービス/プロダクトの開発に欠かせない
アプリ開発とDevOpsを体験してみませんか？

10月　SPAハンズオン
11月　APIハンズオン
12月　モバイルハンズオン
1月　DevOpsハンズオン
2月　チームで腕試しハッカソン

Aizurage

connpassのグループです。
https://tidev-aizu.connpass.com/

TISの会津拠点のエンジニアが中心になって、
エンジニア交流を目的にハンズオンや勉強会をやっています。

興味がありましたらグループのメンバーになってください。
メンバー＝グループのスタッフではないので安心してください。
グループのメンバーはTwitterのフォロワーのようなイメージです。

「メンバーになると、グループのイベントが作成されると通知がきたり、
トップページのおすすめイベントに表示されるので、
興味のあるイベントを見逃すことが少なくなります。」

We’re Hiring!

技術力で活躍したいエンジニアを募集しています！
まずは東京、大阪で経験を積んでそのままでもいいし、
U/Iターンで会津若松でもいいし、一緒に働きませんか？

エントリーページ
https://www.tis.co.jp/recruit/

地図はいらすとやを使用しています。
https://www.irasutoya.com/

■申込方法

まずはTIS 新卒採用サイトにて、エントリーをお願いします！

TIS 新卒採用サイト

https://job.axol.jp/cr/s/tis_22/mypage/login

エントリー時の以下のアンケート項目については、以下のようにお答えください。

・TISを知ったきっかけの詳細を教えてください。

→TIS主催のサービス開発エンジニア体験（Aizurage）

翌営業日迄にご登録のE-mailアドレスに当社マイページをご案内いたします。

※12/27～1/5は対応期間外となります。ご注意ください。

マイページからイベントにお申し込みください。

チーム開発コース 以外のイベントもマイページでご案内します。

ご応募お待ちしています！

アンケート

今後の改善に活用したいのでアンケートへのご協力をお願いします。

アンケートのURLはZoomのチャットで連絡します。


Thank you
Service Dev Engineer TAIKEN