|  |  |  |
| --- | --- | --- |
| オラクルGoのベース技術Oracle JETを触ってみよう  ハンズオン・チュートリアル | | |
|  | Creation Date: |  |
|  | Last Update: |  |
|  | Version: | Folk-1.0 |
| C:\WINNT\Profiles\amisu.000\ﾃﾞｽｸﾄｯﾌﾟ\oracle.bmp | | |

目次

[1 はじめに 4](#_Toc464735717)

[1.1 この資料の表記方法 4](#_Toc464735718)

[2 Oracle JETとは 5](#_Toc464735719)

[2.1 Oracle JETに組み込まれているオープン・ソース・テクノロジ 5](#_Toc464735720)

[RequireJSとは 5](#_Toc464735721)

[Knockoutとは 6](#_Toc464735722)

[2.2 オラクルが提供するOracle JETの機能 7](#_Toc464735723)

[3 Oracle JET開発環境の構築 8](#_Toc464735724)

[3.1 教材用ソースコードの入手 8](#_Toc464735725)

[3.2 NetBeans IDEを用いた開発と設定 8](#_Toc464735726)

[3.3 Webブラウザを使用したJavaScriptのデバッグ 9](#_Toc464735727)

[4 Oracle JETを使用した画面の作成 11](#_Toc464735728)

[4.1 ビューの編集 12](#_Toc464735729)

[4.2 ビュー・モデルの編集 13](#_Toc464735730)

[4.3 データ・バインドの設定 14](#_Toc464735731)

[5 MCS JavaScript SDKの導入 16](#_Toc464735732)

[5.1 JavaScript SDKのダウンロード 16](#_Toc464735733)

[5.2 RequireJSのパス・マッピング 16](#_Toc464735734)

[5.3 モバイル・バックエンドにアクセスするモジュールの作成 18](#_Toc464735735)

[NetBeans 8.1を使用している場合のモジュールの作成 18](#_Toc464735736)

[mbeModule.jsのソースの編集 18](#_Toc464735737)

[6 ユーザー認証の実装 21](#_Toc464735738)

[6.1 ログイン/ログアウト処理の実装 21](#_Toc464735739)

[6.2 ビュー・モデルに対するモジュールの組み込み 21](#_Toc464735740)

[6.3 ビューへのデータ・バインド 22](#_Toc464735741)

[6.4 稼働確認 24](#_Toc464735742)

[7 カスタムAPIの呼び出しとチャートの表示 25](#_Toc464735743)

[7.1 カスタムAPI呼び出しの実装 25](#_Toc464735744)

[7.2 Chartコンポーネントのモジュールのロードとデータ変換処理の実装 26](#_Toc464735745)

[7.3 ビューへのバー・チャートの追加 27](#_Toc464735746)

[7.4 稼働確認 27](#_Toc464735747)

[8 オラクルGoの実装 28](#_Toc464735748)

# はじめに

Oracle JET （JavaScript Extension Toolkit）は、オラクルおよびオープン・ソースが提供するJavaScriptライブラリのコレクションです。中級から上級のJavaScript開発者がOracle JETを使用すると、JavaScript、HTML5およびCSSに基づいたクライアント側のWebアプリケーションおよびハイブリッド・モバイル・アプリを、できるだけシンプルかつ効率的に作成できます。Oracle JET自身もオープン・ソースとして公開されており、Universal Permissive License (UPL) 1.0ライセンスのもとで使用することができます。

このハンズオンでは、Oracle Mobile Cloud Service（以下MCS）上に構築されたモバイル・バックエンドに対するフロントエンドを、Oracle JETを使用して作成する流れを説明します。

## この資料の表記方法

この資料で使用している表記方法は次のとおりです。

|  |  |  |
| --- | --- | --- |
| 表記方法 | 説明 | 例 |
| 固定幅フォント | コマンドやサンプル・コード、URL | grunt serve |
| 太字 | ファイル名 | index.html |
| 斜体 | 使用する環境に合わせて変更する部分 | https://<ホスト名>/ |

この資料は主にWindowsの構文を用いていますが、MacまたはLinux環境でのファイル名およびディレクトリ名などは特に明記がない場合は同一です。

# Oracle JETとは

オラクルはjQueryやRequireJS、Knockoutなど、すでに広く使用されているオープン・ソースのJavaScriptライブラリを活用し、エンタープライズ・アプリケーションをHTML5/JavaScript/CSSで構築するために必要なツールキットをOracle JETとして開発し、提供を開始しました。

ここではOracle JETの概要と提供する主な機能を紹介します。

## Oracle JETに組み込まれているオープン・ソース・テクノロジ

Oracle JETは広く利用されているオープン・ソース・テクノロジをベースとして開発されたJavaScript開発者のためのツールキットです。

表 ‑1: Oracle JETに組み込まれている主なオープン・ソース・テクノロジ

|  |  |
| --- | --- |
| ライブラリ名 | 説明 |
| jQuery | HTMLドキュメントのDOMの操作やイベント処理、アニメーション、Ajaxコールなどを簡素化するJavaScriptライブラリ  詳細: http://jquery.com/ |
| jQuery UI | jQuery上で構築されたUIコンポーネント群。Oracle JETが提供するUIコンポーネントはjQuery UIが提供するコア・ライブラリをベースに実装  詳細: http://www.jqueryui.com/ |
| RequireJS | JavaScriptライブラリの参照および依存性の解決、遅延ロードなどの機能を提供するJavaScriptフレームワーク  詳細: http://www.requirejs.org/ |
| RequireJS CSS | RequireJSのプラグイン。RequireJSのテクノロジを使用してCSSをロード |
| Knockout | Model-View-ViewModel（MVVM）パターンに基づいた双方向データ・バインドを実現するJavaScriptフレームワーク  詳細: <http://www.knockoutjs.com/> |
| Sass | 変数や式などスクリプト・ライクな記法が可能なCSSのプリプロセッサ。Oracle JETはSassのSCSS構文を使用  詳細: http://sass-lang.com/ |

### RequireJSとは

JavaScriptを用いてフロントエンドを開発する場合、通常はHTMLドキュメントに使用するJavaScriptのソース・ファイルをscriptタグで記述します。ただし、使用するJavaScriptソース・ファイルを単純に羅列すればよいというわけではありません。開発者はJavaScriptライブラリ間の依存関係に応じて、scriptタグを適切な順番で記述しなければいけません。また、scriptタグで指定されたJavaScriptソースは、実際に使用されるかどうかに関わらずすべてロードされるので、パフォーマンスに影響を与えることがあります。

RequireJSはJavaScriptライブラリ間の依存関係を解決して、適切にロードするための仕組みを提供するフレームワークです。RequireJSを使用すれば、機能ごとにJavaScriptソースを分割してモジュール化でき、実行時に必要になったタイミングでロードされるので、保守性とパフォーマンスの両方に優れたアプリケーションの開発に役立ちます。

例えば、Oracle JETが提供しているWebページのひな形では、JavaScriptライブラリのロードはRequireJSを使用してコード 2‑1のように記述しています。

コード ‑1

<script type="text/javascript" src="js/libs/require/require.js">  
</script>

<script type="text/javascript" src="js/main.js"></script>

最初のscriptタグでRequireJSのソースを指定し、次のscriptタグでRequireJSが必要とする構成情報と最初に実行されるロジックが記述されたJavaScriptソースを指定しています。さらにのように、1行で記述することも可能です。1行で記述する場合は、最初に実行されるロジックが記述されたJavaScriptソース（拡張子を省略したもの）をdata-main属性で指定します。

コード ‑2

<script data-main="js/main" src="js/libs/require/require.js"></script>

### Knockoutとは

Knockoutは、Model-View-ViewModel （MVVM）パターンに基づいてアプリケーションを構築する際に使用されるJavaScriptフレームワークです。MVVMパターンでは、アプリケーションを三種類のコンポーネントに分割して設計し、作成します。

表 ‑2: MVVMパターン

|  |  |
| --- | --- |
| コンポーネント | 説明 |
| モデル（Model） | RESTサービスなどの業務ロジック呼び出しを実装したJavaScriptオブジェクト |
| ビュー（View） | ユーザー・インタフェース。Oracle JET ではHTMLドキュメントとして記述 |
| ビュー・モデル（ViewModel） | ビューで表示・編集するデータとその操作を実装したJavaScriptオブジェクト |

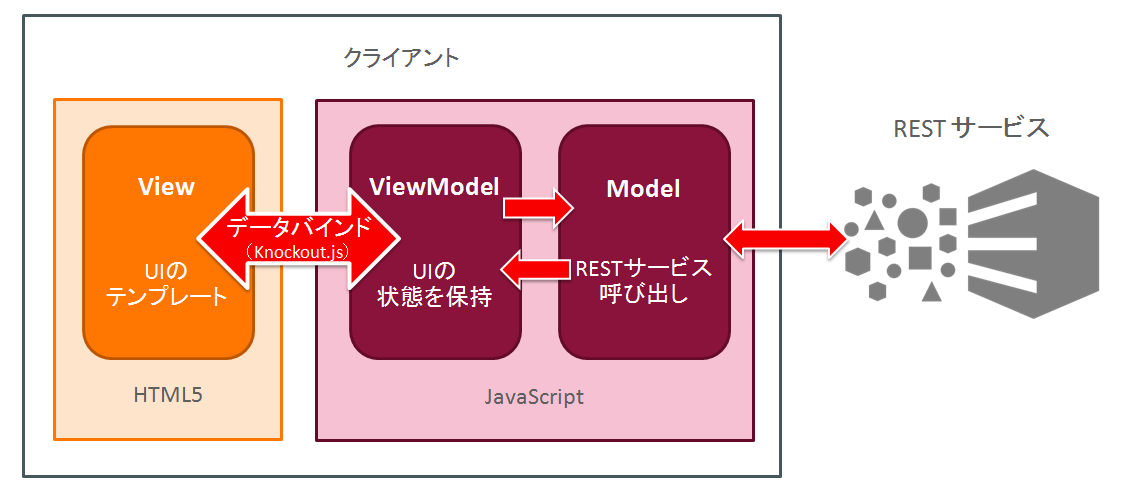


図 ‑1: MVVMパターン

KnockoutはMVVMパターンにおけるビューとビュー・モデルの宣言的なデータ・バインディングの仕組みを提供しています。Knockoutを活用することで、最小限のコードでインタラクティブな画面を開発できます。

## オラクルが提供するOracle JETの機能

オラクルは前述のオープン・ソースのJavaScriptライブラリを活用しつつ、エンタープライズ・アプリケーション開発で必要となる次の機能をOracle JETに追加しています。

* RESTサービスの呼び出しをシンプルに実装するためのCommon Model & Collection API
* 数値や日付、時刻などの書式を変換するためのコンバータ
* 入力されたデータの妥当性を検証するためのバリデータ
* 国際化およびローカライゼーション
* jQuery UIのAPIに基づいて構築されたUIコンポーネント
* Knockoutのデータ・バインディングに基づくWebページのモジュール化
* Yeomanベースの開発ライフサイクルをサポートするツール群

# Oracle JET開発環境の構築

## 教材用ソースコードの入手

Githubから教材用のソースコードを取得します。ブラウザにて以下にアクセスします

**https://github.com/JETs-JP/oraclego-JET**

リポジトリ「**JETs-JP/oraclego-JET**」のトップ画面が表示されたら、Clone or Download > download ZIPをクリックします。Zipファイルがダウンロードされます。

ダウンロードが完了したら、以下のフォルダに当該Zipファイルを展開します。なお本書のwordファイルも同梱しています。ご利用ください

パス要確認

C:\HOL\JETProject\myapp

## NetBeans IDEを用いた開発と設定

Oracle JETは特定のIDEに依存していません。UTF-8に対応したテキスト・エディタやHTML5やJavaScriptに対応したIDEを自由に選択できます。

今回のハンズオンではNetBeans IDEを使用して開発を進めます。NetBeans IDEはHTML5やJavaScriptのサポートが充実しており、Oracle JETを使用したアプリケーション開発のためのプラグインも提供されています。NetBeans IDEを使用してOracle JETのアプリケーションを開発するためには、次の手順で設定します。

1. NetBeans IDEを起動します。起動されたら、画面左上にあるプロジェクト・ウィンドウを右クリックし、コンテキスト・メニューから「プロジェクトを開く」を選択します。

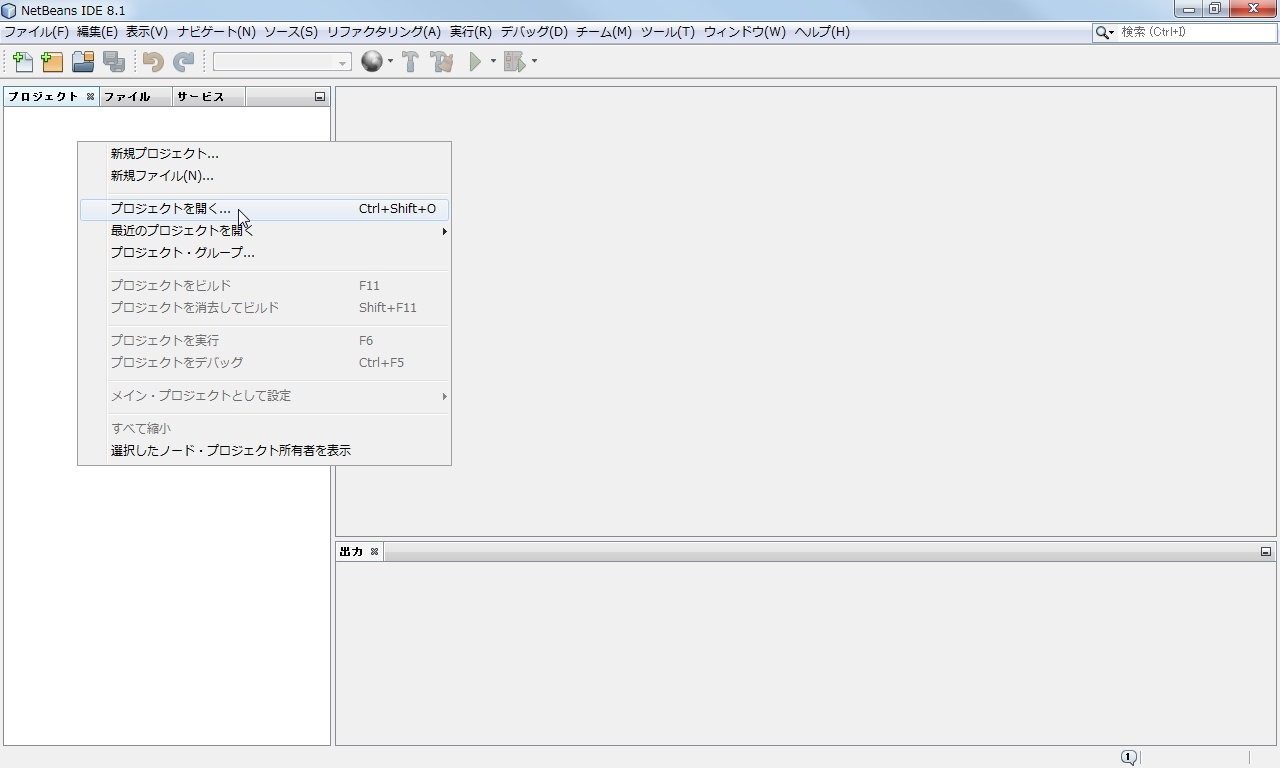


図 ‑1: NetBeans IDEのプロジェクト・ウィンドウを右クリックし「プロジェクトを開く」を選択

1. 「プロジェクトを開く」ウィンドウが表示されたら、C:\HOL\JETProject\myappを選択し、「プロジェクトを開く」ボタンをクリックします。指定したディレクトリ内のファイルをスキャンするため、環境によっては時間がかかることがあります。

この辺記述怪しい

1. ファイルのスキャンが完了すると、NetBeans IDEのプロジェクト・ウィンドウに「oraclejet-app」というプロジェクトが表示されます。

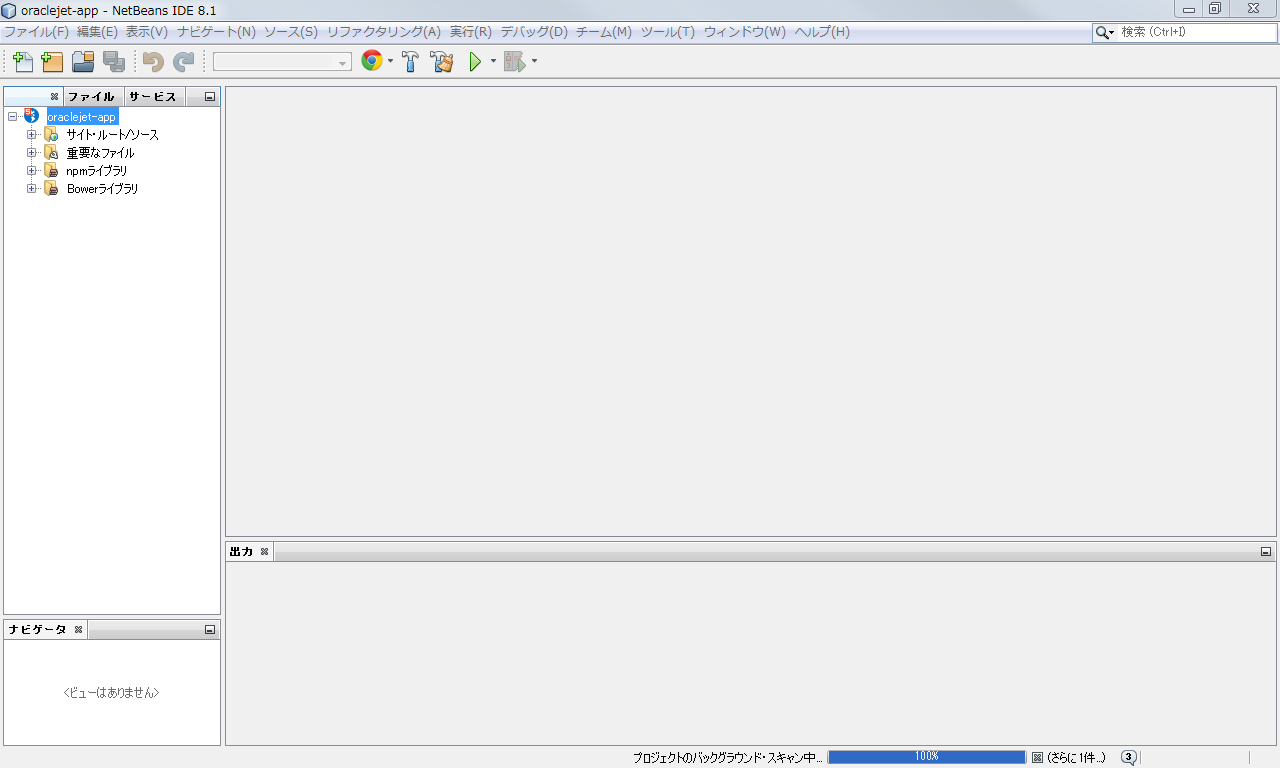


図 3‑2: oraclejet-appプロジェクトが作成された状態

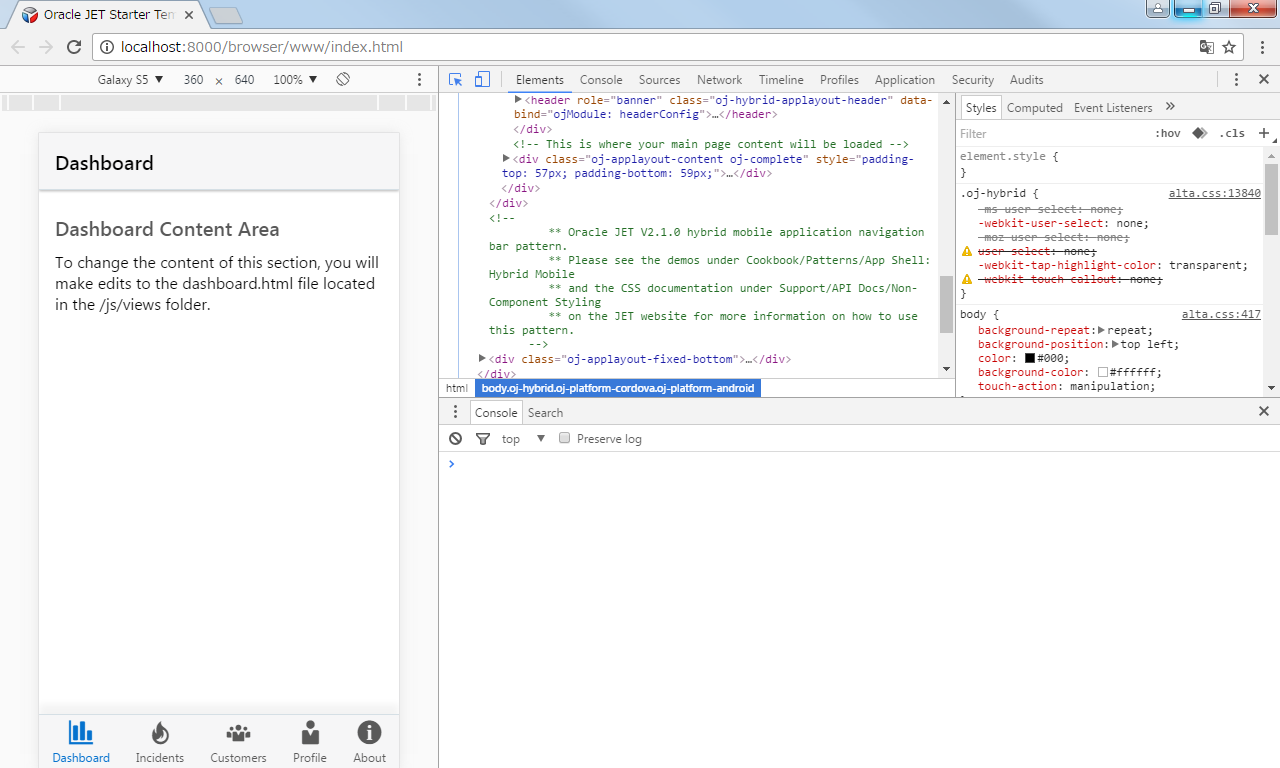
## Webブラウザを使用したJavaScriptのデバッグ

Google ChromeやApple Safari、Mozilla Firefoxには開発者向けのツールが組み込まれています。Google Chromeの場合、デバッグしたいアプリケーションを開いた状態で、アドレスバーの右にある「Google Chromeの設定」アイコンをクリックし、メニューから「その他のツール」→「デベロッパー ツール」を選択すると、「Developer Tools」ウィンドウが表示されます。



図 ‑4: Google Chromeのデベロッパー・ツールの起動

「Developer Tools」ウィンドウの左上にある「Toggle Device Toolbar」アイコンを選択するとのモバイル端末による画面をシミュレートできます。また、「Developer Tools」ウィンドウの「Console」タブ・ページには、JavaScriptのログ出力（console.logの出力結果）やエラー・メッセージが表示されます。



「Console」タブ

「Toggle Device Toolbar」アイコン

「Device Toolbar」

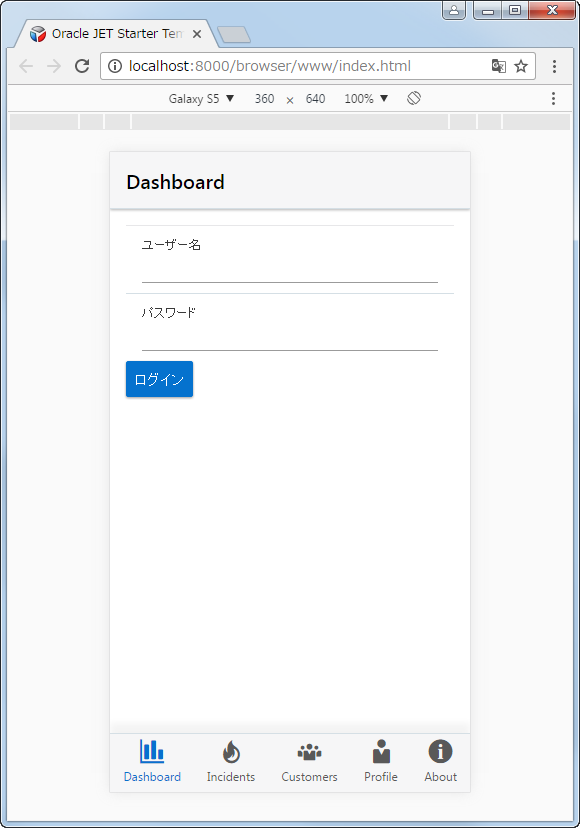
図 ‑5: Google Chromeの「Developer Tools」ウィンドウ

# Oracle JETを使用した画面の作成

この章では、Oracle JETが提供するUI部品を使用して画面を作成する基本的な流れを説明します。最初に作成するのは、ユーザーを認証するための入力フォームです。

今回は、次の４つのUIコンポーネントを使用します。

* Form Layout
* Input Text （ojInputText）
* Input Password （ojInputPassword）
* Button （ojButton）



Input Password (ojInputText)

Form Layout

Input Text (ojInputText)

Button (ojButton)

図 ‑1: この章で作成する画面

## ビューの編集

ここではC:\HOL\JETProject\myapp\src\js\views\dashboard.htmlを編集します。

コード ‑1: dashboard.htmlの変更前 （網掛けした4行をコメントアウト）

<!--

Copyright (c) 2014, 2016, Oracle and/or its affiliates.

The Universal Permissive License (UPL), Version 1.0

-->

<div>

<div class="oj-applayout-fixed-top">

<!--

\*\* Oracle JET V2.1.0 hybrid mobile application header pattern.

\*\* Please see the Oracle JET Cookbook App Shell: ...

\*\* more information on how to use this pattern.

-->

<header role="banner" class="oj-hybrid-applayout-header"  
 data-bind="ojModule: headerConfig">

</header>

</div>

<!-- This is where your main page content will be loaded -->

<div class="oj-applayout-content">

<div role="main" class="oj-hybrid-applayout-content">

<div class="oj-hybrid-padding">

<h3>Dashboard Content Area</h3>

<div>

To change the content of this section,...

</div>

</div>

</div>

</div>

</div>

コード ‑2: dashboard.htmlに追加するHTML（コメントアウトした行の後に追加）

<div>

<h3>ユーザー認証</h3>

<div class="oj-form-layout">

<div class="oj-form">

<div class="oj-flex">

<div class="oj-flex-item">

<label for="username">ユーザー名</label>

<input id="username"/>

</div>

</div>

<div class="oj-flex">

<div class="oj-flex-item">

<label for="password">パスワード</label>

<input id="password"/>

</div>

</div>

</div>

</div>

<button>ログイン</button>

</div>

パス要確認

**C:\HOL\JETProject\myapp\src\index.html**にアクセスして確認します（図 4‑2）

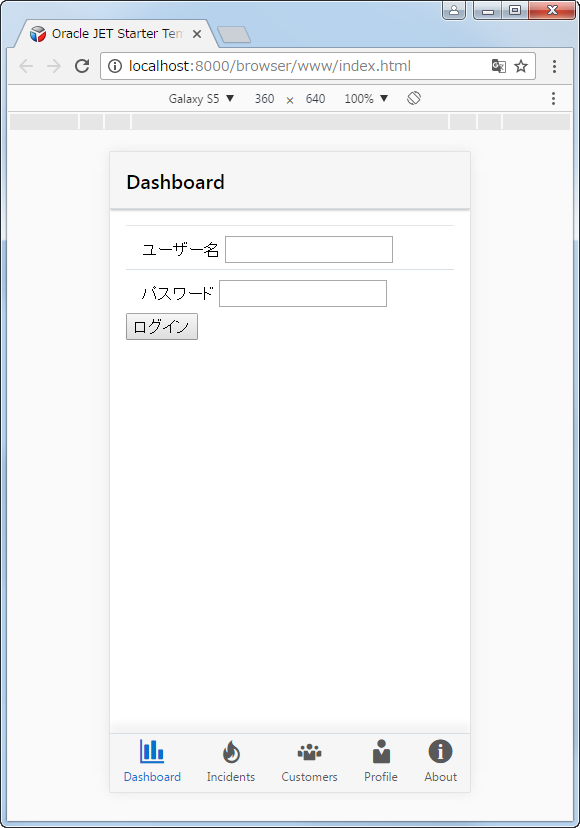


図 ‑2: dashboard.htmlの編集後

今回使用するForm Layoutは、Oracle JETが事前に定義したCSSスタイル（oj-xxxクラス）をHTMLタグに適用することで利用できます。Input TextおよびInput Password、Buttonの３つコンポーネントは、Knockoutが提供するカスタム・データ・バインドの仕組みをとおしてビューに適用されます。そのため、この段階では「ユーザー名」や「パスワード」の入力フィールドや「ログイン」ボタンには、まだUIコンポーネントは適用されていません。

備考: Form Layoutの利用例が、Oracle Cookbookの次のページで公開されています:  
<http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html?component=forms&demo=formvertical>

## ビュー・モデルの編集

ここでは、C:\HOL\JETProject\myapp\src\js\viewModels\dashboard.jsを編集します。dashboard.jsに追加する内容は次のとおりです。

* ビューに追加するUIコンポーネントを定義したモジュールのロード
* InputTextおよびInputPasswordを定義したモジュール: ojs/ojinputtext
* Buttonを定義したモジュール: ojs/ojbutton
* UIコンポーネント入力された値を保持するビュー・モデルのプロパティ
* ユーザー名: usernameValue
* パスワード: passwordValue
* ボタンがクリックされた時に実行されるメソッド: login

コード ‑3: dashboard.jsの編集 （網掛けした部分を追加）

define(['ojs/ojcore', 'knockout', 'jquery', 'appController'  
 , 'ojs/ojinputtext', 'ojs/ojbutton'],

function(oj, ko, $, app) {

function DashboardViewModel() {

var self = this;

self.usernameValue = ko.observable(); // 入力されたユーザー名の値を保持

self.passwordValue = ko.observable(); // 入力されたパスワードの値を保持

// 「ログイン」ボタンがクリックされた時に実行されるメソッド

self.login = function (data, event) {

window.alert(self.usernameValue() + 'さん、こんにちは!');

}

// 以下省略

usernmaeValueとpasswordValueにはko.observable()という関数がセットされています。これによってusernmaeValueとpasswordValueの値Knockoutによって常に監視され、変更はビューからビュー・モデルへ、またはビュー・モデルからビューへ即座に伝播されます。

関数loginの中で使用されているように、ko.observable()がセットされたプロパティの現行の値を取得する場合は、引数なしで呼び出します。また、新しい値をセットする場合は、その値を引数に指定します（例: self.username('Jane')）。

## データ・バインドの設定

ビュー・モデルのJavaScriptソースの編集が終わったら、ビューとバインドします。dashboard.htmlに追加する内容は次のとおりです。

* ユーザー名の入力フィールド（id="username"）にInput Text (ojInputText) をバインド（コード 4‑4）
* パスワードの入力フィールド（id="password"）にInput Password (ojInputPassword) をバインド（コード 4‑5）
* 「ログイン」ボタンにButton（ojButton）をバインドしクリック時loginメソッドを実行（コード 4‑6）

コード ‑4: ユーザー名の入力フィールドにInput Text (ojInputText) をバインド

<input id="username"  
data-bind="ojComponent: {  
 component:'ojInputText',  
 value: usernameValue  
}"/>

コード ‑5: パスワードの入力フィールドにInput Password (ojInputPassword) をバインド

<input id="password"

data-bind="ojComponent: {

component:'ojInputPassword',

value: passwordValue

}"/>

コード ‑6: 「ログイン」ボタンにButton（ojButton）をバインドしクリック時loginメソッドを実行

<button data-bind="click: login,

ojComponent: {

component:'ojButton',

chroming: 'full'

パス要確認

}">ログイン</button>

**C:\HOL\JETProject\myapp\src\index.html**をリロードして確認します（図 4‑3）

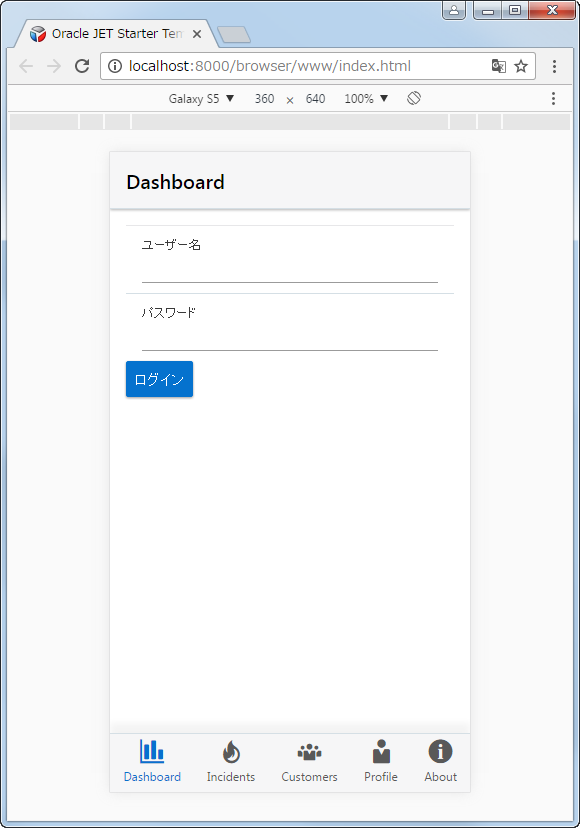


図 ‑3: 実行画面

# MCS JavaScript SDKの導入

MCSは、モバイル・アプリからMCS上の各種リソースへのアクセスを簡素化するためのクライアントSDKを提供しています。オラクルが提供しているクライアントSDKは次の５種類です。

* iOS SDK （Objective-CやSwiftによるiOSネイティブ・アプリ開発用）
* Android SDK （Android JavaによるAndroidネイティブ・アプリ開発用）
* Windows SDK （C#やVisual BasicなどによるWindowsネイティブ・アプリ開発用）
* Cordova SDK （Apache CordovaとJavaScriptによるハイブリッド・モバイル・アプリ開発用）
* JavaScript SDK （JavaScriptによるHTML5アプリ開発用）

今回はJavaScript SDKを利用します。ダウンロードし、Oracle JETを使用したアプリケーションから利用するために必要な設定について説明します。

## JavaScript SDKのダウンロード

1. C:\HOL\JETProject\myapp\src\js\libsディレクトリにmcs-sdkディレクトリを作成します。
2. 教材用のソースコードを取得します。ブラウザにてOracle Documents Cloud（オラクルのファイル共有クラウド）にアクセスします  
   [**https://goo.gl/mDm0SJ**](https://goo.gl/mDm0SJ)*(アクセスコード: CDD2016-JET）*
3. ZIPファイルmcs-js-sdk-v16.2.3.1.zipをダウンロードします。適当なフォルダに保存します。
4. mcs-js-sdk-v16.2.3.1.zipを解凍すると、oracle\_mobile\_js\_sdkディレクトリが生成されます。次の２つのJavaScriptソース・ファイルを手順1で作成したフォルダC:\HOL\JETProject\myapp\src\js\libs\mcs-sdkの中にコピーします。

* **mcs.js** （開発用）
* **mcs.min.js** （本番用; 改行などを取り除き、ファイル・サイズを小さくしたもの）

## RequireJSのパス・マッピング

main.jsではrequirejs.configがコールされています。このrequirejs.configの引数には、アプリが必要とするJavaScriptライブラリのソースを適切にロードするために必要な設定情報を指定します。

表 ‑1

|  |  |
| --- | --- |
| プロパティ名 | 設定する値 |
| baseUrl | JavaScriptソース・ファイルの格納先のベースとなるディレクトリ名 |
| paths | JavaScriptライブラリのソース・ファイルまたはディレクトリ・パスを名前解決するためのマッピング。ソース・ファイルの拡張子は省略できる。  'knockout': 'libs/knockout/knockout-3.4.0.debug' というエントリは、js/libs/knockout/knockout-3.4.0.debug.jsにknockoutという名前でアクセスできることを表している |
| shim | RequreJSのモジュール定義に対応していないライブラリの依存関係を定義する際に使用 |

今回は、ダウンロードしたMCSのJavaScript SDKのソースをロードするためにpathsプロパティにコード 5‑1のようにmcsというエントリを追加します。この変更によって、mcsという名前を指定すればMCS のJavaScript SDKのJavaScript ソースjs\libs\mcs-sdk\mcs.jsをロードできるようになります。

コード ‑1: main.jsの編集 （網掛けされた部分を追加）

requirejs.config(

{

baseUrl: 'js',

// Path mappings for the logical module names

paths:

//injector:mainReleasePaths

{

'knockout': 'libs/knockout/knockout-3.4.0.debug',

'jquery': 'libs/jquery/jquery-3.1.0',

'jqueryui-amd': 'libs/jquery/jqueryui-amd-1.12.0',

'promise': 'libs/es6-promise/es6-promise',

'hammerjs': 'libs/hammer/hammer-2.0.8',

'ojdnd': 'libs/dnd-polyfill/dnd-polyfill-1.0.0',

'ojs': 'libs/oj/v2.1.0/debug',

'ojL10n': 'libs/oj/v2.1.0/ojL10n',

'ojtranslations': 'libs/oj/v2.1.0/resources',

'text': 'libs/require/text',

'signals': 'libs/js-signals/signals',

'mcs': 'libs/mcs-sdk/mcs'

}

// 以下省略

注意: 'signals'のエントリの後にコンマが必要です。

## モバイル・バックエンドにアクセスするモジュールの作成

このハンズオンでは、MCS JavaScript SDKを使用してMCSのモバイル・バックエンドにアクセスする処理をRequireJSのモジュールとして作成します。モジュールは、新規のJavaScriptソース・ファイルとしてC:\HOL\JETProject\myapp\src\js\mbeModule.jsとして保存します。

### NetBeans 8.1を使用している場合のモジュールの作成

NetBeans 8.1を使用している場合、mbeModule.jsは次の手順で作成できます。

1. 「プロジェクト」ウィンドウで「oraclejet-app」→「サイト・ルート」→「js」を右クリックし、コンテキスト・メニューから「新規」→「JavaScriptソース」を選択します。

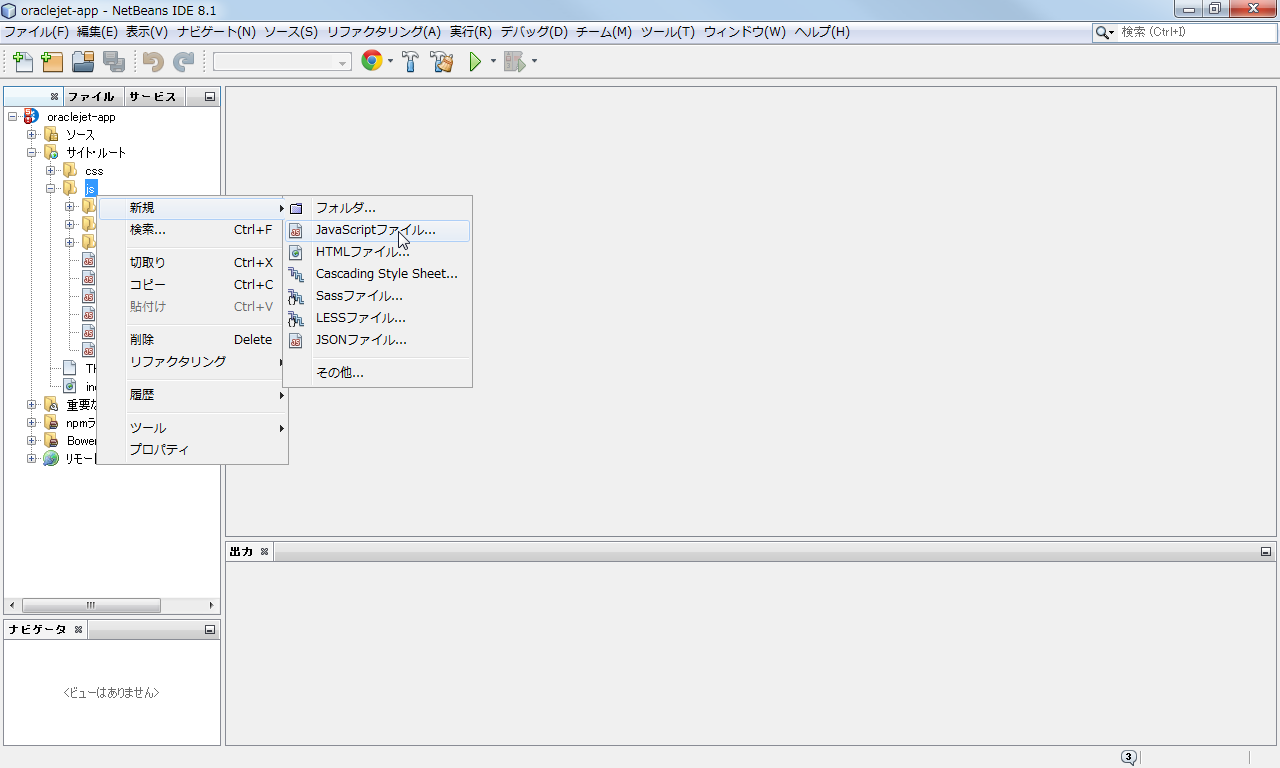


図 ‑4: 「新規」→「JavaScriptファイル」を選択

1. 「New JavaScriptファイル」ウィンドウの「名前と場所」ページが表示されます。「ファイル名」として「mbeModule」と指定し、「終了」ボタンをクリックします。

### mbeModule.jsのソースの編集

RequireJSのモジュールは、関数defineを使用して定義します。１つ目の引数として、このモジュールが依存する他のライブラリを配列で指定します。この時、main.jsのrequres.configのpathsでマッピングした名前が使用できるので、今回は'mcs'を指定しています。２つ目の引数は、モジュールの本体となる実装です。

コード ‑2: mbeModule.js

define(['mcs'], function () {

var mcs\_config = {

'logLevel': mcs.logLevelInfo,

'mobileBackends': {

'*<YOUR\_MBE\_NAME>*': {

'default': true,

'baseUrl': '*<YOUR\_BASE\_URL>*',

'authorization': {

'basicAuth': {

'backendId': '*<YOUR\_MBE\_ID>*',

'anonymousToken': '*<YOUR\_MBE\_ANONYMOUS\_TOKEN>*'

},

'oAuth': {

'clientId': '*<YOUR\_MBE\_CLIENT\_ID>*',

'clientSecret': '*<YOU\_MBE\_ClIENT\_SECRET>*',

'tokenEndpoint': '*<YOUR\_TOKEN\_ENDPOINT>*'

}

}

}

}

};

function MobileBackend() {

var self = this;

self.mobileBackend;

function init() {

mcs.MobileBackendManager.setConfig(mcs\_config);

self.mobileBackend =

mcs.MobileBackendManager.getMobileBackend('*<YOUR\_MBE\_NAME>*');

self.mobileBackend.setAuthenticationType('basicAuth');

}

init();

}

return new MobileBackend();

});

mcs\_configオブジェクトは、モバイル・アプリがアクセスするMCSのモバイル・バックエンドの接続情報を記述します。今回は以下の値を記述ください

表 ‑2

|  |  |
| --- | --- |
| 使用している環境の値と置き換える部分 | 置き換える値の確認（「設定」ページでの表示項目） |
| <YOUR\_MBE\_NAME> | CCDWorkshop |
| <YOUR\_BASE\_URL> | https://mcs-jporacletrial70175.mobileenv.us2.oraclecloud.com:443 |
| <YOUR\_MBE\_ID> | da9590be-6540-4f91-9d8f-cd7d802f883b |
| <YOUR\_MBE\_ANONYMOUS\_TOKEN> | SlBPUkFDTEVUUklBTDcwMTc1X01DU19NT0JJTEVfQU5PTllNT1VTX0FQUElEOnFtamw1N3B5d3pfWGFj |
| <YOUR\_MBE\_CLIENT\_ID> | bae93b32-2b58-4ea4-809f-848027912040 |
| <YOUR\_MBE\_ClIENT\_SECRET> | XZFgxjJkBRnJegprcLj2 |
| <YOUR\_TOKEN\_ENDPOINT> | https://jporacletrial70175.identity.us.oraclecloud.com/oam/oauth2/tokens |

# ユーザー認証の実装

ここでは、『 Oracle JETを使用した画面の作成』（15ページ）で作成したユーザー認証フォームを使用してMCSに登録済みのユーザー名とパスワードで認証できることを確認します。また、ログインできた場合は、ユーザー認証フォームの代わりに「ログアウト」ボタンを表示させます。「ログアウト」ボタンをクリックすると、再びユーザー認証フォームが表示されます。

## ログイン/ログアウト処理の実装

ここでは、『 MCS JavaScript SDKの導入 － モバイル・バックエンドにアクセスするモジュールの作成』（23ページ）で作成したモジュールmbeModule.jsに、MCS JavaScript SDKが提供するAPIを使用してログインとログアウトの処理を実装します。

MCSのモバイル・バックエンドに対するログインは、MCS JavaScript SDKのAuthorizationクラスに実装されているauthenticateメソッド、ログアウトはlogoutメソッドを使用します。

コード ‑1: ログイン/ログアウト処理

// ログイン処理

self.login = function (username, password,  
 successCallback, successCallback) {

self.mobileBackend.Authorization.authenticate(username, password,  
 successCallback, successCallback);

};

// ログアウト処理

self.logout = function () {

self.mobileBackend.Authorization.logout();

};

備考: AuthorizationクラスのAPIドキュメントは次のページを参照してください  
  
[https://docs.oracle.com/cloud/latest/mobilecs\_gs/MCSSJ/  
Authorization.html](https://docs.oracle.com/cloud/latest/mobilecs_gs/MCSSJ/Authorization.html)

## ビュー・モデルに対するモジュールの組み込み

作成したモジュールをビュー・モデルから利用するように設定します。モバイル・バックエンドにアクセスするためのモジュールmbeModule.jsは、main.jsと同じフォルダに配置されているので、次の例のようにdefineの一つ目の引数の配列に'mbeModule'を追加します。また、コールバック関数の引数として、mbeを追加します。これにより、モジュールmbeModule.js内で生成されたMobileBackendオブジェクトのインスタンスにビュー・モデルからアクセスが可能になります。

コード ‑2

define(['ojs/ojcore', 'knockout', 'jquery', 'appController', 'mbeModule'

, 'ojs/ojinputtext', 'ojs/ojbutton'],

function(oj, ko, $, app, mbe) {

function DashboardViewModel() {

var self = this;

self.usernameValue = ko.observable();

self.passwordValue = ko.observable();

self.isLoggedIn = ko.observable(false);

self.login = function (data, event) {

// confirm(self.usernameValue() + 'さん、こんにちは');

mbe.login(self.usernameValue(), self.passwordValue(),

self.loginSuccess, self.loginFailure);

};

self.loginSuccess = function (statusCode, message) {

self.isLoggedIn(true);

};

self.loginFailure = function (statusCode) {

self.isLoggedIn(false);

alert('ログインできません');

};

self.logout = function (data, event) {

self.isLoggedIn(false);

mbe.logout();

};

// 以下省略

## ビューへのデータ・バインド

ここでは、ビューにログアウトのためのボタンを追加します。またビュー・モデルに追加されたプロパティisLoggedInを用いて表示が切り替わるように設定します。表示/非表示の切り替えには、Knockoutが提供するif/ifnotバインドを使用できます。

コード ‑3: src\js\view\dashboard.htmlの編集 （網掛けされた部分を追加）

<!-- This is where your main page content will be loaded -->

<div class="oj-applayout-content">

<div role="main" class="oj-hybrid-applayout-content">

<div class="oj-hybrid-padding">

<div data-bind="ifnot: isLoggedIn">

<div class="oj-form-layout">

<div class="oj-form">

<div class="oj-flex">

<div class="oj-flex-item">

<label for="username">ユーザー名</label>

<input id="username"

data-bind="ojComponent: {

component: 'ojInputText',

value: usernameValue

}"/>

</div>

</div>

<div class="oj-flex">

<div class="oj-flex-item">

<label for="password">パスワード</label>

<input id="password"

data-bind="ojComponent: {

component: 'ojInputPassword',

value: passwordValue

}"/>

</div>

</div>

</div>

</div>

<div>

<button data-bind="click: login,

ojComponent: {

component: 'ojButton',

chroming: 'full' }">ログイン</button>

</div>

</div>

<div data-bind="if: isLoggedIn">

<div>

<button data-bind="click: logout,

ojComponent: {

component: 'ojButton',

chroming: 'full'

}">ログアウト</button>

</div>

</div>

</div>

</div>

</div>

## 稼働確認

**C:\HOL\JETProject\myapp\src\index.html**にアクセスして確認します。以下の情報にてログインを試して下さい

* ユーザー名: hcr
* パスワード: /Mcsdemo-1

# カスタムAPIの呼び出しとチャートの表示

ここでは、MCS上に事前定義されたカスタムAPI FixItFastのエンドポイントGET /stats/incidentsを、MCS JavaScript SDKを使用して呼び出し、結果をOracle JETのUIコンポーネントChart （ojChart）を使用してバー・チャートで表示します。

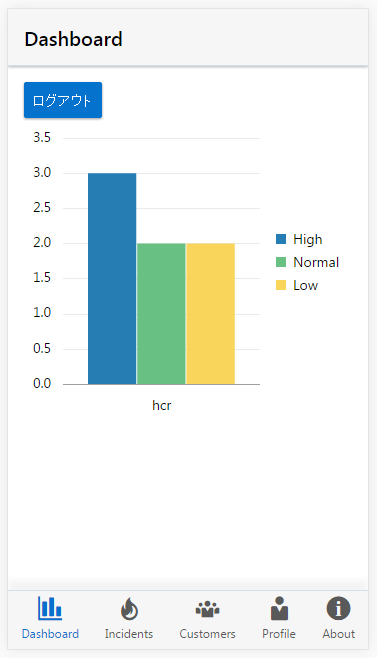


図 ‑1: 作成する画面

今回使用するカスタムAPI FixItFastのエンドポイントGET /stats/incidentsのレスポンスは次のようなJSONドキュメントが返ってきます。

コード ‑1: カスタムAPIのレスポンス

{

"incidentCount": {

"high": 3,

"normal": 2,

"low": 2

}

}

## カスタムAPI呼び出しの実装

mbeModule.jsにカスタムAPI呼び出しの処理を追加します。MCS JavaScript SDKのCustomCodeクラスのinvokeCustomCodeJSONRequestメソッドを使用します。

C:\HOL\JETProject\myapp\src\js\mbeModule.jsで定義されているMobileBackendオブジェクトのコンストラクタの中にコード 7‑2のメソッドを追加します。

コード ‑2: mbeModule.jsのMobileBackendオブジェクトのコンストラクタ内に追加

self.invokeGetIncidentsStats = function (technician,

successCallback, errorCallback) {

self.mobileBackend.CustomCode.invokeCustomCodeJSONRequest(

'ifixitfast/stats/incidents?technician=' + technician,

'GET', null, successCallback, errorCallback);

};

## Chartコンポーネントのモジュールのロードとデータ変換処理の実装

ここではC:\HOL\JETProject\myapp\src\js\viewModels\dashboard.jsを編集します。ビューにバー・チャートを追加するため、ojs/ojchartをロードするように構成します。

コード ‑3: 依存するモジュールとしてojs/ojchartを追加（網掛けされた部分を追加）

define(['ojs/ojcore', 'knockout', 'jquery', 'appController', 'mbe', 'ojs/ojinputtext', 'ojs/ojbutton', 'ojs/ojchart'],

次に、DashBoardViewModelオブジェクトのコンストラクタ内にチャートを表示するためのプロパティおよび処理を追加します。

コード ‑4: dashboard.jsを編集（網掛けされた部分を追加）

self.incidentsStatsSeries = ko.observableArray();

self.incidentsStatsGroups = ko.observableArray();

self.loginSuccess = function (statusCode, message) {

mobileBackend.invokeGetIncidentsStats(

'hcr',

function (statusCode, data) {

console.log(data.incidentCount);

var array = [

{ name: 'High', items: [ data.incidentCount.high, ] },

{ name: 'Normal', items: [ data.incidentCount.normal, ] },

{ name: 'Low', items: [ data.incidentCount.low,] },

];

self.incidentsStatsSeries(array);

self.incidentsStatsgroups(['hcr', ]);

},

function (statusCode) {

console.log('HTTP Status Code: ' + statusCode);

window.alert('データを取得できませんでした。');

}

);

self.isLoggedIn(true);

};

ここでは、ko.observableArrayが新たに使用されています。ko.observableArrayは、ko.observableと同様にKnockoutによって常に監視されていますが、ko.observableが１つのオブジェクトに対する変更を監視されているのに対して、ko.observableArrayは配列に対する要素の追加、削除などが監視されます。

## ビューへのバー・チャートの追加

ここではC:\HOL\JETProject\myapp\src\js\views\dashboard.htmlを編集します。ビュー・モデルのプロパティisLoggedInがtrueの時に表示されるブロック（data-bind="if: isLoggedIn"が指定されたdivタグ）の中にバー・チャートを追加します。

コード ‑5: バー・チャートの追加 （網掛けされた部分を追加）

<div data-bind="if: isLoggedIn">

<div>

<button data-bind="click: logout,

ojComponent: {

component: 'ojButton',

chroming: 'full'

}">ログアウト</button>

</div>

<div data-bind="ojComponent: {

component: 'ojChart',

type: 'bar',

orientation: 'vertical',

series: incidentsStatsSeries,

groups: incidentsStatsGroups,

animationOnDisplay: 'auto',

animationOnDataChange: 'auto'

}", style="width: 100%;">

</div>

</div>

## 稼働確認

**C:\HOL\JETProject\myapp\src\index.html**にアクセスして確認します。

# オラクルGoの実装

WIP

**C:\WINNT\Profiles\amisu.000\ﾃﾞｽｸﾄｯﾌﾟ\oracle.bmp**

**日本オラクル株式会社**

Copyright © Oracle Corporation Japan. All Rights Reserved.

無断転載を禁ず

この文書はあくまでも参考資料であり、掲載されている情報は予告なしに変更されることがあります。日本オラクル社は本書の内容に関していかなる保証もいたしません。また、本書の内容に関連したいかなる損害についても責任を負いかねます。

Oracleは米国Oracle Corporationの登録商標です。文中に参照されている各製品名及びサービス名は米国Oracle Corporationの商標または登録商標です。その他の製品名及びサービス名はそれぞれの所有者の商標または登録商標の可能性があります。