VMware vCloud® Architecture Toolkit™
for Service Providers

Extending VMware vCloud Director® User Interface Using Portal Extensibility – Ticketing Example

Version 2.9
April 2018

Kelby Valenti

VMware, Inc.
3401 Hillview Ave
Palo Alto, CA 94304
www.vmware.com

CLOUD PROVIDER PROGRAM

Contents

List of Figures

Sample Code

Introduction

VMware vCloud Director® version 9.0 enables you to create custom content directly in the vCloud Director HTML5 user interface (UI). This function opens many potential use cases for Service Providers.

This whitepaper walks you through the creation of a sample homegrown ticketing system that integrates directly into the vCloud Director HTML5 UI. You build up the ticketing system in seven phases. Each phase adds new functions that demonstrate various extension building blocks. The ticketing system enables tenants to view, create, edit, and delete tickets via a custom API from directly inside the vCloud Director HTML5 UI.

VMware recommends that you view the original Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper before you walk through these examples. That whitepaper includes a simple static About Page example and a section describing how to publish extensions that this paper does not include.

All sample code associated with these examples is available for reference at https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing.

Phase One – Getting Started

In this phase, you create a new project and an initial “Hello World” example. You learn how to define an extension and develop the code that displays the new extension in the vCloud Director HTML5 UI.

This example is available in GitHub at https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing/ticketing_phase1.

Note Most of the content created for this phase is very similar to the content created in the example static About Page shown in the Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper. The content is duplicated in this whitepaper so that you have the framework in place for the other phases.

2.1 Retrieving Plug-in Seed

1. Download or clone the GitHub repo at https://github.com/vmware/vcd-ext-sdk/.
   The /ui/vcd-plugin-seed directory contains the plug-in seed.
2. Duplicate the /ui/vcd-plugin-seed directory and rename it to match your project name.

For example, duplicate the /ui/vcd-plugin-seed directory to a new directory labeled “ticketing”. You complete all development for this extension in this new directory.

```
git clone https://github.com/vmware/vcd-ext-sdk.git
cp -r ./vcd-ext-sdk/ui/vcd-plugin-seed ticketing
rm -rf ./vcd-ext-sdk
cd ticketing
```

2.2 Deleting Example Stub Files

The seed project includes an example that you do not need. You can delete the following files from the "ticketing" directory:

- ./src/main/*
- ./src/public/assets/*
- ./src/public/i18n.json

```
rm -rf ./src/main/*
rm -rf ./src/public/assets/*
rm -f  ./src/public/i18n.json
```

2.3 Creating the Ticketing Package

You can define the packages required for your extension, including the name, description and version.

Warning Do not change any pre-existing package versions in the package.json file, because vCloud Director ignores all changes to package versions. However, you can add new packages here.

Edit the package.json file with extension information.

Sample Code 1: Phase One – ./package.json

```
{
  "name": "vcd-ui-ticketing-extension",
  "version": "0.0.1",
  "description": "VCD UI Ticketing Extension",
  "scripts": {
    "clean": "rimraf dist",
    "lint-fix": "tslint --fix \"src/**/*.ts\" || exit 0",
    "build": "rollup -c && ncp src/public dist && cd dist && bestzip plugin.zip bundle.js
manifest.json i18n.json assets/ && cd ..",
  },
  "license": "BSD-3-Clause",
  "dependencies": {
    "@angular/animations": "4.4.4",
13      "@angular/common": "4.4.4",
14      "@angular/compiler": "4.4.4",
15      "@angular/core": "4.4.4",
16      "@angular/forms": "4.4.4",
17      "@angular/http": "4.4.4",
18      "@angular/platform-browser": "4.4.4",
19      "@angular/platform-browser-dynamic": "4.4.4",
20      "@angular/router": "4.4.4",
21      "@ngrx/core": "1.2.0",
22      "@ngrx/effects": "2.0.3",
23      "@ngrx/store": "2.2.2",
24      "@webcomponents/custom-elements": "1.0.0",
25      "clarity-angular": "0.10.15",
26      "clarity-icons": "0.10.15",
27      "clarity-ui": "0.10.15",
28      "core-js": "2.5.1",
29      "mutationobserver-shim": "0.3.2",
30      "reflect-metadata": "0.1.10",
31      "reselect": "3.0.0",
32      "rxjs": "5.5.2",
33      "web-animations-js": "2.3.1",
34      "zone.js": "0.8.18",
35      "es6-shim": "0.35.3",
36
37      "@vcd-ui/common": "file:vcd-ui/common"
38    },
39    "devDependencies": {
40      "@types/jasmine": "2.5.47",
41      "@types/node": "7.0.12",
42      "bestzip": "1.1.4",
43      "body-parser": "1.17.1",
44      "express": "4.15.2",
45      "express-history-api-fallback": "2.1.0",
46      "http-proxy": "1.16.2",
47      "ncp": "2.0.0",
48      "node-sass": "4.5.2",
49      "npm": "3.10.10",
50      "open": "0.0.5",
51      "rimraf": "2.6.1",
52      "rollup": "0.41.6",
53      "rollup-plugin-angular": "0.4.4",
54      "rollup-plugin-typescript": "0.8.1",
55      "ts-helpers": "1.1.2",
56      "ts-node": "3.0.2",
57      "tslint": "4.5.1",
58      "typescript": "2.2.2",
59      "typings": "2.1.0"
60    }
61  }
```

- **Line 2**: Enters the name for the extension.
- **Line 3**: Enters the extension version.
- **Line 4**: Enters the description for the extension.

2.4 Creating the Ticketing Manifest

Configure the extension metadata in the manifest.json file.

Sample Code 2: Phase One – ./src/public/manifest.json

```
{
    "urn": "vmware:vcloud:ticketing",
    "name": "Ticketing",
    "containerVersion": "9.1.0",
    "version": "0.0.1",
    "scope": ["tenant"],
    "permissions": [],
    "description": "Manage ticketing functions",
    "vendor": "VMware",
    "license": "SPDX-License-Identifier: BSD-2-Clause; Copyright (C) VMware 2017.  All rights reserved.",
    "link": "http://www.vmware.com/support",
    "module": "TicketingModule",
    "route": "ticketing"
}
```

- **urn**: Unique Uniform Resource Names (URN) style ID for the extension
- **name**: Human readable name for the extension
- containerVersion: Minimum version number of the vCloud Director UI host required for this extension
- **version**: Extension version
- **scope**: Determines where to place the extension. Valid values include "tenant" and "service-provider".
- **permissions**: Minimum role based access control (RBAC) permissions that are required for the host to load the extension. Brackets ([]) indicate that no permissions are required. Currently, only brackets ([]) are allowed.
- **description**: Human readable description for the extension
- **vendor**: Human readable name of the vendor of the extension
- **license**: License for usage of the extension
- **link**: Support link for the extension
- **module**: Name of the exported Angular module that the application loads dynamically
- **route**: Top level URL route for the module

2.5 Creating Ticketing Internationalization

Tenants can view the vCloud Director UI in multiple languages. The i18n.json file stores the locale specific translations for the extension. Create an i18n.json file with the ticketing translations.

Sample Code 3: Phase One – ./src/public/i18n.json

```
1   {
2       "en": {
3           "nav.ticketing": "Ticketing",
4           "nav.ticketing.description": "Ticketing module",
5           "tickcomp.ticketing": "Ticketing"
6       }
7   }
```

- **Line 2**: The locale. If English is the current language, the subentries under a specific locale are used. You can add other locales as needed.
- **Lines 3-5**: Each key is translated into the specific translation value for the current locale.

Note If there are any missing translations, vCloud Director displays the "en" translation value. If the "en" translation key is also missing, vCloud Director displays the translation key prefixed with a question mark.

2.6 Creating the Ticketing HTML

You can create the "Hello World" screen by using the ticketing.component.html file.

Sample Code 4: Phase One – ./src/main/ticketing.component.html

```
1   <div class="content-area">
2     <h2 class="dashboard-header">{{'tickcomp.ticketing'|translate}}</h2>
3     Hello World  -  Look, it works!!!
4   </div>
```

- **Line 1**: Division with the class value set to "content-area".
- **Lines 2-3**: Defines the texts and their format that you want to see on the Ticketing screen.
- **Line 2**: The | translate element takes a specified variable name and displays the locale specific text based on the current locale.

2.7 Creating the Ticketing Component

Components define an application by using a simple configuration structure. Create a new ticketing.component.ts file that defines the TicketingComponent.

Sample Code 5: Phase One – ./src/main/ticketing.component.ts

```
1   import {Component} from '@angular/core';
2
3   @Component({
4     selector: 'ticketing-component',
5     templateUrl: './ticketing.component.html',
6     host: {'class': 'content-container'}
7   })
8
9   export class TicketingComponent {}
```

- **Line 4**: The selector for the component must be unique.
- **Line 5**: Uses the HTML in the external ticketing.component.html file.
- **Line 6**: Specifies the host class.
- **Line 8**: Exports TicketingComponent so that other modules can use it.

2.8 Creating Ticketing Routes

Routes enable navigation between various views when tenants perform various tasks. Create a new ticketing.routes.ts file that defines the necessary routes.

Sample Code 6: Phase One – ./src/main/ticketing.routes.ts

```
1   import {Route} from '@angular/router';
2   import {TicketingComponent} from './ticketing.component';
3
4   export const ROUTES: Route[] = [
5       {
6           path: '',
7           component: TicketingComponent
8       }
9   ];
```

- **Line 2**: Imports the TicketingComponent.
- **Lines 5-8**: By default, sends all traffic to the TicketingComponent.

2.9 Creating the Ticketing Module

NgModules organizes applications into cohesive function blocks. Create a new ticketing.module.ts file that defines the TicketingModule.

Sample Code 7: Phase One – ./src/main/ticketing.module.ts

```
1   import {CommonModule} from '@angular/common';
2   import {Inject, NgModule} from '@angular/core';
3   import {RouterModule} from '@angular/router';
4   import {Store} from '@ngrx/store';
5   import {EXTENSION ROUTE, ExtensionNavRegistration, ExtensionNavRegistrationAction, I18nModule}
    from '@vcd-ui/common';
6   import {TicketingComponent} from './ticketing.component';
7   import {ROUTES} from './ticketing.routes';
8
9   @NgModule({
10      imports: [
11          CommonModule,
12          I18nModule,
13          RouterModule.forChild(ROUTES)
14      ],
15      declarations: [
16          TicketingComponent
17      ],
18      bootstrap: [
19          TicketingComponent
20      ]
21  })
22  export class TicketingModule {
```

```
23      constructor(private appStore: Store<any>, @Inject(EXTENSION ROUTE) extensionRoute: string)
    {
24          const registration: ExtensionNavRegistration = {
25              path: extensionRoute,
26              nameCode: 'nav.ticketing',
27              descriptionCode: 'nav.ticketing.description'
28          };
29          appStore.dispatch(new ExtensionNavRegistrationAction(registration));
30      }
31  }
```

- **Lines 15-17**: Adds TicketingComponent to the declarations.
- **Lines 18-20**: Adds TicketingComponent to bootstrap.
- **Lines 23-31**: Registers the extension.
- **Line 26:** Uses “nav.ticketing” entry in i18n. Used to specify the label in the main navigation.
- **Line 27:** Uses “nav.ticketing.description” entry in i18n. Displays the description when you hover over the label in the main navigation.

2.10 Creating the Ticketing Index

You can define the most top entry point for the tree of code. Create an index.ts file with the exported TicketingModule.

Sample Code 8: Phase One – ./src/main/index.ts

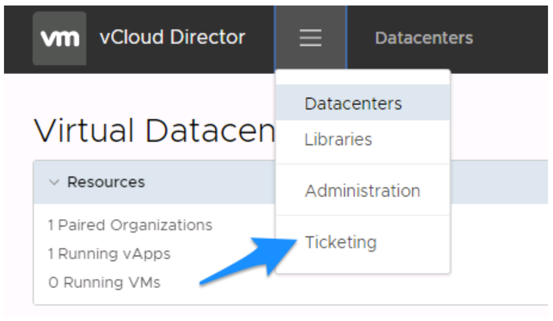
```
export {TicketingModule} from './ticketing.module';
```

2.11 Publishing and Viewing the Page

You have completed all the necessary code to view the sample “Hello World” page. To view the page, follow the procedures in the Publishing Extension section of the Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper.

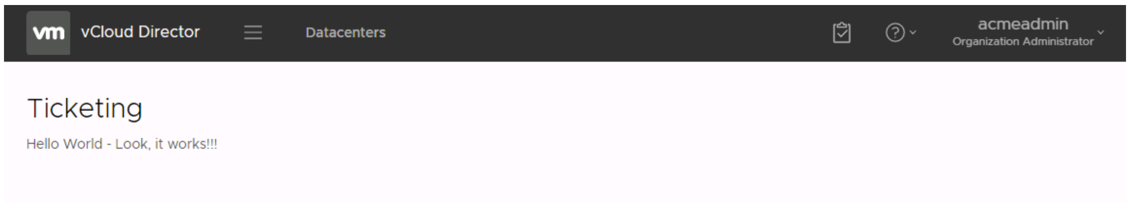
1. Log in to the tenant portal with a user name and password associated with an organization granted access.
   You can see a new navigational menu item labeled “Ticketing”.

   Figure 1: Phase One – Example Ticketing Menu Item



2. Click Ticking.
   The “Ticketing” screen appears.

   Figure 2: Phase One – Example Ticketing Screen



Ticketing

Hello World - Look, it works!!!

Phase Two – Displaying the Ticket List

In this phase, when the application loads the ticketing page, you can view a list of the tickets associated with the current organization.

This example is available in GitHub at https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing/ticketing_phase2.

3.1 Creating the Ticket Object

You define a structure that represents an individual ticket. Create a new ticket.ts file with a Ticket object. The Ticket object includes three elements: an ID, a message, and a status.

Sample Code 9: Phase Two – ./src/main/ticket.ts

```
export class Ticket {
  ticket_id: string;
  ticket_msg: string;
  status: string;
}
```

3.2 Creating Mock Tickets

To display and manage the tickets, you need to define a collection of mock tickets. Create a new mock-tickets.ts file that contains the mock tickets.

Sample Code 10: Phase Two – ./src/main/mock-tickets.ts

```
1   import {Ticket} from './ticket'
2
3   export let TICKETS: Ticket[] = [
4     {
5       "ticket_id": "111",
6       "ticket_msg": "I am opening a ticket",
7       "status": "Open"
8     },
9     {
10      "ticket_id": "112",
11      "ticket_msg": "My server is slow!",
12      "status": "Closed"
13    },
14    {
15      "ticket_id": "113",
16      "ticket_msg": "Please open port 22 for vm labeled ABC",
17      "status": "Open"
18    }
19  ];
```

- **Line 1**: Imports a Ticket object.
- **Line 3**: Exports mock ticket data as TICKETS and declares it as the Ticket type.
- **Lines 4-18**: Defines sample tickets.

3.3 Adding a Ticket List to Internationalization

To include the translation for listing tickets, modify the i18n.json file.

Sample Code 11: Phase Two – ./src/public/i18n.json

```
{
    "en": {
        "nav.ticketing": "Ticketing",
        "nav.ticketing.description": "Ticketing module",
        "tickcomp.ticketing": "Ticketing",
        "tickcomp.ticket_id": "Ticket ID",
        "tickcomp.description": "Description",
        "tickcomp.status": "Status",
        "tickcomp.pagination.of": "of",
        "tickcomp.pagination.tickets": "tickets"
    }
}
```

3.4 Adding a Ticket List to HTML

To display the collection of mock tickets, use a Clarity DataGrid. To include the HTML elements for the datagrid, modify the ticketing.component.html file.

Sample Code 12: Phase Two – ./src/main/ticketing.component.html

```
1   <div class="content-area">
2     <h2class="dashboard-header">{{'tickcomp.ticketing'|translate}}</h2>
3     <div>
4       <clr-datagrid [(clrDgSingleSelected)]="selectedTicket">
5         <clr-dg-column [clrDgField]="'ticket_id'">{{'tickcomp.ticket_id'|translate}}</clr-dg-
    column>
6         <clr-dg-column [clrDgField]="'ticket_msg'">{{'tickcomp.description'|translate}}</clr-
    dg-column>
7         <clr-dg-column [clrDgField]="'status'">{{'tickcomp.status'|translate}}</clr-dg-column>
8
9         <clr-dg-row *clrDgItems="let ticket of tickets" [clrDgItem]="ticket">
10          <clr-dg-cell>{{ticket.ticket_id}}</clr-dg-cell>
11          <clr-dg-cell>{{ticket.ticket_msg}}</clr-dg-cell>
12          <clr-dg-cell>{{ticket.status}}</clr-dg-cell>
13        </clr-dg-row>
14        <clr-dg-footer>
15          {{pagination.firstItem + 1}} - {{pagination.lastItem + 1}}
    {{'tickcomp.pagination.of'|translate}} {{pagination.totalItems}}
    {{'tickcomp.pagination.tickets'|translate}}
16          <clr-dg-pagination #pagination [clrDgPageSize]="10"></clr-dg-pagination>
17        </clr-dg-footer>
18      </clr-datagrid>
19    </div>
20  </div>
```

- **Lines 4-18**: Uses a datagrid in which you can select a single row. The clr-datagrid creates the datagrid and clrDgSingleSelected puts the datagrid into the single selection mode. The selected row is stored in the selectedTicket variable.
- **Lines 5-7**: Defines the headers for the datagrid. [cldDgField] enables sorting and filtering.
- **Line 9**: Loops through all the tickets. [clrDgItem] defines the ticket variable name as "ticket" when looping through the list of tickets.
- **Lines 10-12**: Displays ticket information.
- **Lines 15-16**: Displays pagination information.

3.5 Adding a Ticket List to the Ticketing Component

To include the methods that are required to support viewing tickets at load time, modify the ticketing.component.ts file.

Sample Code 13: Phase Two – ./src/main/ticketing.component.ts

```
1   import {Component, OnInit} from '@angular/core';
2
3   import {Ticket} from './ticket';
4   import {TicketService} from './ticket.service';
5
6   @Component({
7       selector: 'ticketing-component',
8       templateUrl: './ticketing.component.html',
9       host: {'class': 'content-container'},
10      providers: [TicketService]
11  })
12
13  export class TicketingComponent implements OnInit {
14    tickets: Ticket[];
15    selectedTicket: Ticket;
16
17    constructor(private ticketService: TicketService) { }
18
19    getTickets(): void {
20      this.ticketService.getTickets().subscribe(tickets => this.tickets = tickets);
21    }
22
23    ngOnInit(): void {
24      this.getTickets();
25    }
26  }
```

- **Line 1**: Imports OnInit.
- **Lines 3-4**: Imports Ticket and TicketService.
- **Line 10**: Adds TicketService to the list of providers.
- **Lines 14-15**: Defines tickets and selectedTicket variables.
- **Line 17**: Adds TicketService via the constructor.
- **Lines 19-21**: Adds the getTickets method. The getTickets method calls the TicketService getTickets method and stores the returned data in this.tickets.
- **Lines 23-25**: Loads the ticket information when the page is initialized.

3.6 Creating the Ticket Service

You can create a service to interact and manage the ticket data. In this phase, the application uses mock tickets that do not contain all API communication code. Create a ticket.service.ts file that defines the TicketService class.

Sample Code 14: Phase Two – ./src/main/ticket.service.ts

```
1   import {Injectable} from '@angular/core';
2   import {Observable} from 'rxjs';
3
4   import {Ticket} from './ticket';
5   import {TICKETS} from './mock-tickets';
6
7   @Injectable()
8   export class TicketService {
9     getTickets(): Observable<Ticket[]> {
10      return Observable.of<Ticket[]>(TICKETS);
11    }
12  }
```

- **Line 9**: getTickets returns an observable of type Ticket[].
- **Line 10**: Returns the entire TICKETS object.

3.7 Adding Clarity to TicketingModule

To use the ClarityModule, add it to the TicketingModule file. Modify the ticketing.module.ts file to include the ClarityModule.

Sample Code 15: Phase Two – ./src/main/ticketing.module.ts

```
1    import {CommonModule} from '@angular/common';
2    import {Inject, NgModule} from '@angular/core';
3    import {RouterModule} from '@angular/router';
4    import {Store} from '@ngrx/store';
5    import {EXTENSION_ROUTE, ExtensionNavRegistration, ExtensionNavRegistrationAction} from
     '@vcd-ui/common';
6    import {TicketingComponent} from './ticketing.component';
7    import {ROUTES} from './ticketing.routes';
8
9    import {ClarityModule} from 'clarity-angular';
10
11   @NgModule({
12       imports: [
13           ClarityModule,
14           CommonModule,
15           RouterModule.forChild(ROUTES)
16       ],
17       declarations: [
18           TicketingComponent
19       ],
20       bootstrap: [
21           TicketingComponent
22       ],
23       exports: [],
24       providers: []
25   })
26   export class TicketingModule {
27       constructor(private appStore: Store<any>, @Inject(EXTENSION_ROUTE) extensionRoute: string)
     {
28           const registration: ExtensionNavRegistration = {
29               path: extensionRoute,
30               nameCode: 'nav.ticketing',
31               descriptionCode: 'nav.ticketing.description'
32           };
33           appStore.dispatch(new ExtensionNavRegistrationAction(registration));
34       }
35   }
```

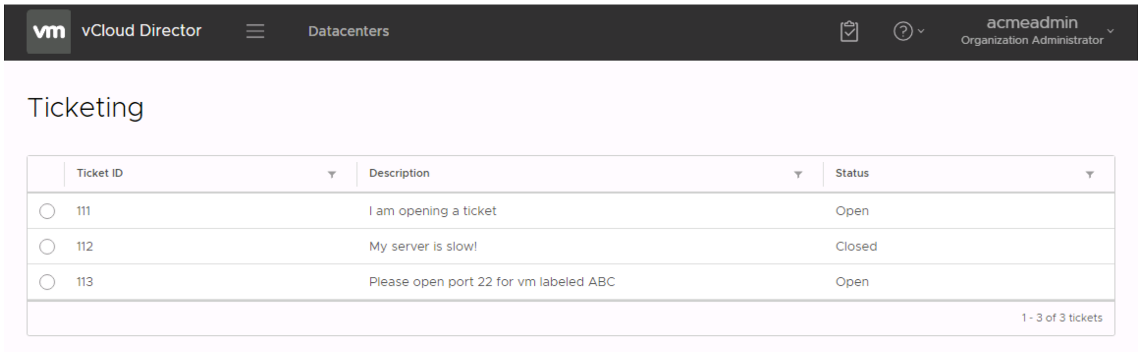
- **Line 9**: Imports ClarityModule.
- **Line 13**: Adds ClarityModule to the list of imports.

3.8 Publishing and Viewing the Page

To view the page, follow the procedures in the Publishing Extension section of the Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper.

1. Log in to the tenant portal with a user name and password associated with the organization granted access.
2. Go to the “Ticketing” page.
   You can see a list of all the mock tickets that the Clarity DataGrid shows.

Figure 3: Phase Two – Example Ticketing DataGrid



Ticketing

	Ticket ID	Description	Status
○	111	I am opening a ticket	Open
○	112	My server is slow!	Closed
○	113	Please open port 22 for vm labeled ABC	Open

1 - 3 of 3 tickets

Phase Three – Adding the Create Ticket Function

In this phase, you can add the function of creating new tickets. Tenants click a “Create Ticket” button to display a pop-up window in which tenants add a new ticket.

This example is available in GitHub at https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing/ticketing_phase3.

4.1 Adding Create Ticket to Internationalization

Translations are required to create tickets. To include the translations, modify the i18n.json file.

Sample Code 16: Phase Three – ./src/public/i18n.json

```
{
    "en": {
        "nav.ticketing": "Ticketing",
        "nav.ticketing.description": "Ticketing module",
        "tickcomp.ticketing": "Ticketing",
        "tickcomp.ticket_id": "Ticket ID",
        "tickcomp.description": "Description",
        "tickcomp.status": "Status",
        "tickcomp.pagination.of": "of",
        "tickcomp.pagination.tickets": "tickets",
        "tickcomp.btn.create_ticket": "Create Ticket",
        "tickcomp.modaladd.title": "Create Ticket",
        "tickcomp.modaladd.description": "Description",
        "tickcomp.btn.cancel": "Cancel",
        "tickcomp.btn.create": "Create"
    }
}
```

4.2 Adding Create Ticket to HTML

To enable tenants to create new tickets from the user interface, the following UI elements are required:

- A button
- A Clarity Modal: Displays the Create Ticket pop-up window.
- ngForm: Collects the user input.

To include the HTML elements, modify the ticketing.component.html file.

Sample Code 17: Phase Three – ./src/main/ticketing.component.html

```
<div class="content-area">
  <2 class="dashboard-header">{{'tickcomp.ticketing'|translate}}</h2>
  <div>
    <clr-datagrid [(clrDgSingleSelected)]="selectedTicket">
      <clr-dg-column [clrDgField]="'ticket_id'">{{'tickcomp.ticket_id'|translate}}</clr-dg-column>
      <clr-dg-column [clrDgField]="'ticket_msg'">{{'tickcomp.description'|translate}}</clr-dg-column>
      <clr-dg-column [clrDgField]="'status'">{{'tickcomp.status'|translate}}</clr-dg-column>

      <clr-dg-row *clrDgItems="let ticket of tickets" [clrDgItem]="ticket">
        <clr-dg-cell>{{ticket.ticket_id}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.ticket_msg}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.status}}</clr-dg-cell>
      </clr-dg-row>
      <clr-dg-footer>
        {{pagination.firstItem + 1}} - {{pagination.lastItem + 1}} {{'tickcomp.pagination.of'|translate}} {{pagination.totalItems}} {{'tickcomp.pagination.tickets'|translate}}
        <clr-dg-pagination #pagination [clrDgPageSize]="10"></clr-dg-pagination>
      </clr-dg-footer>
    </clr-datagrid>
    <button class="btn btn-sm btn-secondary" (click)="ticketaddmodal = true">{{'tickcomp.btn.create_ticket'|translate}}</button>

    <form #ticketaddform="ngForm" (ngSubmit)="createTicketSubmit(ticketaddform)" [hidden]="submitted" novalidate>
      <clr-modal [(clrModalOpen)]="ticketaddmodal" [clrModalSize]="'xl'">
        <h3 class="modal-title">{{'tickcomp.modaladd.title'|translate}}</h3>
```

```
24          <div class="modal-body">
25            <section class="form-block">
26              <div class="form-group">
27                <label for="ticketadddesc"
    class="required">{{'tickcomp.modaladd.description'|translate}}</label>
28                <textarea id="ticketadddesc" name="ticketadddesc" ngModel></textarea>
29              </div>
30            </section>
31          </div>
32          <div class="modal-footer">
33            <button type="button" class="btn btn-outline"
    (click)="createTicketCancel(ticketaddform)">{{'tickcomp.btn.cancel'|translate}}</button>
34            <button type="submit" class="btn btn-
    primary">{{'tickcomp.btn.create'|translate}}</button>
35          </div>
36        </clr-modal>
37      </form>
38    </div>
39  </div>
```

- **Line 19**: Adds a “Create Ticket” button. When the user clicks the button, the application sets the ticketaddmodal boolean that displays the modal dialog box.
- **Lines 21-37**: Adds a form to collect the description for the new ticket.
- **Lines 21,34**: When the “Create” button is clicked, the application calls the createTicketSubmit method of the component.
- **Lines 22-36**: Uses a Clarity Modal to display the form. Sets modal size to xl. Displays the modal when the ticketaddmodal value equals to true.
- **Line 33**: When the “Cancel” button is clicked, the application calls the cancelTicketSubmit method of the component.

4.3 Creating Ticketing Component CSS

You can define cascading style sheets (CSS) in an external file, which is similar to how you do with HTML. Create a new ticketing.component.css file that contains the formatting styles to display forms properly.

Sample Code 18: Phase Three – ./src/main/ticketing.component.css

```
form {
  display:inline;
  margin:0px;
  padding:0px;
}
```

4.4 Adding Create Ticket to the Ticketing Component

Component methods support creating tickets. To include the component methods, modify the ticketing.component.ts file.

Sample Code 19: Phase Three – ./src/main/ticketing.component.ts

```
import {Component, OnInit} from @angular/core';
import {NgForm} from '@angular/forms';

import {Ticket} from './ticket';
import {TicketService} from './ticket.service';

@Component({
  selector: 'ticketing-component',
  templateUrl: './ticketing.component.html',
  styleUrls: ['./ticketing.component.css'],
  host: {'class': 'content-container'},
  providers: [TicketService]
})

export class TicketingComponent implements OnInit {
  tickets: Ticket[];
  selectedTicket: Ticket;
  ticketaddmodal: boolean = false;

  constructor(private ticketService: TicketService) { }

  getTickets(): void {
    this.ticketService.getTickets().subscribe(tickets => this.tickets = tickets);
24    }
25
26    createTicketSubmit(form: NgForm) {
27      this.ticketService.createTicket(form.controls['ticketadddesc'].value)
28        .subscribe(newticket => {
29        this.tickets.push(newticket);
30        this.ticketaddmodal = false;
31        form.reset();
32      });
33    }
34
35    createTicketCancel(form: NgForm) {
36      this.ticketaddmodal = false;
37      form.reset();
38    }
39
40    ngOnInit(): void {
41      this.getTickets();
42    }
43  }
```

- **Line 10**: Adds an external CSS file.
- **Line 18**: Defines boolean to determine when to show the Create Ticket modal.
- **Lines 26-33**: Methods that handle the Create Ticket submission.
- **Line 27**: Sends the new ticket description to ticketSevice.createTicket.
- **Lines 28-32**: When the observable returns, add the ticket to the tickets array, reset the form and close the pop-up window.
- **Lines 35-38**: Methods that handle closing the Create Ticket pop-up window.

4.5 Adding the Create Ticket Service to Ticket Service

The createTicket method enables tenants to create new tickets. To include the createTicket method, modify the ticket.service.ts file.

Sample Code 20: Phase Three – ./src/main/ticket.service.ts

```
1   import {Injectable} from '@angular/core';
2   import {Observable} from 'rxjs';
3
4   import {Ticket} from './ticket';
5   import {TICKETS} from './mock-tickets';
6
7   @Injectable()
8   export class TicketService {
9     ticketID = 200;
10
11    getTickets(): Observable<Ticket[]> {
12      return Observable.of<Ticket[]>(TICKETS);
13    }
14
15    createTicket(description: string): Observable<Ticket[]> {
16      this.ticketID += 1
17      return Observable.of<Ticket[]>({
18        "ticket id": this.ticketID,
19        "ticket msg": description,
20        "status": "Open"
21      });
22    }
23  }
```

- **Line 9**: Defines unique identifier for a ticket
- **Lines 15-22**: Adds a new method for creating tickets. The new method returns an observable.
- **Lines 17-21**: Returns the new ticket as an observable.

4.6 Adding FormsModule to Ticketing Module

To collect ticket descriptions, you need to add the FormsModule to the TicketingModule. To add the FormsModule, modify the ticketing.module.ts file.

Sample Code 21: Phase Three – ./src/main/ticketing.module.ts

```
1   import {CommonModule} from '@angula/common';
2   import {FormsModule}  from '@angular/forms';
3   import {Inject, NgModule} from '@angular/core';
4   import {RouterModule} from '@angular/router';
5   import {Store} from '@ngrx/store';
6   import {EXTENSION ROUTE, ExtensionNavRegistration, ExtensionNavRegistrationAction, I18nModule}
    from '@vcd-ui/common';
7   import {TicketingComponent} from './ticketing.component';
8   import {ROUTES} from './ticketing.routes';
9
10  import {ClarityModule} from 'clarity-angular';
11
12  @NgModule({
13      imports: [
14          ClarityModule,
15          CommonModule,
16          I18nModule,
17          RouterModule.forChild(ROUTES),
18          FormsModule
19      ],
20      declarations: [
21          TicketingComponent
22      ],
23      entryComponents: [
24          TicketingComponent
25      ],
26      exports: [],
27      providers: []
28  })
29  export class TicketingModule {
30      constructor(private appStore: Store<any>, @Inject(EXTENSION ROUTE) extensionRoute: string)
    {
31          const registration: ExtensionNavRegistration = {
32              path: extensionRoute,
33              nameCode: 'nav.ticketing',
34              descriptionCode: 'nav.ticketing.description'
35          };
36          appStore.dispatch(new ExtensionNavRegistrationAction(registration));
37      }
38  }
```

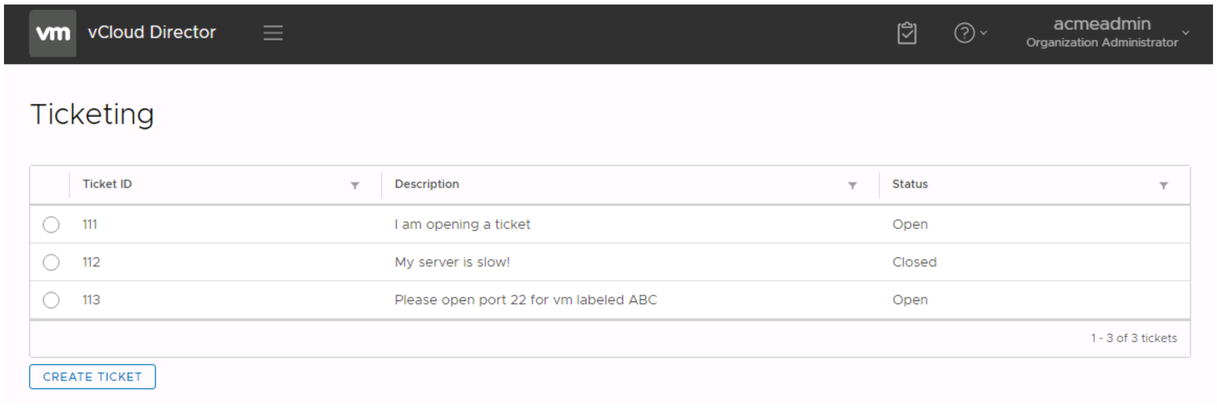
- **Line 2**: Imports FormsModule.
- **Line 18**: Adds FormsModule to the list of imports.

4.7 Publishing and Viewing the Page

To view the page, follow the procedures defined in the Publishing Extension section of the Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper.

1. Log in to the tenant portal with a user name and password associated with an organization granted access.
2. Go to the “Ticketing” page.
3. Click the “Create Ticket” button.

Figure 4: Phase Three – Example Create Ticket Button



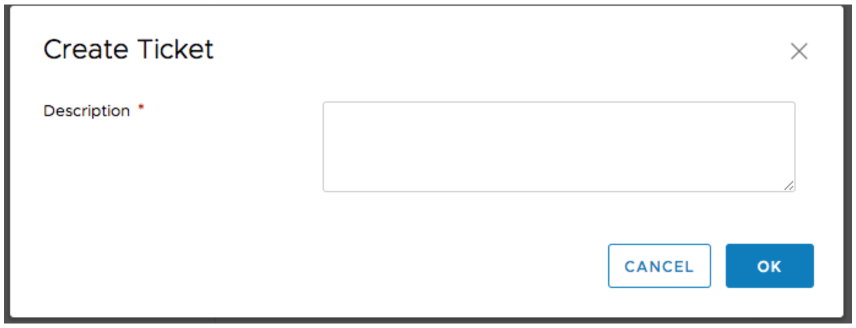
Ticketing

Ticket ID	Description	Status
111	I am opening a ticket	Open
112	My server is slow!	Closed
113	Please open port 22 for vm labeled ABC	Open

1 - 3 of 3 tickets

The "Create Ticket" modal appears.

Figure 5: Phase Three – Example Create Ticket Pop-up Window



Create Ticket

Description *

Phase Four – Adding the Ticket Details Function

In this phase, you can add the function of viewing and editing tickets. Tenants can click a “Ticket Details” button to display a pop-up window in which tenants view and edit ticket details.

This example is available in GitHub at https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing/ticketing_phase4.

5.1 Adding Ticket Details to Internationalization

Translations are required to display ticket details. To include the translations, modify the i18n.json file.

Sample Code 22: Phase Four – ./src/public/i18n.json

```
{
    "en": {
        "nav.ticketing": "Tickting",
        "nav.ticketing.description": "Ticketing module",
        "tickcomp.ticketing": "Ticketing",
        "tickcomp.ticket id": "Ticket ID",
        "tickcomp.description": "Description",
        "tickcomp.status": "Status",
        "tickcomp.pagination.of": "of",
        "tickcomp.pagination.tickets": "tickets",
        "tickcomp.btn.create ticket": "Create Ticket",
        "tickcomp.modaladd.title": "Create Ticket",
        "tickcomp.modaladd.description": "Description",
        "tickcomp.btn.cancel": "Cancel",
        "tickcomp.btn.create": "Create",
        "tickcomp.btn.ticket details": "Ticket Details",
        "tickcomp.modaldetails.title": "Ticket Details",
        "tickcomp.modaldetails.id": "ID",
        "tickcomp.modaldetails.status": "Status",
        "tickcomp.modaldetails.description": "Description",
        "tickcomp.btn.update": "Update"
    }
}
```

5.2 Adding Ticket Details to HTML

To enable tenants to view and edit ticket details from the user interface, the following UI elements are required:

- A button
- A Clarity Modal: Displays ticket details in a pop-up window.
- ngForm: Collects user input.

To include the HTML elements, modify the ticketing.component.html file.

Sample Code 23: Phase Four – ./src/main/ticketing.component.html

```
<div class="content-area">
  <h class="dashboard-header">{{'tickcomp.ticketing'|translate}}</h2>
  <div>
    <clr-datagrid [(clrDgSingleSelected)]="selectedTicket">
      <clr-dg-column [clrDgField]="'ticket id'">{{'tickcomp.ticket id'|translate}}</clr-dg-column>
      <clr-dg-column [clrDgField]="'ticket msg'">{{'tickcomp.description'|translate}}</clr-dg-column>
      <clr-dg-column [clrDgField]="'status'">{{'tickcomp.status'|translate}}</clr-dg-column>

      <clr-dg-row *clrDgItems="let ticket of tickets" [clrDgItem]="ticket">
        <clr-dg-cell>{{ticket.ticket id}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.ticket msg}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.status}}</clr-dg-cell>
      </clr-dg-row>
      <clr-dg-footer>
        {{pagination.firstItem + 1}} - {{pagination.lastItem + 1}} {{'tickcomp.pagination.of'|translate}} {{pagination.totalItems}} {{'tickcomp.pagination.tickets'|translate}}
        <clr-dg-pagination #pagination [clrDgPageSize]="10"></clr-dg-pagination>
      </clr-dg-footer>
    </clr-datagrid>
    <button class="btn btn-sm btn-secondary" (click)="ticketaddmodal = true">{{'tickcomp.btn.create_ticket'|translate}}</button>
```

```
20      <button class="btn btn-sm btn-secondary" (click)="ticketdetailsmodal = true"
    [disabled]="!selectedTicket">{{'tickcomp.btn.ticket details'|translate}}</button>
21
22      <form #ticketaddform="ngForm" (ngSubmit)="createTicketSubmit(ticketaddform)"
    [hidden]="submitted" novalidate>
23        <clr-modal [(clrModalOpen)]="ticketaddmodal" [clrModalSize]="'xl'">
24          <h3 class="modal-title">{{'tickcomp.modaladd.title'|translate}}</h3>
25          <div class="modal-body">
26            <section class="form-block">
27              <div class="form-group">
28                <label for="ticketadddesc"
    class="required">{{'tickcomp.modaladd.description'|translate}}</label>
29                <textarea id="ticketadddesc" name="ticketadddesc" ngModel></textarea>
30              </div>
31            </section>
32          </div>
33          <div class="modal-footer">
34            <button type="button" class="btn btn-outline"
    (click)="createTicketCancel(ticketaddform)">{{'tickcomp.btn.cancel'|translate}}</button>
35            <button type="submit" class="btn btn-
    primary">{{'tickcomp.btn.create'|translate}}</button>
36          </div>
37        </clr-modal>
38      </form>
39
40      <form #ticketdetailsform="ngForm" (ngSubmit)="ticketDetailsSubmit(ticketdetailsform)"
    [hidden]="submitted" novalidate *ngIf="selectedTicket">
41        <clr-modal [(clrModalOpen)]="ticketdetailsmodal" [clrModalSize]="'xl'">
42          <h3 class="modal-title">{{'tickcomp.modaldetails.title'|translate}}</h3>
43          <div class="modal-body">
44            <section class="form-block">
45              <div class="form-group">
46                <label for="ticketdetailsid">{{'tickcomp.modaldetails.id'|translate}}</label>
47                <input id="ticketdetailsid" name="ticketdetailsid"
    [ngModel]="selectedTicket.ticket id" [disabled]="true">
48              </div>
49              <div class="form-group">
50                <label
    for="ticketdetailsstatus">{{'tickcomp.modaldetails.status'|translate}}</label>
51                <input id="ticketdetailsstatus" name="ticketdetailsstatus"
    [ngModel]="selectedTicket.status" [disabled]="true">
52              </div>
53              <div class="form-group">
54                <label for="ticketdetailsdesc"
    class="required">{{'tickcomp.modaldetails.description'|translate}}</label>
55                <textarea id="ticketdetailsdesc" name="ticketdetailsdesc"
    [ngModel]="selectedTicket.ticket msg"></textarea>
56              </div>
57            </section>
58          </div>
59          <div class="modal-footer">
60            <button type="button" class="btn btn-outline"
    (click)="ticketDetailsCancel(ticketdetailsform)">{{'tickcomp.btn.cancel'|translate}}</button>
61            <button type="submit" class="btn btn-
    primary">{{'tickcomp.btn.update'|translate}}</button>
62          </div>
63        </clr-modal>
64      </form>
65    </div>
66  </div>
67
```

- **Line 20**: Adds a “Ticket Details” button. When the user clicks the button, the application sets the ticketdetailsmodal boolean that displays the modal dialog box.
- **Lines 40-64**: Adds a form in which tenants can view and edit ticket details.
- **Lines 40,61**: When the “Update” button is clicked, the application calls the ticketDetailsSubmit method of the component.
- **Line 41**: Uses a Clarity Modal to display the form. Sets modal size to xl. Displays the modal when the ticketdetailsmodal value equals to true.
- **Lines 47,51**: Disables the fields. Uses [ngModal] to populate the field values in advance.
- **Line 55**: Uses [ngModel] to populate the ticket description in advance.
- **Line 60**: When the “Cancel” button is clicked, the application calls the ticketDetailsCancel method of the component.

5.3 Adding Ticket Details to the Ticketing Component

Component methods enable tenants to view and edit ticket details. To include the component methods, modify the ticketing.component.ts file.

Sample Code 24: Phase Four – ./src/main/ticketing.component.ts

```
1   import {Component, OnInit} from 'angular/core';
2   import {NgForm} from '@angular/forms';
3
4   import {Ticket} from './ticket';
5   import {TicketService} from './ticket.service';
6
7   @Component({
8     selector: 'ticketing-component',
9     templateUrl: './ticketing.component.html',
10    styleUrls: ["./ticketing.component.css"],
11    host: {'class': 'content-container'},
12    providers: [TicketService]
13  })
14
15  export class TicketingComponent implements OnInit {
16    tickets: Ticket[];
17    selectedTicket: Ticket;
18    ticketaddmodal: boolean = false;
19    ticketdetailsmodal: boolean = false;
20
21    constructor(private ticketService: TicketService) { }
22
23    getTickets(): void {
24      this.ticketService.getTickets().subscribe(tickets => this.tickets = tickets);
25    }
26
27    createTicketSubmit(form: NgForm) {
28      this.ticketService.createTicket(form.controls['ticketadddesc'].value)
29        .subscribe(newticket => {
30        this.tickets.push(newticket);
31        this.ticketaddmodal = false;
32        form.reset();
33      });
34    }
35
36    createTicketCancel(form: NgForm) {
37      this.ticketaddmodal = false;
38      form.reset();
39    }
40
41    ticketDetailsSubmit(form: NgForm) {
42      this.selectedTicket.ticket msg = form.controls['ticketdetailsdesc'].value;
43      this.ticketdetailsmodal = false;
44      form.reset();
45    }
46
47    ticketDetailsCancel(form: NgForm) {
48      this.ticketdetailsmodal = false;
49    }
50
51    ngOnInit(): void {
52      this.getTickets();
53    }
54  }
```

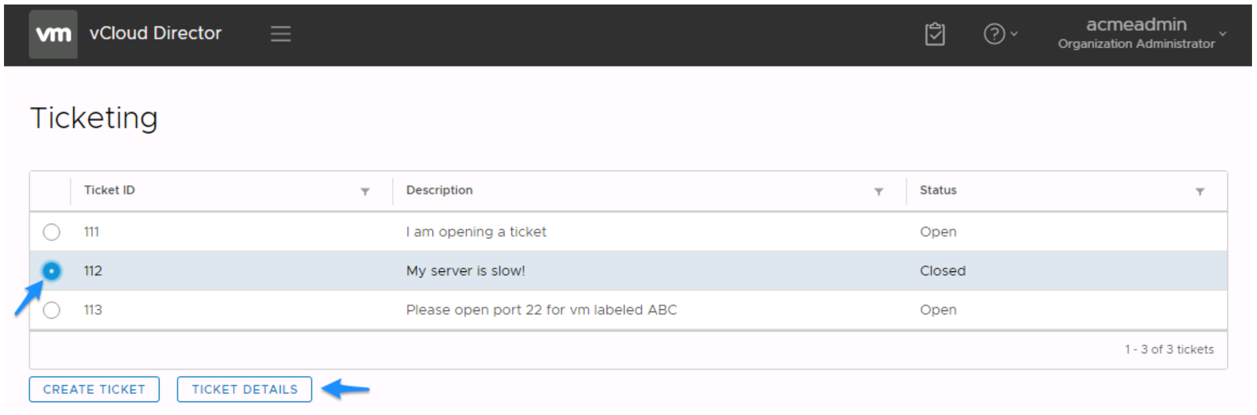
- **Line 19**: Defines boolean to determine when to show the Ticket Details modal.
- **Lines 41-45**: Methods that handle the Ticket Details submission.
- **Line 42**: Modifies the selectedTicket with the updated description.
- **Lines 43-44**: Resets the form and close the pop-up window.
- **Lines 47-49**: Methods that handle closing the Ticket Details pop-up window.

5.4 Publishing and Viewing the Page

To view the page, follow the procedures defined in the Publishing Extension section of the Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper.

1. Log in to the tenant portal with a user name and password associated with an organization granted access.
2. Go to the “Ticketing” page.
3. Select a ticket.
4. Click the “Ticket Details” button.

Figure 6: Phase Four – Example Ticket Details Selection



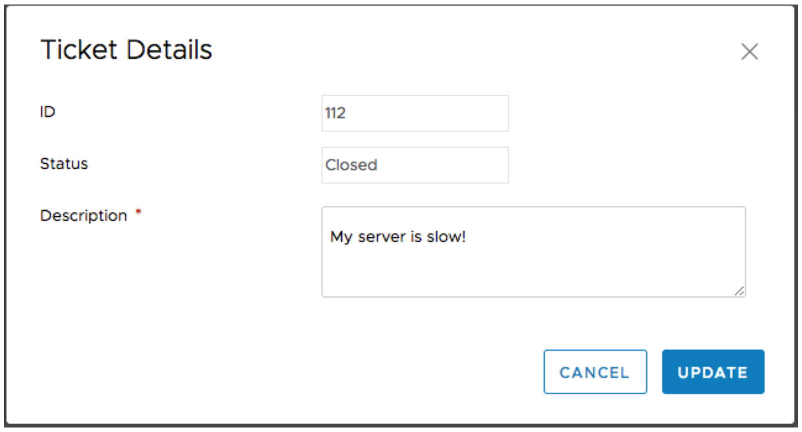
Ticketing

	Ticket ID	Description	Status
○	111	I am opening a ticket	Open
●	112	My server is slow!	Closed
○	113	Please open port 22 for vm labeled ABC	Open

1 - 3 of 3 tickets

The “Ticket Details” modal appears.

Figure 7: Phase Four – Example Ticket Details



Ticket Details

ID	112
Status	Closed
Description *	My server is slow!

Phase Five – Adding the Delete Ticket Function

In this phase, you can add the function of deleting existing tickets. Tenants can delete a selected ticket by clicking the “Delete Ticket” button.

This example is available in GitHub at https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing/ticketing_phase5.

6.1 Adding Delete Ticket to Internationalization

Translations are required to delete tickets. To include the translations, modify the i18n.json file.

Sample Code 25: Phase Five – ./src/public/i18n.json

```
{
    "en": {
        "nav.ticketing": "Tickting",
        "nav.ticketing.description": "Ticketing module",
        "tickcomp.ticketing": "Ticketing",
        "tickcomp.ticket id": "Ticket ID",
        "tickcomp.description": "Description",
        "tickcomp.status": "Status",
        "tickcomp.pagination.of": "of",
        "tickcomp.pagination.tickets": "tickets",
        "tickcomp.btn.create ticket": "Create Ticket",
        "tickcomp.modaladd.title": "Create Ticket",
        "tickcomp.modaladd.description": "Description",
        "tickcomp.btn.cancel": "Cancel",
        "tickcomp.btn.create": "Create",
        "tickcomp.btn.ticket details": "Ticket Details",
        "tickcomp.modaldetails.title": "Ticket Details",
        "tickcomp.modaldetails.id": "ID",
        "tickcomp.modaldetails.status": "Status",
        "tickcomp.modaldetails.description": "Description",
        "tickcomp.btn.update": "Update",
        "tickcomp.btn.delete ticket": "Delete Ticket"
    }
}
```

6.2 Adding Delete Ticket to HTML

To enable tenants to delete tickets from the user interface, a Delete Ticket button is required. To include the button on the user interface, modify the ticketing.component.html file.

Sample Code 26: Phase Five – ./src/main/ticketing.component.html

```
<div class="content-area">
  <h class="dashboard-header">{{'tickcomp.ticketing'|translate}}</h2>
  <div>
    <clr-datagrid [(clrDgSingleSelected)]="selectedTicket">
      <clr-dg-column [clrDgField]="'ticket id'">{{'tickcomp.ticket id'|translate}}</clr-dg-column>
      <clr-dg-column [clrDgField]="'ticket msg'">{{'tickcomp.description'|translate}}</clr-dg-column>
      <clr-dg-column [clrDgField]="'status'">{{'tickcomp.status'|translate}}</clr-dg-column>

      <clr-dg-row *clrDgItems="let ticket of tickets" [clrDgItem]="ticket">
        <clr-dg-cell>{{ticket.ticket id}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.ticket msg}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.status}}</clr-dg-cell>
      </clr-dg-row>
      <clr-dg-footer>
        {{pagination.firstItem + 1}} - {{pagination.lastItem + 1}} {{'tickcomp.pagination.of'|translate}} {{pagination.totalItems}} {{'tickcomp.pagination.tickets'|translate}}
        <clr-dg-pagination #pagination [clrDgPageSize]="10"></clr-dg-pagination>
      </clr-dg-footer>
    </clr-datagrid>
    <button class="btn btn-sm btn-secondary" (click)="ticketaddmodal = true">{{'tickcomp.btn.create ticket'|translate}}</button>
    <button class="btn btn-sm btn-secondary" (click)="ticketdetailsmodal = true" [disabled]="!selectedTicket">{{'tickcomp.btn.ticket details'|translate}}</button>
    <button class="btn btn-sm btn-secondary" (click)="deleteTicket()" [disabled]="!selectedTicket">{{'tickcomp.btn.delete ticket'|translate}}</button>

```

```
23      <form #ticketaddform="ngForm" (ngSubmit)="createTicketSubmit(ticketaddform)"
    [hidden]="submitted" novalidate>
24        <clr-modal [(clrModalOpen)]="ticketaddmodal" [clrModalSize]="'xl'">
25          <h3 class="modal-title">{{'tickcomp.modaladd.title'|translate}}</h3>
26          <div class="modal-body">
27            <section class="form-block">
28              <div class="form-group">
29                <label for="ticketadddesc"
    class="required">{{'tickcomp.modaladd.description'|translate}}</label>
30                <textarea id="ticketadddesc" name="ticketadddesc" ngModel></textarea>
31              </div>
32            </section>
33          </div>
34          <div class="modal-footer">
35            <button type="button" class="btn btn-outline"
    (click)="createTicketCancel(ticketaddform)">{{'tickcomp.btn.cancel'|translate}}</button>
36            <button type="submit" class="btn btn-
    primary">{{'tickcomp.btn.create'|translate}}</button>
37          </div>
38        </clr-modal>
39      </form>
40
41      <form #ticketdetailsform="ngForm" (ngSubmit)="ticketDetailsSubmit(ticketdetailsform)"
    [hidden]="submitted" novalidate *ngIf="selectedTicket">
42        <clr-modal [(clrModalOpen)]="ticketdetailsmodal" [clrModalSize]="'xl'">
43          <h3 class="modal-title">{{'tickcomp.modaldetails.title'|translate}}</h3>
44          <div class="modal-body">
45            <section class="form-block">
46              <div class="form-group">
47                <label for="ticketdetailsid">{{'tickcomp.modaldetails.id'|translate}}</label>
48                <input id="ticketdetailsid" name="ticketdetailsid"
    [ngModel]="selectedTicket.ticket id" [disabled]="true">
49              </div>
50              <div class="form-group">
51                <label
    for="ticketdetailsstatus">{{'tickcomp.modaldetails.status'|translate}}</label>
52                <input id="ticketdetailsstatus" name="ticketdetailsstatus"
    [ngModel]="selectedTicket.status" [disabled]="true">
53              </div>
54              <div class="form-group">
55                <label for="ticketdetailsdesc"
    class="required">{{'tickcomp.modaldetails.description'|translate}}</label>
56                <textarea id="ticketdetailsdesc" name="ticketdetailsdesc"
    [ngModel]="selectedTicket.ticket msg"></textarea>
57              </div>
58            </section>
59          </div>
60          <div class="modal-footer">
61            <button type="button" class="btn btn-outline"
    (click)="ticketDetailsCancel(ticketdetailsform)">{{'tickcomp.btn.cancel'|translate}}</button>
62            <button type="submit" class="btn btn-
    primary">{{'tickcomp.btn.update'|translate}}</button>
63          </div>
64        </clr-modal>
65      </form>
66    </div>
67  </div>
```

- **Line 21**: Adds a “Delete Ticket” button to trigger the delete action. When the user clicks the button, the application calls the DeleteTicket method of the component.

6.3 Adding Delete Ticket to the Ticketing Component

To enable tenants to delete tickets, a Delete Ticket method is required in the component. To include the Delete Ticket method, modify the ticketing.component.ts file.

Sample Code 27: Phase Five – ./src/main/ticketing.component.ts

```
import {Component, OnInit} from 'angular/core';
import {NgForm} from '@angular/forms';

import {Ticket} from './ticket';
import {TicketService} from './ticket.service';

@Component({
  selector: 'ticketing-component',
  templateUrl: './ticketing.component.html',
  styleUrls: ["./ticketing.component.css"],
11    host: {'class': 'content-container'},
12    providers: [TicketService]
13  })
14
15  export class TicketingComponent implements OnInit {
16    tickets: Ticket[];
17    selectedTicket: Ticket;
18    ticketaddmodal: boolean = false;
19    ticketdetailsmodal: boolean = false;
20
21    constructor(private ticketService: TicketService) { }
22
23    getTickets(): void {
24      this.ticketService.getTickets().subscribe(tickets => this.tickets = tickets);
25    }
26
27    createTicketSubmit(form: NgForm) {
28      this.ticketService.createTicket(form.controls['ticketadddesc'].value).subscribe(newticket
    => {
29        this.tickets.push(newticket);
30        this.ticketaddmodal = false;
31        form.reset();
32      });
33    }
34
35    createTicketCancel(form: NgForm) {
36      this.ticketaddmodal = false;
37      form.reset();
38    }
39
40    ticketDetailsSubmit(form: NgForm) {
41      this.selectedTicket.ticket msg = form.controls['ticketdetailsdesc'].value;
42      this.ticketdetailsmodal = false;
43      form.reset();
44    }
45
46    ticketDetailsCancel(form: NgForm) {
47      this.ticketdetailsmodal = false;
48    }
49
50    deleteTicket() {
51      this.ticketService.deleteTicket(this.selectedTicket.ticket id).subscribe(tickets =>
    this.tickets = tickets);
52      this.selectedTicket = null;
53    }
54
55    ngOnInit(): void {
56      this.getTickets();
57    }
58  }
```

- **Lines 50-53**: Methods that handle deleting tickets.
- **Line 51**: Calls the ticketService.deleteTicket method. When the observable returns, modify this.tickets with the returned tickets.

6.4 Adding the Delete Ticket Service to Ticket Service

To enable tenants to delete tickets, a Delete Ticket method is required in the TicketService. To include the deleteTicket method, modify the ticket.service.ts file.

Sample Code 28: Phase Five – ./src/main/ticket.service.ts

```
import {Injectable} from '@angular/cor';
import {Observable} from 'rxjs';

import {Ticket} from './ticket';
import {TICKETS} from './mock-tickets';

@Injectable()
export class TicketService {
  ticketID = 200;

  getTickets(): Observable<Ticket[]> {
    return Observable.of<Ticket[]>(TICKETS);
  }

  createTicket(description: string): Observable<Ticket[]> {
```

```
16      this.ticketID += 1
17      return Observable.of<Ticket[]>({
18        "ticket id": this.ticketID,
19        "ticket msg": description,
20        "status": "Open"
21      });
22    }
23
24    deleteTicket(ticket id): Observable<Ticket[]> {
25      for (let i=0; i<TICKETS.length; i++) {
26        if (TICKETS[i].ticket id == ticket id)
27        {
28          TICKETS.splice(i, 1);
29          break;
30        }
31      }
32      return Observable.of<Ticket[]>(TICKETS);
33    }
34  }
```

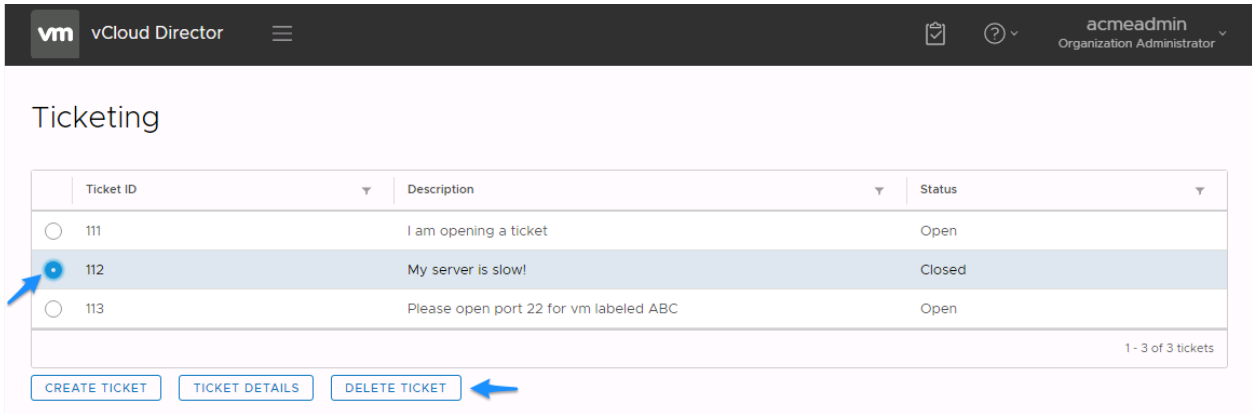
- **Lines 24-33**: Adds a new method for deleting tickets. The new method returns an observable.
- **Lines 25-31**: Check tickets and remove the specified ticket from the array.
- **Line 32**: Returns an observable that contains all the tickets.

6.5 Publishing and Viewing the Page

To view the page, follow the procedures defined in the Publishing Extension section of the Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper.

1. Log in to the tenant portal with a user name and password associated with an organization granted access.
2. Go to the "Ticketing" page.
3. Select a ticket.
4. Click the "Delete Ticket" button.

Figure 8: Phase Five – Example Delete Ticket

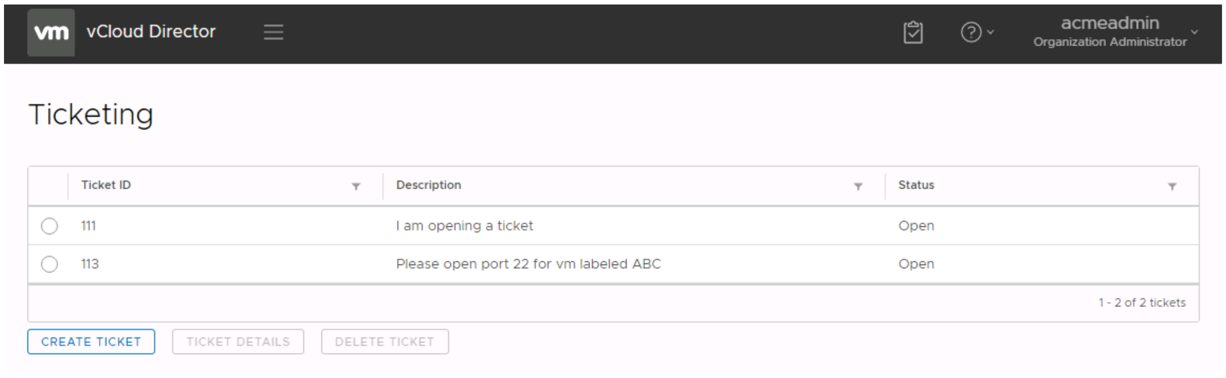


	Ticket ID	Description	Status
○	111	I am opening a ticket	Open
●	112	My server is slow!	Closed
○	113	Please open port 22 for vm labeled ABC	Open

1 - 3 of 3 tickets

The ticket is removed from the list.

Figure 9: Phase Five – Example Delete Ticket Result



Ticketing

	Ticket ID	Description	Status
◯	111	I am opening a ticket	Open
◯	113	Please open port 22 for vm labeled ABC	Open

1 - 2 of 2 tickets

Phase Six – Connecting to the API

In this phase, you can connect the ticketing system to a new vCloud Director API endpoint that is defined as a vCloud Director Extension Service. You can reuse the code developed in a previous whitepaper as the backend API. For more information about the whitepaper, see Extending VMware vCloud API with vCloud Extensibility Framework.

This example is available in GitHub at https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing/ticketing_phase6.

7.1 Preparing an API Extension

To configure the backend API, follow the instructions in the Extending VMware vCloud API with vCloud Extensibility Framework whitepaper. After you follow this whitepaper, you have an API server running. This API server provides a vCloud Director Extension Service that enable you to retrieve tickets, edit and delete tickets directly from the vCloud API.

7.2 Adding API Related Changes to the HTML

Given that the API response time is unpredictable, you can add a loading indicator on the UI. To avoid tenants clicking any button accidentally when a request is in progress, you can disable all the buttons. HTML elements are required to display the loading indicator and to disable buttons during the period waiting for an API response. To include the HTML elements, modify the ticketing.component.html file.

Sample Code 29: Phase Six – ./src/main/ticketing.component.html

```
<div class="content-area">
  <h2class="dashboard-header">{{'tickcomp.ticketing'|translate}}</h2>
  <div>
    <clr-datagrid [(clrDgSingleSelected)]="selectedTicket"
[clrDgLoading]="ticketdatagridloading">
      <clr-dg-column [clrDgField]="'ticket id'">{{'tickcomp.ticket id'|translate}}</clr-dg-
column>
      <clr-dg-column [clrDgField]="'ticket msg'">{{'tickcomp.description'|translate}}</clr-
dg-column>
      <clr-dg-column [clrDgField]="'status'">{{'tickcomp.status'|translate}}</clr-dg-column>

      <clr-dg-row *clrDgItems="let ticket of tickets" [clrDgItem]="ticket">
        <clr-dg-cell>{{ticket.ticket id}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.ticket msg}}</clr-dg-cell>
        <clr-dg-cell>{{ticket.status}}</clr-dg-cell>
      </clr-dg-row>
      <clr-dg-footer>
        {{pagination.firstItem + 1}} - {{pagination.lastItem + 1}}
{{'tickcomp.pagination.of'|translate}} {{pagination.totalItems}}
{{'tickcomp.pagination.tickets'|translate}}
        <clr-dg-pagination #pagination [clrDgPageSize]="10"></clr-dg-pagination>
      </clr-dg-footer>
    </clr-datagrid>

    <button class="btn btn-sm btn-secondary" (click)="ticketaddmodal = true"
[disabled]="ticketdatagridloading">{{'tickcomp.btn.create ticket'|translate}}</button>
    <button class="btn btn-sm btn-secondary" (click)="ticketdetailsmodal = true"
[disabled]="ticketdatagridloading
|| !selectedTicket">{{'tickcomp.btn.ticket details'|translate}}</button>
    <button class="btn btn-sm btn-secondary" (click)="deleteTicket()"
[disabled]="ticketdatagridloading
|| !selectedTicket">{{'tickcomp.btn.delete ticket'|translate}}</button>

    <form #ticketaddform="ngForm" (ngSubmit)="createTicketSubmit(ticketaddform)"
[hidden]="submitted" novalidate>
      <clr-modal [(clrModalOpen)]="ticketaddmodal" [clrModalSize]="'xl'">
        <h3 class="modal-title">{{'tickcomp.modaladd.title'|translate}}</h3>
        <div class="modal-body">
          <section class="form-block">
            <div class="form-group">
              <label for="ticketadddesc"
class="required">{{'tickcomp.modaladd.description'|translate}}</label>
              <textarea id="ticketadddesc" name="ticketadddesc" ngModel></textarea>
            </div>
          </section>
        </div>
```

```
35          <div class="modal-footer">
36            <button type="button" class="btn btn-outline"
    (click)="createTicketCancel(ticketaddform)">{{'tickcomp.btn.cancel'|translate}}</button>
37            <button type="submit" class="btn btn-
    primary">{{'tickcomp.btn.create'|translate}}</button>
38          </div>
39        </clr-modal>
40      </form>
41
42      <form #ticketdetailsform="ngForm" (ngSubmit)="ticketDetailsSubmit(ticketdetailsform)"
    [hidden]="submitted" novalidate *ngIf="selectedTicket">
43        <clr-modal [(clrModalOpen)]="ticketdetailsmodal" [clrModalSize]="'xl'">
44          <h3 class="modal-title">{{'tickcomp.modaldetails.title'|translate}}</h3>
45          <div class="modal-body">
46            <section class="form-block">
47              <div class="form-group">
48                <label for="ticketdetailsid">{{'tickcomp.modaldetails.id'|translate}}</label>
49                <input id="ticketdetailsid" name="ticketdetailsid"
    [ngModel]="selectedTicket.ticket id" [disabled]="true">
50              </div>
51              <div class="form-group">
52                <label
    for="ticketdetailsstatus">{{'tickcomp.modaldetails.status'|translate}}</label>
53                <input id="ticketdetailsstatus" name="ticketdetailsstatus"
    [ngModel]="selectedTicket.status" [disabled]="true">
54              </div>
55              <div class="form-group">
56                <label for="ticketdetailsdesc"
    class="required">{{'tickcomp.modaldetails.description'|translate}}</label>
57                <textarea id="ticketdetailsdesc" name="ticketdetailsdesc"
    [ngModel]="selectedTicket.ticket msg"></textarea>
58              </div>
59            </section>
60          </div>
61          <div class="modal-footer">
62            <button type="button" class="btn btn-outline"
    (click)="ticketDetailsCancel(ticketdetailsform)">{{'tickcomp.btn.cancel'|translate}}</button>
63            <button type="submit" class="btn btn-
    primary">{{'tickcomp.btn.update'|translate}}</button>
64          </div>
65        </clr-modal>
66      </form>
67    </div>
68  </div>
```

- **Line 4**: Shows a loading indicator when ticketdatgridloading is true.
- **Lines 20-22**: Disables the buttons when data is loading.

7.3 Adding API Related Changes to the Ticketing Component

To display the loading indicator and to disable buttons during the period waiting for an API response, modify the methods in the ticketing.component.ts file.

Sample Code 30: Phase Six – ./src/main/ticketing.component.ts

```
import {Component, OnInit} from '@ngular/core';
import {NgForm} from '@angular/forms';

import {Ticket} from './ticket';
import {TicketService} from './ticket.service';

@Component({
  selector: 'ticketing-component',
  templateUrl: './ticketing.component.html',
  styleUrls: ['./ticketing.component.css'],
  host: {'class': 'content-container'},
  providers: [TicketService]
})

export class TicketingComponent implements OnInit {
  tickets: Ticket[];
  selectedTicket: Ticket;
  ticketdatagridloading: boolean = false;
  ticketaddmodal: boolean = false;
  ticketdetailsmodal: boolean = false;

  constructor(private ticketService: TicketService) { }

24    getTickets(): void {
25      this.ticketdatagridloading = true;
26      this.ticketService.getTickets().subscribe(tickets => {
27        this.tickets = tickets;
28        this.ticketdatagridloading = false;
29      });
30    }
31
32    createTicketSubmit(form: NgForm) {
33      this.ticketdatagridloading = true;
34      this.ticketaddmodal = false;
35      this.ticketService.createTicket(form.controls['ticketadddesc'].value).subscribe(newticket
    => {
36        this.tickets.push(newticket);
37        form.reset();
38        this.ticketdatagridloading = false;
39      });
40    }
41
42    createTicketCancel(form: NgForm) {
43      this.ticketaddmodal = false;
44      form.reset();
45    }
46
47    ticketDetailsSubmit(form: NgForm) {
48      this.ticketdatagridloading = true;
49      this.ticketdetailsmodal = false;
50      this.ticketService.modifyTicket(this.selectedTicket.ticket_id,
    form.controls['ticketdetailsdesc'].value).subscribe(newticket => {
51        this.selectedTicket.ticket msg = form.controls['ticketdetailsdesc'].value;
52        form.reset();
53        this.ticketdatagridloading = false;
54      });
55    }
56
57    ticketDetailsCancel(form: NgForm) {
58      this.ticketdetailsmodal = false;
59    }
60
61    deleteTicket() {
62      this.ticketdatagridloading = true;
63      this.ticketService.deleteTicket(this.selectedTicket.ticket id).subscribe(tickets => {
64        this.tickets = tickets;
65        this.ticketdatagridloading = false;
66      });
67      this.selectedTicket = null;
68    }
69
70    ngOnInit(): void {
71      this.getTickets();
72    }
73  }
```

- **Line 18**: Defines the variables that determine when to show the loading indicator.
- **Lines 25-29, 33-38, 48-53, 62-67**: Shows or hides the loading indicator at proper times.

7.4 Adding API Related Changes to the Ticket Service

You can convert the TicketService from using mock tickets to using the live API ticket data. To interact with the API by using the HTTP module, you need to update each of the TicketService methods. You also create some helper functions to parse the XML data that the ticketing API extension returns. Modify the ticket.service.ts file as follows.

Sample Code 31: Phase Six – ./src/main/ticket.service.ts

```
1   import {Injectable, Inject}    from '@agular/core';
2   import {Headers, Http, Response} from '@angular/http';
3   import {Observable} from 'rxjs';
4
5   import {Ticket} from './ticket';
6
7   import {AuthTokenHolderService, API ROOT URL} from '@vcd-ui/common';
8
9   @Injectable()
10  export class TicketService {
11    private currentOrgId:String;
12    private currentUserId:String;
```

```
  private headers:{};

  constructor(
    private http:Http,
    authTokenHolderService: AuthTokenHolderService,
    @Inject(API_ROOT_URL) private apiRootUrl: string)
    {
      this.headers = {'headers' : {'x-vcloud-authorization': authTokenHolderService.token, 'Accept':'application/*+xml;version=29.0'}};
    }

  getOrgId(): Observable<String> {
    if (this.currentOrgId) {
      return Observable.of<String>(this.currentOrgId);
    }
    return this.http.get('/api/org', this.headers)
      .map((res:Response) => {
        let dom = parseXml(res.text());
        return dom.getElementsByTagName('Org')[0].getAttribute('href').split('/').slice(-1)[0];
      })
      .do(orgId => this.currentOrgId = orgId);
  }

  getTickets(): Observable<Ticket[]> {
    return this.getOrgId()
      .mergeMap(orgId => {
        return this.http
          .get(`${this.apiRootUrl}/api/org/${orgId}/ticketing`, this.headers)
          .map((res:Response) => parseTickets(res.text()));
      });
  }

  createTicket(description:string): Observable<Ticket>  {
    return this.http
      .post(`${this.apiRootUrl}/api/org/${this.currentOrgId}/ticketing`,
            `<?xml version="1.0" encoding="UTF-8"?><ticket><ticket_msg>${description}</ticket_msg></ticket>`,
            this.headers)
      .map(res => parseTickets(res.text())[0]);
  }

  modifyTicket(ticket_id:string, description:string): Observable<Response> {
    return this.http
      .put(`${this.apiRootUrl}/api/org/${this.currentOrgId}/ticketing/${ticket_id}`,
            `<?xml version="1.0" encoding="UTF-8"?><ticket><ticket_msg>${description}</ticket_msg></ticket>`,
            this.headers)
  }

  deleteTicket(ticket_id:string): Observable<Ticket[]> {
    return this.http
      .delete(`${this.apiRootUrl}/api/org/${this.currentOrgId}/ticketing/${ticket_id}`,
              this.headers)
      .map((res:Response) => parseTickets(res.text()));
  }
}

function parseXml(xml:string) {
   var dom = null;
   if (window.DOMParser) {
      try {
         dom = (new DOMParser()).parseFromString(xml, 'text/xml');
      }
      catch (e) { dom = null; }
   }
   else if (window.ActiveXObject) {
      try {
         dom = new ActiveXObject('Microsoft.XMLDOM');
         dom.async = false;
         if (!dom.loadXML(xml)) // parse error ..

            window.alert(dom.parseError.reason + dom.parseError.srcText);
      }
      catch (e) { dom = null; }
   }
   else
      alert('cannot parse xml string!');
   return dom;
}

```

```
90  function parseTickets(xml:string) {
91    let result:Ticket[] = [];
92    if (xml) {
93      let dom = parseXml(xml);
94      let tickets = dom.getElementsByTagName('ticket');
95      for (let i=0; i<tickets.length; i++) {
96        let ticket = tickets[i];
97        result.push(
98          {
99            ticket_id: getNodeValue(ticket, 'ticket_id'),
100           ticket_msg: getNodeValue(ticket, 'ticket_msg'),
101           status: getNodeValue(ticket, 'status')
102         }
103       )
104     }
105   }
106   return result;
107 }
108
109 function getNodeValue(dom:any, tagname:string) {
110   let ret = '';
111   if (dom) {
112     let elems = dom.getElementsByTagName(tagname);
113     if (elems && elems[0] && elems[0].firstChild) {
114       ret = elems[0].firstChild.nodeValue;
115     }
116   }
117   return ret;
118 }
```

- **Lines 1,2,7**: Import classes that you want to use.
- **Lines 11-13**: Defines class variables.
- **Lines 15-21**: Adds HTTP and AuthTokenHolderService. AuthTokenHolderService contains the current session token. Use AuthTokenHolderService to create the headers variable used for API calls.
- **Lines 23-34**: The getOrgId method, if necessary, calls the vCloud Director API and retrieves the current organization ID. The method returns the ID using an observable.
- **Lines 36-41**: This method retrieves the organization ID from getOrgId, and retrieves the tickets associated with the organization ID by calling the API. The method returns the tickets as an observable.
- **Lines 52-57**: Updates the ticket with the description that the API provides.
- **Lines 60-63**: Deletes the ticket via the API. Returns an observable that contains the full list of tickets.
- **Lines 67-88:** The parseXML function reads the xml that the API returns.
- **Lines 90-107**: Parses ticket xml and converts ticket xml to ticket structure.
- **Lines 109-118**: Helper function that parses xml and retrieves nodeValue.

7.5 Removing Mock Tickets

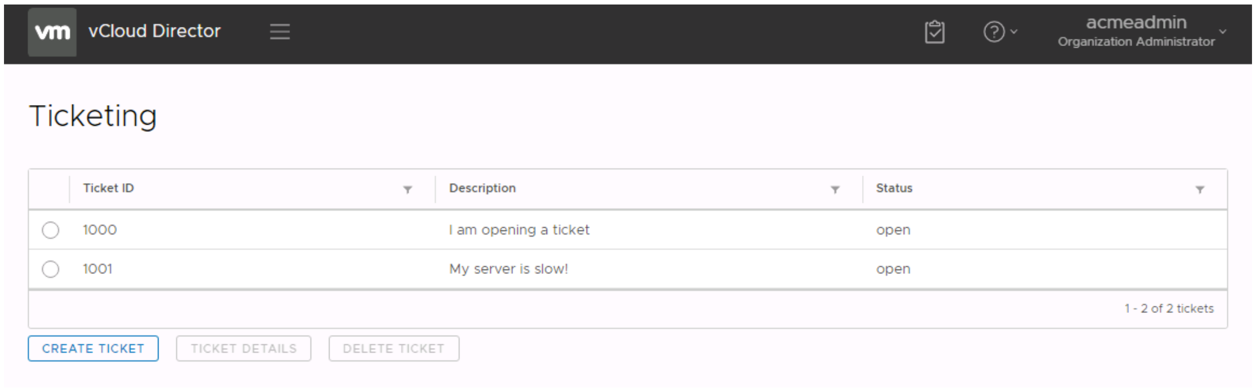
The mock-ticket.ts file is no longer needed and you can delete it.

7.6 Publishing and Viewing the Page

To view the page, follow the procedures defined in the Publishing Extension section of the Extending VMware vCloud Director User Interface Using Portal Extensibility whitepaper.

1. Log in to the tenant portal with a user name and password associated with an organization granted access.
2. Go to the “Ticketing” page and check all the functions.
   Data comes to and from the API.

Figure 10: Phase Six – Example Ticketing Screen



	Ticket ID	Description	Status
○	1000	I am opening a ticket	open
○	1001	My server is slow!	open

1 - 2 of 2 tickets

Conclusion

This white paper describes creating a functional ticketing example of a vCloud Director Portal Extension. It describes how to:

- Use the sample project to create new extensions.
- Organize and define extension code.
- Use Clarity components to display DataGrids and PopUps.
- Connect extensions to the vCloud API.

vCloud Director Portal Extensions enables you to create simple and complex pages directly inside the vCloud Director HTML5 portal.

References

This section provides additional information pertinent to this document and its topics.

Document title	Link or URL
VMware vCloud® Architecture Toolkit™ for Service Providers	https://www.vmware.com/cloud-computing/cloud-architecture/vcat-sp.html
VMware vCloud® Architecture Toolkit™ (vCAT) Blog	https://blogs.vmware.com/vcat/
VMware vCloud® API Programming Guide for Service Providers	https://vdc-download.vmware.com/vmwb-repository/dcr-public/1b6cf07d-adb3-4dba-8c47-9c1c92b04857/9f039b76-fcc5-456f-9e6a-2fc41a46dded/vcloud_sp_api_guide_30_0.pdf
Extending VMware vCloud® API with vCloud Extensibility Framework	https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/vcat/vmware-vcloud-api-extension-whitepaper.pdf
Extending VMware vCloud Director® User Interface Using Portal Extensibility	https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/vcat/vmware-extending-vcloud-director-user-interface-using-portal-extensibility.pdf
Portal Extensibility Ticketing Sample Code	https://github.com/vmware/vcd-ext-sdk/tree/master/ui/samples/ticketing
vCloud Director UI Extensibility Plug-In Seed	https://github.com/vmware/vcd-ext-sdk/tree/master/ui/vcd-plugin-seed
Clarity Design System	https://vmware.github.io/clarity/
Node.JS®	https://nodejs.org/