

Login with One Time Password(OTP)

Implementing the feature to provide multifactor authentication with one time password.

1. Introduction:

This feature enables users to authenticate themselves using a one-time password (OTP) sent to their mobile devices.

2. Feature Description:

The Mobile OTP Login feature provides an additional layer of security by requiring users to verify their identity using a unique OTP sent to their registered mobile numbers. This enhances the login process and helps prevent unauthorized access to user accounts.

3.Usage Guide:

To use the Mobile OTP Login feature, follow these steps:

1. Enter your registered email on the login screen.
2. Click on the "Get OTP" button to request an OTP.
3. Check your mobile device for the OTP message.
4. Enter the received OTP in the provided field on the login screen.
5. Click on the "Login" button to authenticate and log in to your account.

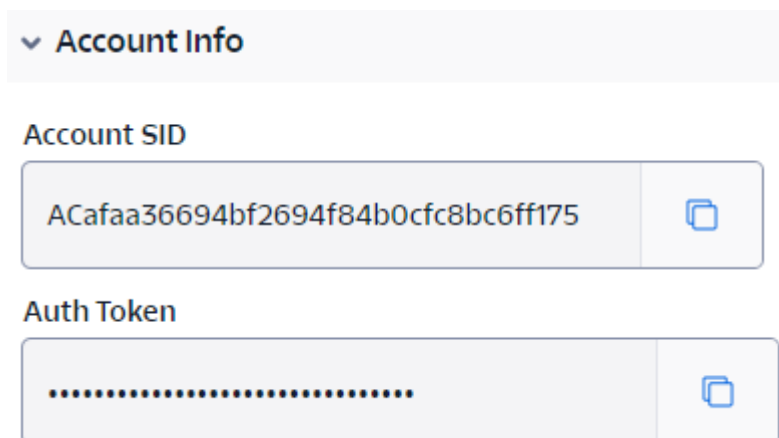
4.Implementation Details:

Generating and Sending OTPs:



- Here, by using the twillio services we are generating the OTPs and sending those to the registered users mobile numbers.
- For this requirement first we need to register with twillio and get the “twillio sid” and “Auth token”.
- The twillio platform will gives us the services for generating and validating the OTPs in our SFCC environment.
- By configuring the twillio services in our SFCC platform , we can able to provide the multi factor authentication to the customers. The below are steps to get the services for generating and validating the OTPs from twillio.

Step-1:

- Create an account in twillio and get the account “sid” and “Auth token” as shown in below.



The screenshot displays the Twilio 'Account Info' section. It features two input fields. The first field, labeled 'Account SID', contains the alphanumeric string 'ACafaa36694bf2694f84b0cfc8bc6ff175'. The second field, labeled 'Auth Token', is filled with a series of dots to represent a masked token. Both fields have a copy icon to their right.

▼ Account Info	
Account SID	
ACafaa36694bf2694f84b0cfc8bc6ff175	
Auth Token	
.....	

Step-2:

- Get the services for generating and validating the OTPs.

MacOS/Linux PC-curl Powershell Node.js Python

☐ Show auth token

```
curl -X POST "https://verify.twilio.com/v2/Services/VAdd71580796d
--data-urlencode "To="+919133650394" \
--data-urlencode "Channel=sms" \
-u "ACafaa36694bf2694f84b0cfc8bc6ff175:$TWILIO_AUTH_TOKEN"

echo
echo -n "Please enter the OTP:"
read OTP_CODE

curl -X POST "https://verify.twilio.com/v2/Services/VAdd71580796d
--data-urlencode "To="+919133650394" \
--data-urlencode "Code=$OTP_CODE" \
-u "ACafaa36694bf2694f84b0cfc8bc6ff175:$TWILIO_AUTH_TOKEN"
```

) Replace `$TWILIO_AUTH_TOKEN` with your Auth token in Account Info

- Select the programming language whatever you are convenient. The first URL mentioned here is for sending OTP to the mentioned mobile number followed by its “country code”.
- After requesting the first twillio service for generating the otp we will get the response as shown in below.

```
POST https://verify.twilio.com/v2/Services/VAdd71580796da551f467c55de45224994/Verifications

{
  "status": "pending",
  "payee": null,
  "date_updated": "2024-03-12T06:16:07Z",
  "send_code_attempts": [
    {
      "attempt_sid": "VLfee6e93e410fb1da400d43d5d596b6d4",
      "channel": "sms",
      "time": "2024-03-12T06:16:07.000Z"
    }
  ],
  "account_sid": "ACafaa36694bf2694f84b0cfc8bc6ff175",
  "to": "+1[REDACTED]",
  "amount": null,
  "valid": false,
  "lookup": {
    "carrier": null
  },
  "url": "https://verify.twilio.com/v2/Services/VAdd71580796da551f467c55de45224994/Verifications/"
}
```

- For verifying the OTP which were received to our mobile number we need to use the second service of the twilio.
- If the OTP was verified successfully then we will get the response as Approved, as shown in below.

```
POST https://verify.twilio.com/v2/Services/VAdd71580796da551f467c55de45224994/VerificationCheck

{
  "status": "approved",
  "payee": null,
  "date_updated": "2024-03-12T06:25:42Z",
  "account_sid": "ACafaa36694bf2694f84b0cfc8bc6ff175",
  "to": "+1[REDACTED]",
  "amount": null,
  "valid": true,
  "sid": "VE5a7d550c3a1e6b4b23b00531370e5ab4",
  "date_created": "2024-03-12T06:16:07Z",
  "service_sid": "VAdd71580796da551f467c55de45224994",
  "channel": "sms"
}
```

5.Code implementation:

- For implementing the multi factor authentication in our SFRA, we need to do the changes in Account.js controller and accountHelpers.js file and loginForm.isml files.

Changes made in Account.js controller:

- In Login route of the Account.js we need to verify the Otp, and if the status is “OK” then we need to redirect the customer to his account dashboard page. The changes are shown below.

```
Account.js
pp_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > controllers > JS Account.js > server.post('Login') call
93  server.post(
94      'Login',
95      server.middleware.https,
96      csrfProtection.validateAjaxRequest,
97      function (req, res, next) {
98          var CustomerMgr = require('dw/customer/CustomerMgr');
99          var Resource = require('dw/web/Resource');
100          var Site = require('dw/system/Site');
101
102          var accountHelpers = require('*/cartridge/scripts/helpers/accountHelpers');
103          var emailHelpers = require('*/cartridge/scripts/helpers/emailHelpers');
104          var hooksHelper = require('*/cartridge/scripts/helpers/hooks');
105
106          var email = req.form.loginEmail;
107          var password = req.form.loginPassword;
108
109          var profileForm = server.forms.getForm('profile');
110
111          var f = req.form;
112          var rememberMe = req.form.loginRememberMe
113              ? (!!req.form.loginRememberMe)
114              : false;
115          var cus= CustomerMgr.getCustomerByLogin( email)
116          var otp = profileForm.customer.otp.htmlValue;
117          var OTPService = require('../scripts/OTPService');
118          var status = OTPService.VerifyOtp(otp,phone);
119          var customerLoginResult;
120
121          if(status.status == 'OK'){
122              customerLoginResult = accountHelpers.loginCustomer(email,password , rememberMe);
123          }
124      }
```

```

Account.js
app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > controllers > JS Account.js > server.post('Login') callback
24 function (req, res, next) {
25   if (customerLoginResult.error) {
26     if (customerLoginResult.status === 'ERROR_CUSTOMER_LOCKED') {
27       var context = {
28         customer: CustomerMgr.getCustomerByLogin(email) || null
29       };
30       var emailObj = {
31         to: email,
32         subject: Resource.msg('subject.account.locked.email', 'login', null),
33         from: Site.current.getCustomPreferenceValue('customerServiceEmail') || 'no-reply@testorganization.com',
34         type: emailHelpers.emailTypes.accountLocked
35       };
36       hooksHelper('app.customer.email', 'sendEmail', [emailObj, 'account/accountLockedEmail', context], function () {});
37     }
38     res.json({
39       error: [customerLoginResult.errorMessage || Resource.msg('error.message.login.form', 'login', null)]
40     });
41     return next();
42   }
43   if (customerLoginResult.authenticatedCustomer) {
44     res.setViewData({ authenticatedCustomer: customerLoginResult.authenticatedCustomer });
45     res.json({
46       success: true,
47       redirectUrl: accountHelpers.getLoginRedirectURL(req.querystring.rurl, req.session.privacyCache, false)
48     });
49     req.session.privacyCache.set('args', null);
50   } else {
51     res.json({ error: [Resource.msg('error.message.login.form', 'login', null)] });
52   }
53   return next();
54 }
55 };

```

Customer Authentication:

- The customer associated with the provided email is retrieved using CustomerMgr.getCustomerByLogin(email).
- The OTP is verified using the OTPService.VerifyOtp(otp, phone) function.
- If the OTP verification is successful (status.status === 'OK'), the customer is authenticated using accountHelpers.loginCustomer(email, password, rememberMe).

Error Handling:

- If there is an error during the login process, such as an invalid OTP or a locked account, appropriate error messages are generated and returned in the response JSON.
- If the login is unsuccessful, an error message is returned.

Script Files:

OTPService.js:

- In **OTPService.js** file we are sending the OTP to the customers and verifying it.

```
OTPService.js
app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > scripts > JS OTPService.js > SendOtp > getOtp > createRequest
1  'use strict';
2
3
4  function SendOtp(num){
5
6
7      var LocalServiceRegistry = require('dw/svc/LocalServiceRegistry');
8      var getOtp = LocalServiceRegistry.createService('app_custom_wellnesswagon.http.otp.post', {
9          createRequest: function (svc, params) {
10              // Set the request parameters
11              var To = '+num';
12              var Channel = 'sms';
13              var sid = 'ACafaa36694bf2694f84b0cfc8bc6ff175';
14              var auth = ' ';
15
16              var encodedAuthStr = require('dw/util/StringUtils').encodeBase64(sid + ':' + auth);
17              svc.addHeader('Content-Type', 'application/x-www-form-urlencoded');
18              var body = "To="+encodeURIComponent('+91'+num)+"&Channel="+encodeURIComponent("sms");
19              // Set the request method
20              svc.setRequestMethod('POST');
21
22              // Return the service instance for the call function to use
23              return body;
24          },
25          parseResponse: function (svc, response) {
26              return response;
27          }
28      });
29      // Call the service
30      var result = getOtp.call();
31
32      return result;
33
34  }
35
36
37
```

```

OTPService.js
app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > scripts > JS OTPService.js > SendOtp > getOtp > createRequest
69
40 function VerifyOtp(otp,phone){
41
42     var LocalServiceRegistry = require('dw/svc/LocalServiceRegistry');
43     var VerifyOtp = LocalServiceRegistry.createService('app_custom_wellnesswagon.http.otpVerify.post', {
44         createRequest: function (svc, params) {
45
46             var sid = 'ACafaa36694bf2694f84b0cfc8bc6ff175';
47             var auth = '2bcc6cafa054a232d333e7fdedb47608';
48
49             var encodedAuthStr = require('dw/util/StringUtils').encodeBase64(sid + ':' + auth);
50             svc.addHeader('Content-Type','application/x-www-form-urlencoded');
51             var body = "To="+encodeURIComponent(phone)+"&Code="+encodeURIComponent(otp);
52             // Set the request method
53             svc.setRequestMethod('POST');
54
55             // Return the service instance for the call function to use
56             return body;
57         },
58         parseResponse: function (svc, response) {
59             return response;
60         }
61     });
62     // Call the service
63     var result = VerifyOtp.call();
64
65     return result;
66 }
67
68
69 module.exports.SendOtp = SendOtp;
70 module.exports.VerifyOtp = VerifyOtp;

```

SendOtp(num):

- This function is used to send an OTP to the specified phone number.
- **Parameters:**
num: The phone number to which the OTP will be sent.
- It utilizes the LocalServiceRegistry to create a service named 'app_custom_wellnesswagon.http.otp.post'.

- Inside the service creation, it defines the request parameters including the destination phone number, channel (e.g., SMS), SID (Service Identifier), and authentication details.
- The service call is made using the call() method.
- **Return Value:** The result of the service call, which typically contains information about the success or failure of the OTP sending process.

VerifyOtp(otp, phone):

- This function is used to verify the OTP received from the user against the phone number.
- Parameters:
 - otp:** The OTP received from the user.
 - phone:** The phone number against which the OTP is to be verified.
- It follows a similar pattern to SendOtp, creating a service named 'app_custom_wellnesswagon.http.otpVerify.post' using LocalServiceRegistry.
- The request parameters include the phone number and OTP to be verified.
- The service call is made using the call() method.
- **Return Value:** The result of the service call, typically indicating whether the OTP verification was successful or not.

Changes made in loginForm.isml:

- In order to receiving the otp from the user we need to modify the loginForm.isml .
- It defines a button for initiating an action and an input field for entering the OTP.
- The JavaScript function handles the button click event and makes an asynchronous request to the server.
- Upon receiving a response from the server, it logs the response to the browser console.as shown in below..

```

loginForm.isml 9+
> app_custom_wellnesswagon > cartridge > templates > default > account > components > loginForm.isml > form.login > div.form-g
1  <form action="{pdict.actionUrl}" class="login" method="POST" name="login-form">
21  <div class="form-group">
22  <button class="btn btn-outline-primary" onclick="executeMultipleRequests()">
23  |   ${Resource.msg('label.input.otp', 'forms', null)}
24  </button>
25  </div>
26  <div class="form-group
27  |   <isif condition="{!!pdict.profileForm.customer.otp.mandatory === false}"></isif>
28  |   <label class="form-control-label" for="registration-form-phone">
29  |   |   <isprint value="{pdict.profileForm.customer.otp.label}" encoding="htmlcontent" />
30  |   </label>
31  |   <input
32  |   |   type="text"
33  |   |   class="form-control"
34  |   |   id="registration-form-phone"
35  |   |   <isprint value="{pdict.profileForm.customer.otp.attributes}" encoding="off" />
36  |   </div>
37
38  <script>
39  |   function executeMultipleRequests() {
40  |   |   // Create XMLHttpRequest objects
41  |   |   var request1 = new XMLHttpRequest();
42  |   |   var request2 = new XMLHttpRequest();
43  |   |   // Specify the URLs for the requests
44  |   |   var url1 = "${URLUtils.url('OTP1-Show')}"
45  |   |   // Configure and send the first request
46  |   |   request1.open('GET', url1, true);
47  |   |   request1.send();
48  |
49  |   request1.onload = function () {
50  |   |   // Handle response for the first URL
51  |   |   console.log('Request 1 completed:', request1.responseText);

```

- The above are the code changes in loginForm.isml template.

Result:

- By following the above we can able to implement multi factor authentication in our ecommerce sites in SFCC as shown below..

Login Page:

Home

LoginCreate Account

Email

Password

Get OTP

OTP

☐ Remember me [forgot password?](#)

Login

Login with Google

Login with Facebook

Check order

See your order even if you are not a registered user. Enter the order number and the billing address ZIP code.

* Order number

* Order Email

* Billing ZIP code

Check status

- If the customer enters login details correctly, then he will be redirect to the account dashboard.

Account Dashboard:

Profile [Edit](#)

First Name

CHENCHU

Last Name

██████████

Email

██████████@gmail.com

Phone

██████████

Password [Edit](#)

Password

Address Book [View](#)

Default Address

hyderabad - Tirupati - 12345

Chenchu Hanuman ██████████


hyderabad

Tirupati, TX 12345

██████████

Order History [View](#)

Most Recent Order



Order Number: 00000701

Date Ordered: 3/12/24

Order Status: NEW

Shipped to: Chenchu Hanuman Bojja

Total items

1

Total

\$30.80

Payment [View](#)

Credit Visa

*****1111

Ending 3/2024

[Add New](#)

ORDER INVOICE DOWNLOAD

Implementing the feature to download the order invoice after placing the order.

Introduction:

Welcome to the documentation for the Order Invoice Download feature in Salesforce Commerce Cloud (SFCC). This feature allows customers to download their order invoices immediately after placing an order on your website.

Feature Description:

The Order Invoice Download feature enables customers to access and download their order invoices directly from their order confirmation page or email. This functionality enhances the post-purchase experience by providing customers with quick and convenient access to their invoices.

Usage Guide:

To download the order invoice after placing an order, customers can follow these steps:

1. After successfully placing an order, customers will be directed to the order confirmation page.
2. On the order confirmation page, a "Download Invoice" button will be available.
3. Clicking the "Download Invoice" button will prompt the invoice to open in PDF format.
4. Customers can then save or print the invoice for their records.

Customization and Integration:

To integrate the feature I SFCC platform we need to do the some of the changes in controllers(Order.js),ISML files(conformation.isml) and script files.

Changes made in Order.js :

```
Order.js
app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > controllers > JS Order.js > server.get('invoice') call
14 server.get('invoice',function(req,res,next){
15
16     var OrderHistory = require('dw/customer/OrderHistory');
17     var arr = Array();
18     var customer = req.currentCustomer.raw;
19
20     while(customer.orderHistory.orders.hasNext()){
21         arr.push(customer.orderHistory.orders.next());
22         break;
23     }
24
25     var OrderMgr = require('dw/order/OrderMgr');
26
27     var invoiceDownloadHelper = require('*/cartridge/scripts/invoiceDownloadHelper');
28     var site = require('dw/system/Site');
29
30     //var orderNo = Order1.currentOrderNo;
31     var fileName = site.current.name
32         + '_' + customer.profile.firstName
33         + '_' + customer.profile.lastName
34         + '.txt';
35
36     //var order = OrderMgr.getOrder(req.form.orderID, req.form.orderToken);
37
38     var invoiceDownload = invoiceDownloadHelper.invoice(arr[0].currentOrderNo);
39
40     res.setHeader(res.base.CONTENT_DISPOSITION, 'attachment; filename="'+ fileName + '"');
41     res.setContentType('application/txt');
42     res.print(invoiceDownload);
43
44     next();
45 })
```

```

JS Order.js
app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > controllers > JS Order.js > server.get('invoice') callba
48 // sending the order invoice to the customer's email
49
50 server.get('Send', function (req, res, next) {
51     try {
52
53         var OrderHistory = require('dw/customer/OrderHistory');
54         var arr = Array();
55         var customer = req.currentCustomer.raw;
56
57         while(customer.orderHistory.orders.hasNext()){
58             arr.push(customer.orderHistory.orders.next());
59             break;
60         }
61
62         var OrderMgr = require('dw/order/OrderMgr');
63
64         var invoiceDownloadHelper = require('*/cartridge/scripts/invoiceDownloadHelper');
65         var site = require('dw/system/Site');
66
67         //var orderNo = Order1.currentOrderNo;
68         var fileName = site.current.name
69             + '_' + customer.profile.firstName
70             + '_' + customer.profile.lastName
71             + '.txt';
72
73         //var order = OrderMgr.getOrder(req.form.orderID, req.form.orderToken);
74
75         var invoiceDownload = invoiceDownloadHelper.invoice(arr[0].currentOrderNo);
76
77
78         res.setHeader(res.base.CONTENT_DISPOSITION, 'attachment; filename="' + fileName + '"');

```

Here's the explanation for each part:

1. **Load Necessary Modules:** Import required modules for interacting with SFCC objects and functionalities.
2. **Retrieve Customer's Order History:** Obtain the current customer's order history and store it in an array named **arr**.
3. **Generate File Name for the Invoice:** Construct the file name for the invoice based on the current site name and the customer's first and last names.
4. **Get Invoice Download Information:** Retrieve invoice download information using the **invoiceDownloadHelper** script and the order number obtained from the customer's order history.
5. **Set Response Headers for File Download:** Set the HTTP headers necessary for initiating a file download in the client's browser.

6. **Sending the Invoice to the Customer via Email:** Use the `emailHelper` script to send the invoice to the customer's email address. Define the sender, recipient, subject, template, and context for the email.
7. **Handle Errors Gracefully:** Catch any exceptions that may occur during the process and return an error response.
8. **Next Middleware Function:** Invoke the next middleware function in the server chain.

Script files:

5 invoiceDownloadHelper.js X

app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > scripts > JS invoiceDownloadHelper.js > invoice

```
1  'use strict';
2  var ISML = require('dw/template/ISML');
3
4  function invoice(orderNo){
5
6      var OrderMgr = require('dw/order/OrderMgr');
7      var Order = OrderMgr.getOrder(orderNo);
8
9      var order = OrderMgr.getOrder(orderNo);
10     var inv = {};
11     inv.BillingAddress = order.billingAddress;
12     inv.customerName = order.billingAddress.fullName;
13     inv.mobile= order.billingAddress.phone;
14
15
16     var invoiceText = "***** Invoice *****\n" +
17         "Customer Name: " + inv.customerName + "      "+"Invoice No: "+order.invoiceNo+"\n"
18         "Mobile Number: " + inv.mobile + "\n" +
19         "Billing Address:\n" +
20         "-----\n" +
21         "    Address 1  : " + order.billingAddress.address1 + "\n" +
22         "    Address 2  : " + order.billingAddress.address2 + "\n" +
23         "    City       : " + order.billingAddress.city + "\n" +
24         "    State      : " + order.billingAddress.stateCode + "\n" +
25         "    Zip Code   : " + order.billingAddress.postalCode + "\n" +
26         "    Order No   : " + orderNo + "\n" +
27         "-----\n" +
28         "Products:\n";
29
30     var gtotal=0;
31
32     var lineItemsArray = order.allProductLineItems.toArray();
```

```

invoiceDownloadHelper.js X
app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > scripts > JS invoiceDownloadHelper.js > ...
4 function invoice(orderNo){
51
52     var lineItemsArray = order.allProductLineItems.toArray();
53
54     // Use forEach on the JavaScript array
55     invoiceText += "Product Name\t\t\t\t\t Quantity | Price | Total \n";
56     invoiceText += "-----\n";
57
58     lineItemsArray.forEach(function(lineItem) {
59         // Format each line in a table-like structure
60         invoiceText += `${lineItem.productName}\t\t\t\t\t ${lineItem.quantity}\t\t\t\t\t ${lineItem.basePrice.value}\t\t\t\t\t ${lineItem.quantity * lineItem.basePrice.value}\n`;
61
62         // Update the grand total
63         gtotal = order.originalOrder.totalGrossPrice;
64     });
65
66     // Add a separator line and display the grand total
67     invoiceText += "-----\n";
68     invoiceText += "                '+' Shipping Cost: ${order.shippingTotalPrice}\n";
69     invoiceText += "                '+' Total Tax : ${order.totalTax}\n";
70     invoiceText += "                '+' Grand Total : ${gtotal}\n";
71     invoiceText += "*****\n";
72     invoiceText += "    Thank you for your order";
73
74     return invoiceText;
75 }
76 module.exports.invoice = invoice;

```

Here's the explanation for each part:

1. Load Necessary Modules:

- The script starts by loading the necessary modules, including the OrderMgr module from Salesforce Commerce Cloud.

2. Retrieve Order Details:

- The function retrieves the order details based on the provided order number using the OrderMgr module.

3. Define Invoice Object:

- It defines an object named **inv** to hold invoice details such as the billing address, customer name, and mobile number.

4. **Construct Invoice Text:**

- The function constructs the text for the invoice, including customer details, billing address, and order summary.

5. **Iterate Over Line Items:**

- It iterates over each line item in the order and appends the product name, quantity, price, and total to the invoice text.

6. **Calculate Grand Total:**

- The function calculates the grand total by summing up the total gross price of the order.

7. **Display Additional Order Details:**

- It displays additional order details such as shipping cost, total tax, and grand total in the invoice text.

8. **Export the Function:**

- Finally, the invoice function is exported to make it accessible to other modules within the application.

This function essentially generates the text for an invoice based on the provided order number. It retrieves order details, calculates totals, and formats the information into a printable invoice format.

Template changes:

- For providing the buttons in the order conformation page we need to modify the code in checkout/conformation.isml template.
- In this conformation.ismlfile we are giving two of the buttons, one is for downloading the order invoice and another one is for sending the order invoice to the customers email.

```


confirmation.isml 9 X
app_custom_wellnesswagon > cartridges > app_custom_wellnesswagon > cartridge > templates > default > checkout > confirmation > confirmation.isml > isdecorate > div.container.receipt <isif.condition= > div.row
1  <isdecorate template="common/layout/page">
15  <div class="container receipt <isif condition="{pdict.order.shipping.length > 1}">multi-ship</isif">
16  <div class="row">
17  <div class="{pdict.returningCustomer ? 'col-sm-6 offset-sm-3' : 'col-sm-6 offset-sm-3 offset-md-0'}">
22  </div>
23  </div>
24  <div class="row">
25  <isif condition="{pdict.returningCustomer == false && pdict.order.orderEmail}">
26  <div class="col-sm-6 offset-sm-3 offset-md-0 push-md-6">
27  <isinclude template="checkout/confirmation/confirmationCreateAccount" />
28  </div>
29  </isif>
30  <div class="{pdict.returningCustomer ? 'col-sm-6 offset-sm-3' : 'col-sm-6 offset-sm-3 offset-md-0 pull-md-6'}">
31  <isinclude template="checkout/confirmation/confirmationDetails" />
32  <div class="mb-3">
33  <a href="{URLUtils.url('Home-Show')}}" class="btn btn-primary btn-block order-confirmation-continue-shopping" role="button" aria-pressed="true">
34  ${Resource.msg('button.continue.shopping', 'confirmation', null)}
35  </a>
36  <div>
37  </div>
38  </div>
39  <div class="row mb-3 clearfix hidden-sm-down">
40  <div class="col">
41  <div>
42  <a href="{URLUtils.url('Order-invoice')}}" class="btn btn-outline-primary pull-right"
43  role="button" aria-pressed="true">
44  ${Resource.msg('button.download.invoice', 'invoicedownload', null)}
45  </a>
46  </div>
47  </div>
48  </div>
49  </div>
50  <a href="{URLUtils.url('Order-Send')}}" class="btn btn-outline-primary pull-left"
51  role="button" aria-pressed="true">
52  ${Resource.msg('button.download.invoiceEmail', 'invoicedownload', null)}
53  </a>
54  </div>
55  </div>
56  </div>
57  </div>
58  </div>
59  </div>
60  </div>
61  </div>
62  </div>
63  </div>
64  </div>

```

This code snippet combines the functionality of sending the order invoice as an email attachment with the previous feature of enabling customers to download the invoice directly from the website.

Results:

Order conformation page:

1 Items		\$25.00
Energy Drink - Kiwi Guava		
		
Quantity	Total	
1	\$25.00	

Subtotal	\$25.00
Shipping	\$3.00
Sales Tax	\$2.80
Total	\$30.80

Continue Shopping

Invoice Email

Download invoice

- If we click on Download invoice button ,then the order invoice will be downloaded directly.
- If we click on Invoice Email button ,then the order invoice will be send to the customer's email.

Order Invoice:

WellnessWagon_CHENCHU_Bojja X +

File Edit View

***** Invoice *****

Customer Name: Chenchu Hanuman Bojja Invoice No: 00004001

Mobile Number: 09234567890

Billing Address:

Address 1 : hyderabad

Address 2 : null

City : Tirupati

State : TX

Zip Code : 12345

Order No : 00000701

Products:

Product Name	Quantity	Price	Total
Energy Drink - Kiwi Guava	1	25	25

Shipping Cost: 3

Total Tax : 2.8

Grand Total : 30.8

Thank you for your order

- The order invoice will be looks like the above.