**USER EVENT SCRIPT ASSIGNMENT**

1. What is user event script?

1. A User Event Script is a type of script in NetSuite SuiteScript, where we can perform some action when records are **created**, **updated** and **deleted**.
2. It is executed in **NetSuite application Server**.
3. It manipulates the **database** to operate some operations like Create, Edit and Delete.
4. These scripts are triggered when some events are applied on some record like when a user creates a new customer, updates something in the sales order or deletes some invoice.

2. What is the purpose of user event scripts?

The purpose of user-event script can be -

1. Let we want to **automate** some tasks when a record like sales order got saved.
2. It will help to **update** some field value based on the client’s requirement.
3. We have to **validate** the data we entered in a field, which should meet the business requirement.
4. It implements real time **data synchronization**.
5. It **triggers workflows** when some event occurs. For ex – A mail should be automatically sent to the customer when the invoice gets generated.

3. What are the user event script functions?

There are three main functions in a User Event Script, each of which will get triggered by some certain event that we write in the script.

* ***beforeLoad*** :
  + - It runs before the record is loaded or displayed in the user interface.
    - It helps to modify the fields in a record.
    - It also helps to hide some fields from the user.

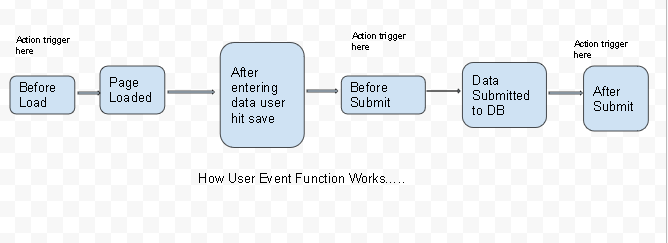
Ex – We can hide some sensitive fields for specific roles or users.

* ***beforeSubmit*** :
  + - It runs before the record is saved in the Database.
    - We use this function to validate or modify some data before it gets saved to the database through NetSuite application server.

Ex – We can validate all the fields we made the change upon using certain criteria.

* ***afterSubmit*** :
  + - It runs after the record is saved in the Database.
    - It is used to trigger some process or actions after the record gets saved in the function like sending mails.
    - Ex – Sending an email after a sales order is created.

You can refer to the image below for better understanding.



4. Can we call a user event script from other scripts?

Yes, we can call a user event script from other scripts as well but there are some limitations over there.

* We generally know that user event script gets triggered when we do some operation like create, edit, delete with the record and they are not designed to directly invoke as we invoke a function in our code.
* The user event script is typically associated with specific actions with the record that we do like beforeSubmit, afterSubmit etc.
* So, if you are working with multiple scripts, make sure to manage the module, entry points and dependencies correctly using the define function in SuiteScript 2.X.

By following these steps, we can indirectly call the user event script.

5. Can we execute user event script as admin role?

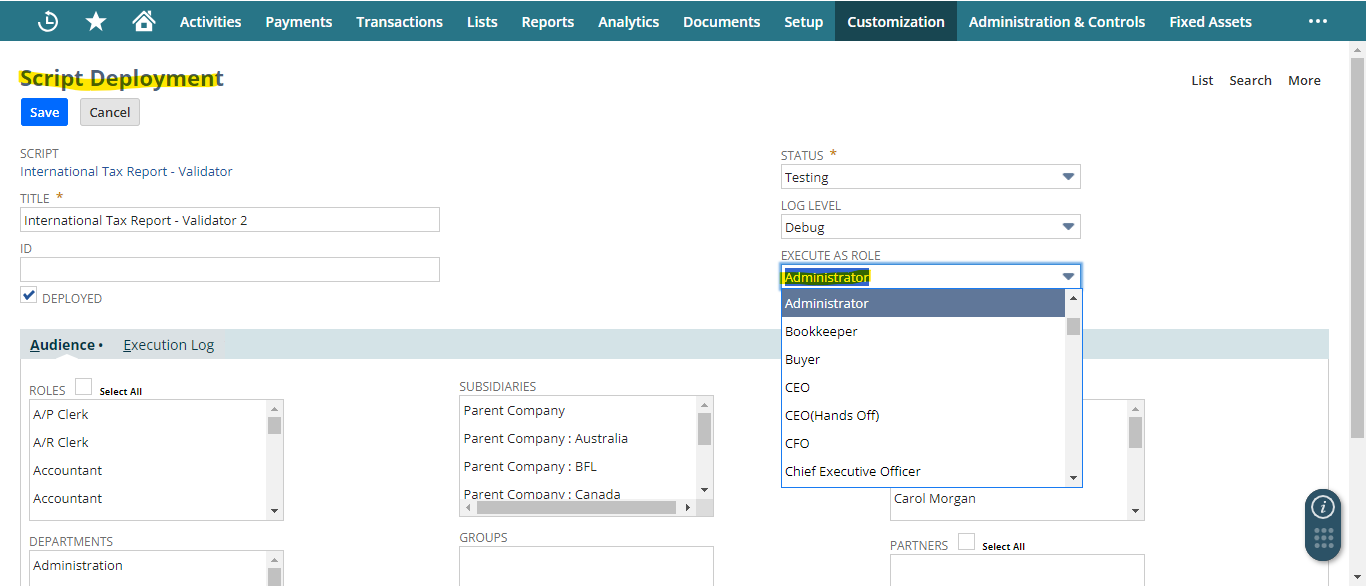
Yes, we can execute a User Event Script as then admin role in NetSuite.

* User Event Script runs in the context of the user who triggered the event. So, if a non admin user performed some action like create, update or delete, that will also execute by the user’s permission only.
* We can set the execution role for the script deployment to Administrator during the script deployment process.

Step 1 – Go to Customization -> Scripting -> Script Deployments.

Step 2 – Edit the deployment record for user event script.

Step 3 – In the Execute as role field, select Administrator and click on Save.



6. What is the governance limit for user event scripts?

* The governance limit for a User Event Script is **1,000 units.**
* If the script exceeds this limit, it will throw a “**SSS\_USAGE\_LIMIT\_EXCEEDED**” error.
* To stay in the limit, optimize the script as much as you can, remove unnecessary operations.

7. How can we create custom button using user event script?

function beforeLoad(scriptContext) {

var rec = scriptContext.record;

if(scriptContext.type === scriptContext.UserEventType.CREATE) {

var form = scriptContext.form; //

form.addButton({

id: 'custpage\_buttonid',

label: 'Click Me'

});

}

}

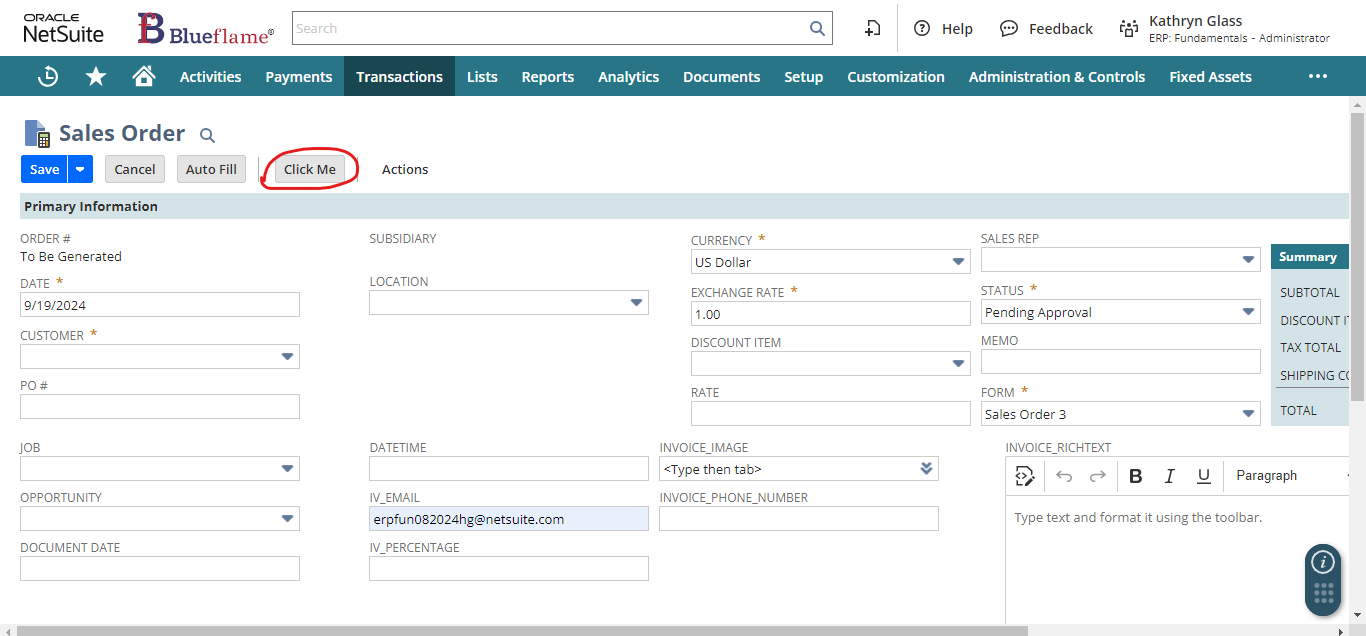
return {

beforeLoad: beforeLoad,

beforeSubmit: beforeSubmit,

afterSubmit: afterSubmit

};



8. How can we prevent users from submitting a record?

9. Is there any limit for the number of user event scripts that can be deployed in one record?

10. How to execute one user event script after or before another?

11. Can I attach a client script in a user event script?

12. How can we get the execution context?

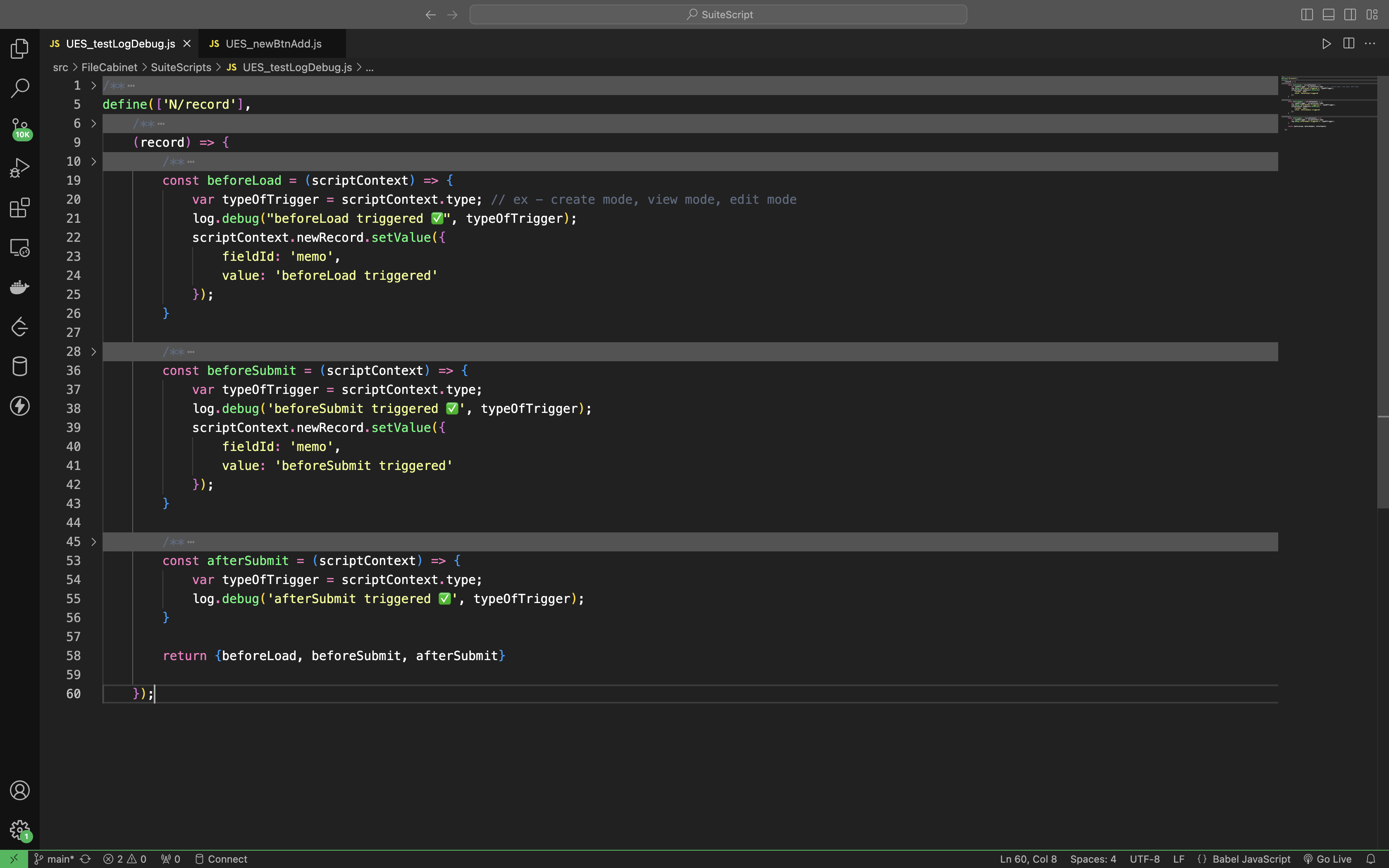
13. Is there any API that can't be used in a user event script?

14. Use log.debug to notice the type of three events that is beforeLoad, beforeSubmit, afterSubmit in the user event subscript.

Here I have implemented the three entry points of user event script and deployed this on ‘invoice’ record in the memo field. Then I used **scriptContext.type** to see in which mode we are accessing that record.

Now, we can see everything under execution log subtab in that particular script.

***Code:***



***Output:***

15. Related to the newRecord and oldRecord.

16. Email and Phone number of customer should be populated on the transaction before the loading of the record.

Here I used N/record and N/search module to find that particular record in the context and search module to Search for a single record using keywords.

-> I also used an API called lookupFields from search module that Performs a search for one or more body fields on a record. Returns select fields as an object with value and text properties. Returns multiselect fields as an object with value:text pairs.

const beforeLoad = (scriptContext) => {

log.debug('beforeLoad triggered ✅');

var current\_record = scriptContext.newRecord; // Get the current record. i.e. invoice

// var customerId = current\_record.getValue({ // Get the customer

// fieldId: 'entity' // 'entity' is the field where the customer is stored

// });

var customerId = 1126;

log.debug('Customer ID:', customerId);

if (customerId) {

log.debug('Customer found ✅', customerId);

var fields = search.lookupFields({

type: search.Type.CUSTOMER,

id: customerId,

columns: ['email', 'phone'] // Fetch email and phone, it returns an object

});

log.debug('Customer Fields (Email and Phone):', fields); // Log the returned fields for email and phone

// Set the email in the custom field

current\_record.setValue({

fieldId: 'custbody3',

value: fields.email || '' // if email is available then set, else set empty string

});

// Set the phone number in the custom field

current\_record.setValue({

fieldId: 'custbody7',

value: fields.phone || ''

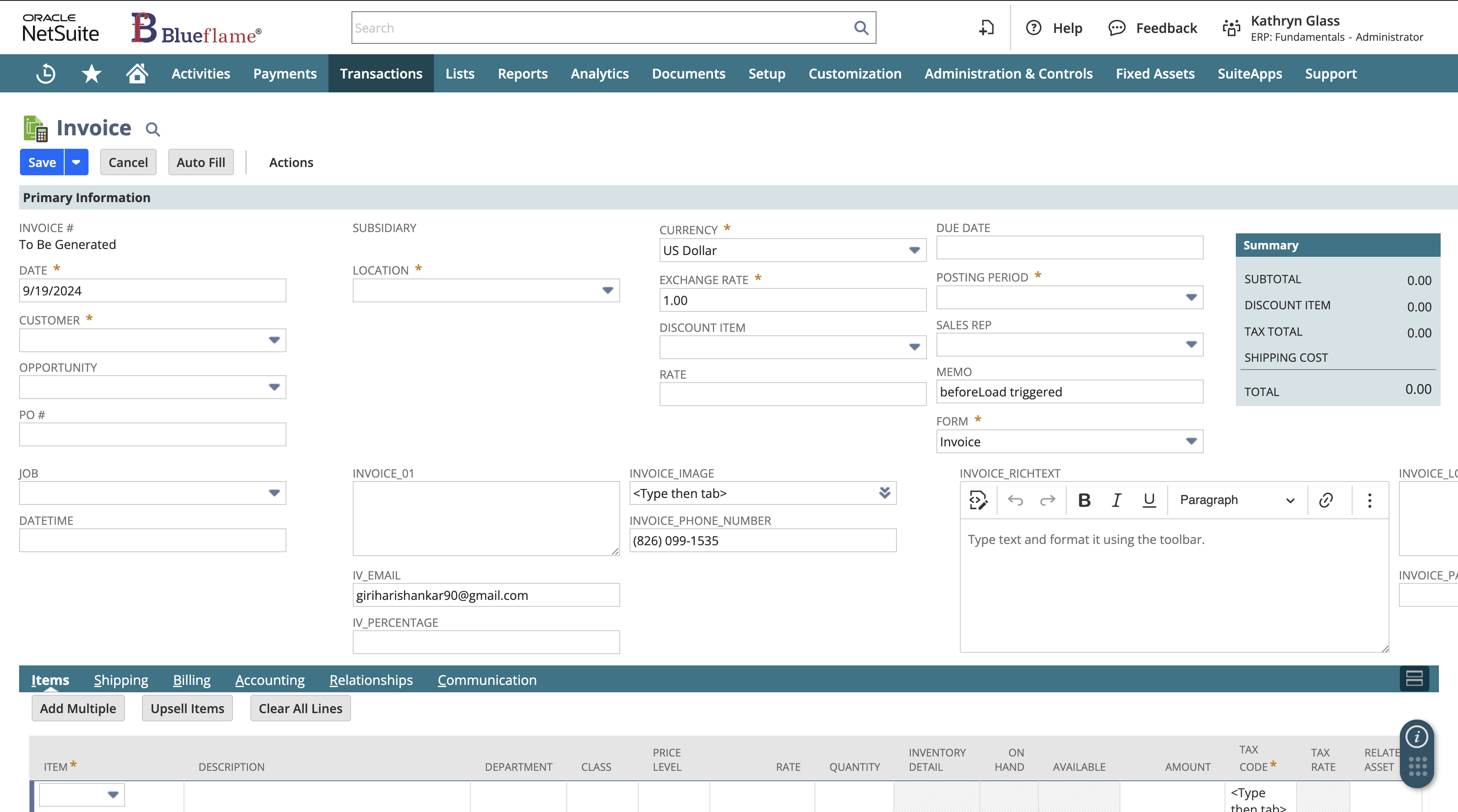
});

} else {

log.debug('Customer not found ❌');

}

}



17. Like the above one but now it should get populated dynamically after the record is submitted.

* This script runs after the record has been submitted (afterSubmit), so the email and phone fields are dynamically populated based on the selected customer.
* The script fetches the customer ID from the entity field, which holds the customer on the transaction (eg. Invoice)
* It then uses search.lookupFields() to retrieve the email and phone of the customer dynamically.
* The record.submitFields() method is used to update the transaction fields (e.g., custbody3 for email and custbody7 for phone) with the fetched customer information after the record is saved.

const afterSubmit = (scriptContext) => {

if (scriptContext.type !== scriptContext.UserEventType.CREATE &&

scriptContext.type !== scriptContext.UserEventType.EDIT) {

return;

}

log.debug('afterSubmit triggered ✅');

var current\_record = scriptContext.newRecord; // Get the current transaction record (e.g. Invoice)

var customerId = current\_record.getValue({ // Fetch the customer ID from the 'entity' field (the customer on the transaction)

fieldId: 'entity' // 'entity' is the field that holds the customer

});

log.debug('Customer ID:', customerId);

if (customerId) { // Checking if a customer is selected or not

log.debug('Customer found ✅', customerId);

// Lookup customer's email and phone using the customer ID

var fields = search.lookupFields({

type: search.Type.CUSTOMER, // Customer record type

id: customerId, // Use the dynamically fetched customerId

columns: ['email', 'phone'] // Fetch email and phone

});

log.debug('Customer Fields (Email and Phone):', fields);

record.submitFields({ // Update the transaction with the customer's email and phone

type: current\_record.type, // The type of the current transaction (e.g., Sales Order or Invoice)

id: current\_record.id, // The internal ID of the current transaction

values: {

custbody3: fields.email || '',

custbody7: fields.phone || ''

}

});

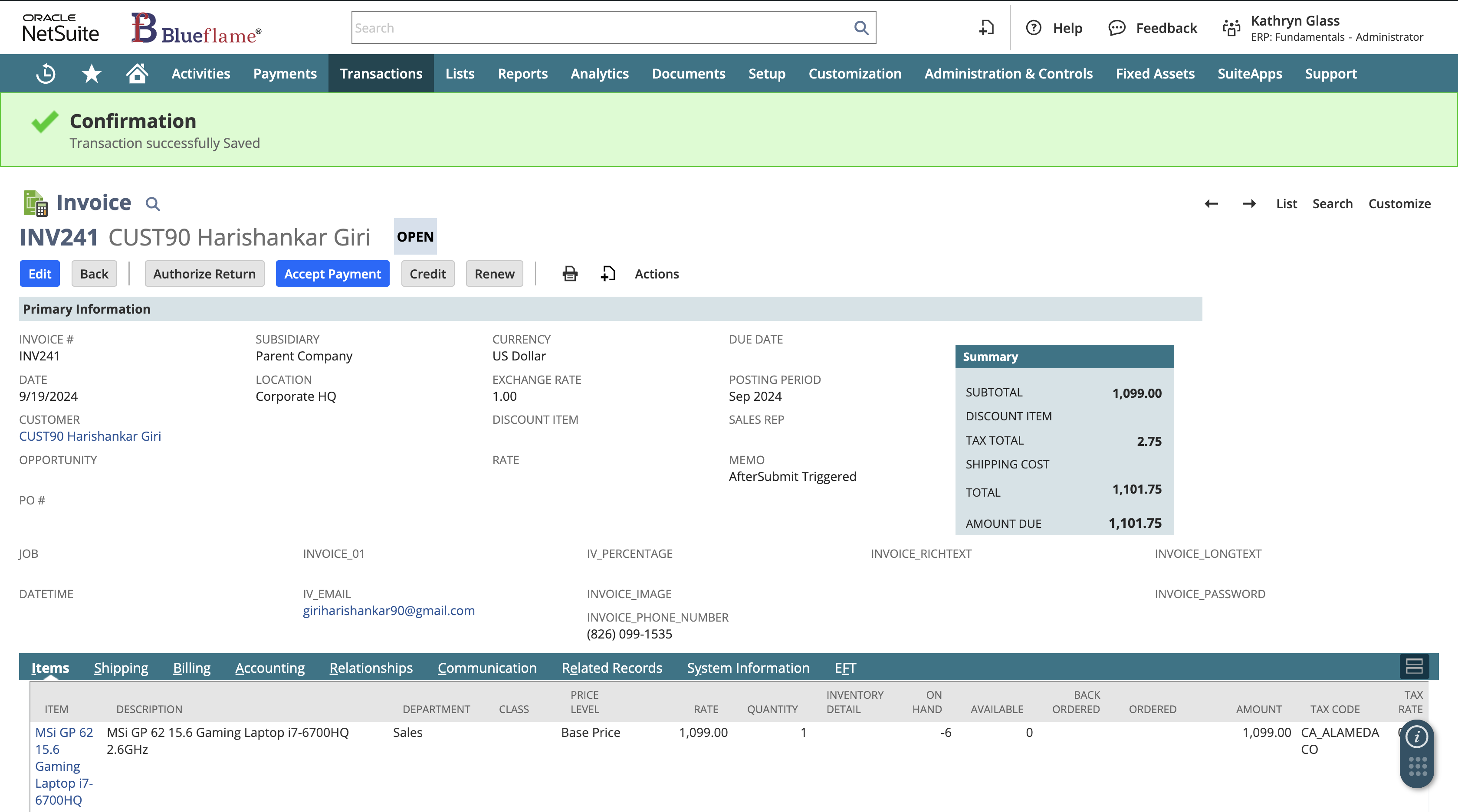
log.debug('Email and Phone updated ✅');

} else {

log.debug('No customer found ❌');

}

};



18. Make the above question dynamic (customer chosen's email and phone number needs to be get reflected).