Use Case

Round Hole Projects is a spin-off division of Square Peg Accounting, soon to be a Big 5 accounting firm. Square Peg Accounting has been growing rapidly since moving many of their customers' legacy systems to the cloud. The Round division provides project management services to Square Peg customers. More forward-thinking than its parent company, Round's motto is "cloud first"—the firm is proud of its 100 percent cloud-based infrastructure.

Round uses Salesforce to track its customers and project opportunities for its professional services operation. When Round successfully closes a deal, the project is tracked in Square Peg's project management system. And when Round completes a project (it has a 100 percent success rate to date!), Square Peg sends out the bill (it's what the firm does best).

Ever heard the saying that a blacksmith's house has only wooden spoons? No? Well Square Peg is a classic example. The staff have been so busy supporting their customers that they haven't had time to migrate their own critical back office systems to the cloud. The project management and billing systems are still maintained on premise. Worse yet, Round staff have no direct access. #notWinning

This is something that Round plans to help Square Peg with, but that's a long-term project. In the meantime, the best Round can do is to build a variety of SOAP, REST, and OData web services to allow external integrations.

Happily, that's where you come in...

Round hires you, a data integration specialist, to implement the first phase of data-centric integrations to support its project management and billing processes. Your job is to apply the appropriate integration mechanisms necessary to connect with each legacy system and synchronize data as needed.

You meet with the key stakeholders and compile a comprehensive set of integration requirements. You also gather security and authentication information, and identify the specific integration mechanisms available for secure access with each on-premise legacy system and their respective supporting services.

Standard Objects

You work with the following standard objects:

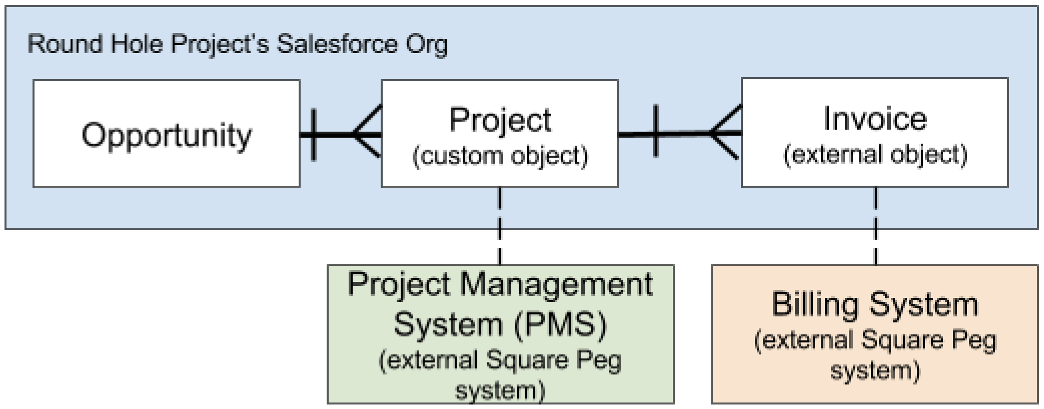
- Opportunity—Deals related to Round's professional services packages.

Custom Objects

- Project—Represents a synced partial copy of the project record in Square Peg's external project management system (PMS). It's a child of the Opportunity object and contains just enough information for Round to work the project.

External Objects

- Invoice—A read-only external object, connected in real time to the "invoices" table in the external Square Peg billing system. It's a child of the Project object.


Round Hole Project's Salesforce Org
Opportunity
Project (custom object)
Invoice (external object)
Project Management System (PMS) (external Square Peg system)
Billing System (external Square Peg system)

Business Requirements

This section represents the culmination of your meetings with key stakeholders. It's your blueprint to implement all click and code integrations to support Round's business processes.

Configure Integration Security

Your outbound and inbound integrations include SOAP and REST web services. Configure your org with information regarding each target endpoint and the source of any inbound integrations, as well as associated authentication data.

Connect to Square Peg's Billing System

The billing system has a **SOAP-based** endpoint; always whitelist it as a **Remote Site**. Its authentication model uses simple user and password credentials passed as parameters in the service call, which you store and fetch from **Custom Settings**. Next, you consume a provided WSDL and generate a proxy class to call the service.

Use the following endpoint information to configure a Remote Site Setting:

Remote site name	BillingService
Remote site URL	http://sb-integration-bs.herokuapp.com
Active	Checked

Additional Billing System API Details

Return Status	'OK' 'UNAUTHORIZED'

In addition, the billing service is secured with user and password credentials. Store them as a ServiceCredentials Custom Setting (named BillingServiceCredential), and pass to the outbound call:

Username	bsUser1
Password	bsPass1

*Note: While it is best practice to encrypt credentials, for the purpose of this superbadge store them in clear text. *

Connect to Square Peg's Project Management System (PMS)

The PMS uses a secured **REST-based** API; configure a **Named Credential** containing these authentication and endpoint details:

Label	ProjectService
Name	ProjectService
URL	https://sb-integration-pms.herokuapp.com/projects
Identity Type	Named Principal
Authentication Protocol	Password Authentication
Username	pmsUser1
Password	pmsPass1
Generate Authorization Header	Checked

Additional Project Management System API Details

HTTP Method	POST
Required Headers	Content-Type: JSON
Return Status	'OK' or error message
Response Code	201 = Success 500 = Failure

The **external** PMS service calls your org's custom Apex REST service back; configure it as a **Connected App** with the following information:

Connected App Name	ProjectService
API Name	ProjectService
Contact email	Your email
Enable OAuth Settings	Checked
Callback URL	https://sb-integration-pms.herokuapp.com/oauth/_callback
Selected OAuth Scopes	Full access & Perform requests on your behalf at any time (refresh_token, offline_acce

Register the generated **Consumer Key** and **Consumer Secret** with the Square Peg security whitelisting process.

Configure Org Registration

You will use a custom Heroku app to register your org by clicking this link: https://sb-integration-pms.herokuapp.com. This app registers your username with the Connected App’s consumer key and consumer secret in the Square Peg registry. This process returns a security token to store in your org as a **ServiceTokens** Custom Setting record (named **ProjectServiceToken**). Pass the token with any outbound call to the PMS RESTful web service as part of the header (with key named **token**). Follow the instructions on the registration app, and remember to test your connection. (Note: When you begin testing, bear in mind that it can take some time for your Connected App to propagate.)

Synchronize Outbound Project Data

Business growth is growing fast! While this keeps the staff busy (which they love), it also means they haven’t had the time to implement their own management systems. At this point, they just want to implement a bare-bones integration from their Salesforce org to sync and track minimal project and billing information with Square Peg’s external systems.

When Round wins a new project opportunity, create a project record in the external PMS so the parent company can track the work:

- When an existing opportunity’s type is a **New Project** and it’s stage is changed to **Closed Won**, make an outbound RESTful call. Then send a POST request to the service endpoint with a JSON packet containing the fields OpportunityId, OpportunityName, AccountName, CloseDate, and Amount.
- Use the preconfigured **ProjectService** Named Credential previously defined.
- Match the name and type of the fields based on the following sample JSON mapped to the Opportunity record.

Sample POST JSON

```
{
 "opportunityId": "00641000004EKmFAAW",
 "opportunityName": "Super-Duper Big Deal",
 "accountName": "ACME Corp, Inc.",
 "closeDate": "2016-10-30",
 "amount": 150000
}
```

Copy

Create "Update Opportunity" Process

Round want’s you to use low-code solutions where possible—use Process Builder to create a new **Update Opportunity** process to call an Apex action (named **Post Opportunity To PMS**) to pass the Opportunity ID to the Apex logic that makes the callout.

Implement a method (named **PostOpportunityToPMS**) in an Apex class (named **ProjectCalloutService**), and invoke it from the process action. Ensure your method gets the necessary opportunity data and invokes an authenticated REST callout. We want to design for potential future enqueuing inside other asynchronous jobs, so implement asynchronous logic with queueable Apex in an inner class (named **QueueablePMSCall**) inside **ProjectCalloutSevice** to execute the callout logic.

In addition, include the Square Peg registration token you got during the registration process in the header of your service call with the key as "token"—this identifies your org.

If the call is successful, set the opportunity Stage to **Submitted Project**. However, if it’s not successful, set it to **Resubmit Project**, which lets a user reattempt the process.

To support these requirements, add **New Project** as an Opportunity Types value. Add the following values to opportunity Stage.

Stage Name	**Probability**	**Type**
Submitted Project	100	Closed/Won
Resubmit Project	100	Closed/Won

Note that this process is not designed to operate in bulk. It’s designed to only process the first ID passed even if activated on a bulk load of opportunities.

Synchronize Inbound Project Data

A successful call to the PMS service creates a new (or updates an existing) project record in the external system’s database. This triggers the PMS to make a RESTful call back to Round’s Salesforce org with a subset of the new Project record’s data. The inbound call’s authentication with Salesforce is dependent upon the **Org Registration** of the **Connected App** authentication keys.

Implement an Apex REST Service class (named **ProjectRESTService**) with a method (named **postProjectData**) that accepts the post and inserts (or updates) a corresponding project record associated with the opportunity from which it originated. The inbound call to:

https://YOUR_INSTANCE.salesforce.com/services/apexrest/project contains a data packet for the following elements closely matching the names and types of the Project custom object fields as defined in the table below.

The tech lead has insisted that you must use parameters on the service method for fetching the fields in the JSON body rather than using Apex deserialization, (she wants them clearly identified in the method signature and considers it a simpler implementation pattern.)

The signature will include the following parameters in the following sequence:

Field Type	Field Name
String	ProjectRef
String	ProjectName
String	OpportunityId
Date	StartDate
Date	EndDate
Double	Amount
String	Status

Ensure the service method in the **ProjectRESTService** class creates (or updates) a project record in Salesforce using the passed field values from the external system. In addition, add logic to update the **Delivery/Installation Status** field as **In Progress** on the associated opportunity record. Create the project record as a child of the associated opportunity. To safeguard data integrity, ensure all data operations are managed in an explicit single transaction using a savepoint. The method should return a string value of "OK" or an error message to the calling service.

To support these requirements, you need to expose Projects by adding the Project Tab to the Sales application and adding the Project Related List to the Opportunity page layouts.

Synchronize Outbound Billing Data

Round wants to use its Salesforce org to notify Square Peg when a project is ready to bill. Your task is to trigger an outbound SOAP call anytime the project **Status** in their Salesforce org is set to **Billable**. This then triggers Square Peg's legacy billing system to create a new invoice and bill the customer.

Use the existing trigger (named **ProjectTrigger**) to invoke the callout that creates the invoice, and pass the **ProjectRef** (the external ID) and **Amount** to help with invoice creation.

You follow coding best practices and ensure all business logic is kept out of triggers. Instead, you use the trigger to call a class (named **BillingCalloutService**) with a method (named **callBillingService**). Your class method tests whether to execute the integration based on the criteria defined above. If so, use asynchronous (rather than queueable) Apex to make the callout from within the same class.

The BillingService is exposed through a SOAP API. So consume the WSDL provided by the billing system's IT team, and generate a proxy class (named **BillingServiceProxy**) to use for your callout. There is only one service method definition. It requires you to pass the following arguments:

- ProjectRef (the external ID)
- BillingAmount

- Billing Service user credentials that you previously configured in a custom setting

If the outbound call is successful, the project record's status is updated to **Billed**, and the external billing system generates a project invoice.

Synchronize Inbound Billing Data

The IT team has already started building additional integration services around the billing system, and has begun with an OData provider to expose invoice data in a read-only mode. Your task is to leverage that endpoint to provide real-time billing information within Round's Salesforce org.

As part of Round's no-code/low-code values, use Salesforce Connect to configure real-time access to invoices generated as result of your billing integration. Display appropriate invoices in a related list on the Salesforce project detail page.

The IT team has given you the OData endpoint and authentication information (see chart) to make the necessary configurations to map and expose fields from the Invoice table.

Use the following to configure your external data source connection, accept all other default values:

External Data Source	BillingService
Name	BillingService
Type	Salesforce Connect OData 2.0
URL	https://sb-integration-is.herokuapp.com/odata
Identity Type	Anonymous
Authentication Protocol	No Authentication

Once saved, you can Validate and Sync, which creates an external object named "invoices". Configure an indirect relationship from the external Invoice object to its parent project inside your org, and ensure that invoice data is visible from its parent project detail page. The related list should only display: External ID, Bill Amount, and Bill Date.

Create Unit Tests

As a seasoned integration specialist you understand the benefits of good unit tests. Build tests that test both failed and passing tests using service mocking. **Important:** Make sure that you chose 'Run All' tests in the Developer Console at least once before attempting to verify this challenge.