Salesforce Developer Interview Kit - v3
System Design, Advanced Topics & Recent Updates
For developers with 8+ years experience returning to development roles

---

Table of Contents

Hands-on Coding Challenges

Q15: Design a trigger framework that prevents recursive calls and supports multiple handlers.
Answer:
```apex
// Trigger Framework Handler
public abstract class TriggerHandler {

    // Static map to prevent recursion
    private static Map<String, LoopCount> loopCountMap;
    private static Set<String> bypassedHandlers;

    static {
        loopCountMap = new Map<String, LoopCount>();
        bypassedHandlers = new Set<String>();
    }

    // Current trigger context
    protected TriggerContext context;

    // Constructor
    public TriggerHandler() {
        this.setTriggerContext();
    }

    // Main entry point for triggers
    public void run() {
```

```
        // Check if this handler should be bypassed
        if (bypassedHandlers.contains(getHandlerName())) {
            return;
        }

        // Check for recursion
        if (this.context != null && this.context.isExecuting) {
            this.addToLoopCount();

            // Check max loop count
            if (this.context.isBefore && this.getMaxLoopCount() >= 0 && this.getLoopCount() >
this.getMaxLoopCount()) {
                String message = 'Maximum loop count of ' + String.valueOf(this.getMaxLoopCount())
+ ' reached in ' + getHandlerName();
                throw new TriggerException(message);
            }

            if (this.context.isAfter && this.getMaxLoopCount() >= 0 && this.getLoopCount() >
this.getMaxLoopCount()) {
                String message = 'Maximum loop count of ' + String.valueOf(this.getMaxLoopCount())
+ ' reached in ' + getHandlerName();
                throw new TriggerException(message);
            }

            // Execute the appropriate method
            if (this.context.isBefore) {
                this.beforeInsert();
                this.beforeUpdate();
                this.beforeDelete();
            }

            if (this.context.isAfter) {
                this.afterInsert();
                this.afterUpdate();
                this.afterDelete();
                this.afterUndelete();
            }
        }
    }

    // Bypass methods
    public static void bypass(String handlerName) {
        bypassedHandlers.add(handlerName);
    }
```

```
public static void clearBypass(String handlerName) {
    bypassedHandlers.remove(handlerName);
}

public static Boolean isBypassed(String handlerName) {
    return bypassedHandlers.contains(handlerName);
}

public static void clearAllBypasses() {
    bypassedHandlers.clear();
}

// Virtual methods to be overridden
protected virtual void beforeInsert() {}
protected virtual void beforeUpdate() {}
protected virtual void beforeDelete() {}
protected virtual void afterInsert() {}
protected virtual void afterUpdate() {}
protected virtual void afterDelete() {}
protected virtual void afterUndelete() {}

// Override this to set max loop count (default is 5)
protected virtual Integer getMaxLoopCount() {
    return 5;
}

// Private methods
private void setTriggerContext() {
    this.context = new TriggerContext(
        Trigger.isExecuting,
        Trigger.isInsert,
        Trigger.isUpdate,
        Trigger.isDelete,
        Trigger.isUndelete,
        Trigger.isBefore,
        Trigger.isAfter,
        Trigger.size
    );
}

private void addToLoopCount() {
    String handlerName = getHandlerName();
    if (loopCountMap.containsKey(handlerName)) {
```

```
            Boolean exceeded = loopCountMap.get(handlerName).increment();
            if (exceeded) {
                Integer max = this.getMaxLoopCount();
                throw new TriggerException('Maximum number of trigger loops exceeded: ' +
handlerName + ' (max: ' + max + ')');
            }
        } else {
            loopCountMap.put(handlerName, new LoopCount(this.getMaxLoopCount()));
        }
    }

    private Integer getLoopCount() {
        String handlerName = getHandlerName();
        if (loopCountMap.containsKey(handlerName)) {
            return loopCountMap.get(handlerName).getCount();
        } else {
            return 0;
        }
    }

    private String getHandlerName() {
        return String.valueOf(this).substring(0, String.valueOf(this).indexOf(':'));
    }

    // Inner classes
    private class LoopCount {
        private Integer max;
        private Integer count;

        public LoopCount() {
            this.max = 5;
            this.count = 0;
        }

        public LoopCount(Integer max) {
            this.max = max;
            this.count = 0;
        }

        public Boolean increment() {
            this.count++;
            return this.exceeded();
        }
```

```
        public Boolean exceeded() {
            return this.max >= 0 && this.count > this.max;
        }

        public Integer getCount() {
            return this.count;
        }

        public Integer getMax() {
            return this.max;
        }
    }

    private class TriggerContext {
        public Boolean isExecuting { get; set; }
        public Boolean isInsert { get; set; }
        public Boolean isUpdate { get; set; }
        public Boolean isDelete { get; set; }
        public Boolean isUndelete { get; set; }
        public Boolean isBefore { get; set; }
        public Boolean isAfter { get; set; }
        public Integer size { get; set; }

        public TriggerContext(Boolean isExecuting, Boolean isInsert, Boolean isUpdate,
                              Boolean isDelete, Boolean isUndelete, Boolean isBefore,
                              Boolean isAfter, Integer size) {
            this.isExecuting = isExecuting;
            this.isInsert = isInsert;
            this.isUpdate = isUpdate;
            this.isDelete = isDelete;
            this.isUndelete = isUndelete;
            this.isBefore = isBefore;
            this.isAfter = isAfter;
            this.size = size;
        }
    }

    public class TriggerException extends Exception {}
}

// Example implementation for Account
public class AccountTriggerHandler extends TriggerHandler {

    private List<Account> newAccounts;
```

```
    private List<Account> oldAccounts;
    private Map<Id, Account> newAccountMap;
    private Map<Id, Account> oldAccountMap;

    public AccountTriggerHandler() {
        super();
        this.newAccounts = (List<Account>) Trigger.new;
        this.oldAccounts = (List<Account>) Trigger.old;
        this.newAccountMap = (Map<Id, Account>) Trigger.newMap;
        this.oldAccountMap = (Map<Id, Account>) Trigger.oldMap;
    }

    protected override void beforeInsert() {
        AccountService.validateRequiredFields(this.newAccounts);
        AccountService.setDefaultValues(this.newAccounts);
    }

    protected override void beforeUpdate() {
        AccountService.validateBusinessRules(this.newAccounts, this.oldAccountMap);
    }

    protected override void afterInsert() {
        AccountService.createDefaultContacts(this.newAccounts);
        AccountService.sendWelcomeEmails(this.newAccounts);
    }

    protected override void afterUpdate() {
        AccountService.syncRelatedRecords(this.newAccountMap, this.oldAccountMap);
    }

    protected override Integer getMaxLoopCount() {
        return 3; // Custom max loop count for Account triggers
    }
}

// Trigger implementation
trigger AccountTrigger on Account (before insert, before update, after insert, after update) {
    new AccountTriggerHandler().run();
}
```

Q16: Create a batch job with email notifications and error handling.
Answer:
```apex
```

```
public class AccountUpdateBatch implements Database.Batchable<SObject>,
Database.Stateful {

    private String query;
    private String updateField;
    private Object updateValue;
    private List<String> errorMessages;
    private Integer totalProcessed;
    private Integer totalErrors;

    public AccountUpdateBatch(String updateField, Object updateValue) {
        this.updateField = updateField;
        this.updateValue = updateValue;
        this.errorMessages = new List<String>();
        this.totalProcessed = 0;
        this.totalErrors = 0;

        // Build dynamic query
        this.query = 'SELECT Id, Name, ' + updateField + ' FROM Account WHERE ' + updateField
+ ' = null';
    }

    public Database.QueryLocator start(Database.BatchableContext context) {
        System.debug('Starting AccountUpdateBatch with query: ' + this.query);
        return Database.getQueryLocator(this.query);
    }

    public void execute(Database.BatchableContext context, List<Account> accounts) {
        List<Account> accountsToUpdate = new List<Account>();

        for (Account acc : accounts) {
            acc.put(this.updateField, this.updateValue);
            accountsToUpdate.add(acc);
        }

        // Perform DML with partial success allowed
        Database.SaveResult[] results = Database.update(accountsToUpdate, false);

        // Process results
        for (Integer i = 0; i < results.size(); i++) {
            Database.SaveResult result = results[i];
            Account acc = accountsToUpdate[i];

            if (result.isSuccess()) {
```

```
            this.totalProcessed++;
        } else {
            this.totalErrors++;
            String errorMsg = 'Failed to update Account ' + acc.Name + ' (ID: ' + acc.Id + '): ';

            for (Database.Error error : result.getErrors()) {
                errorMsg += error.getMessage() + '; ';
            }

            this.errorMessages.add(errorMsg);
            System.debug('Error updating account: ' + errorMsg);
        }
    }
}

public void finish(Database.BatchableContext context) {
    System.debug('AccountUpdateBatch completed. Processed: ' + this.totalProcessed + ',
Errors: ' + this.totalErrors);

    // Send completion email
    sendCompletionEmail(context.getJobId());

    // Log batch execution
    logBatchExecution(context);
}

private void sendCompletionEmail(Id jobId) {
    AsyncApexJob job = [
        SELECT Id, Status, NumberOfErrors, JobItemsProcessed, TotalJobItems, CreatedDate,
CompletedDate
        FROM AsyncApexJob
        WHERE Id = :jobId
    ];

    Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();

    // Set email properties
    email.setSubject('Account Update Batch Job Completion - ' + job.Status);
    email.setToAddresses(new List<String>{'admin@company.com'});

    // Build email body
    String emailBody = buildEmailBody(job);
    email.setHtmlBody(emailBody);
```

```
        // Send email
        try {
            Messaging.sendEmail(new List<Messaging.SingleEmailMessage>{email});
        } catch (Exception e) {
            System.debug('Failed to send completion email: ' + e.getMessage());
        }
    }

    private String buildEmailBody(AsyncApexJob job) {
        String body = '<html><body>';
        body += '<h2>Account Update Batch Job Results</h2>';
        body += '<table border="1" style="border-collapse: collapse;">';
        body += '<tr><td><strong>Job ID:</strong></td><td>' + job.Id + '</td></tr>';
        body += '<tr><td><strong>Status:</strong></td><td>' + job.Status + '</td></tr>';
        body += '<tr><td><strong>Started:</strong></td><td>' + job.CreatedDate + '</td></tr>';
        body += '<tr><td><strong>Completed:</strong></td><td>' + job.CompletedDate +
'</td></tr>';
        body += '<tr><td><strong>Total Items:</strong></td><td>' + job.TotalJobItems +
'</td></tr>';
        body += '<tr><td><strong>Processed:</strong></td><td>' + job.JobItemsProcessed +
'</td></tr>';
        body += '<tr><td><strong>Errors:</strong></td><td>' + job.NumberOfErrors + '</td></tr>';
        body += '<tr><td><strong>Update Field:</strong></td><td>' + this.updateField +
'</td></tr>';
        body += '<tr><td><strong>Update Value:</strong></td><td>' + this.updateValue +
'</td></tr>';
        body += '</table>';

        if (!this.errorMessages.isEmpty()) {
            body += '<h3>Error Details:</h3><ul>';
            for (String error : this.errorMessages) {
                body += '<li>' + error + '</li>';
            }
            body += '</ul>';
        }

        body += '</body></html>';
        return body;
    }

    private void logBatchExecution(Database.BatchableContext context) {
        // Create custom log record (assuming you have a Batch_Log__c custom object)
        try {
            Batch_Log__c logRecord = new Batch_Log__c();
```

```
            logRecord.Job_Id__c = String.valueOf(context.getJobId());
            logRecord.Batch_Class__c = 'AccountUpdateBatch';
            logRecord.Total_Processed__c = this.totalProcessed;
            logRecord.Total_Errors__c = this.totalErrors;
            logRecord.Execution_Date__c = System.now();
            logRecord.Status__c = this.totalErrors > 0 ? 'Completed with Errors' : 'Completed
Successfully';

            if (!this.errorMessages.isEmpty()) {
                String errorSummary = String.join(this.errorMessages, '\n');
                logRecord.Error_Details__c = errorSummary.length() > 32000 ?
                                    errorSummary.substring(0, 32000) : errorSummary;
            }

            insert logRecord;

        } catch (Exception e) {
            System.debug('Failed to create batch log record: ' + e.getMessage());
        }
    }
}

// Scheduler class for the batch job
public class AccountUpdateScheduler implements Schedulable {

    private String updateField;
    private Object updateValue;
    private Integer batchSize;

    public AccountUpdateScheduler(String updateField, Object updateValue, Integer batchSize) {
        this.updateField = updateField;
        this.updateValue = updateValue;
        this.batchSize = batchSize != null ? batchSize : 200;
    }

    public void execute(SchedulableContext context) {
        AccountUpdateBatch batch = new AccountUpdateBatch(this.updateField,
this.updateValue);
        Database.executeBatch(batch, this.batchSize);
    }
}

// Usage examples:
// Execute immediately:
```

```
// AccountUpdateBatch batch = new AccountUpdateBatch('Industry', 'Technology');
// Database.executeBatch(batch, 100);

// Schedule for later:
// String cronExp = '0 0 2 * * ?'; // Every day at 2 AM
// AccountUpdateScheduler scheduler = new AccountUpdateScheduler('Industry', 'Technology',
100);
// System.schedule('Account Update Job', cronExp, scheduler);
```

System Design Questions

Q17: Design a solution for handling high-volume data integration with external systems.
Answer:

Architecture Components:

1. **Data Ingestion Layer**
   - Platform Events for real-time data streaming
   - Bulk API 2.0 for large data loads
   - REST APIs for transactional updates

2. **Processing Layer**
   - Queueable jobs for asynchronous processing
   - Batch jobs for bulk operations
   - Future methods for callouts

3. **Storage Strategy**
   - External Objects for real-time external data access
   - Big Objects for massive data storage
   - Custom Objects for transactional data

```apex
// Central orchestration class
public class DataIntegrationOrchestrator {

    public enum ProcessType { REAL_TIME, BATCH, STREAMING }

    public static void processIncomingData(String dataType, List<Object> dataRecords,
ProcessType processType) {

        DataProcessor processor = DataProcessorFactory.getProcessor(dataType);
```

```
    switch on processType {
        when REAL_TIME {
            processor.processRealTime(dataRecords);
        }
        when BATCH {
            processor.processBatch(dataRecords);
        }
        when STREAMING {
            processor.processStreaming(dataRecords);
        }
    }
}

// Factory pattern for different data processors
public class DataProcessorFactory {

    public static DataProcessor getProcessor(String dataType) {
        switch on dataType.toUpperCase() {
            when 'CUSTOMER' {
                return new CustomerDataProcessor();
            }
            when 'ORDER' {
                return new OrderDataProcessor();
            }
            when 'PRODUCT' {
                return new ProductDataProcessor();
            }
            when else {
                throw new ProcessorException('Unknown data type: ' + dataType);
            }
        }
    }
}

// Abstract processor interface
public abstract class DataProcessor {

    public abstract void processRealTime(List<Object> records);
    public abstract void processBatch(List<Object> records);
    public abstract void processStreaming(List<Object> records);

    protected void validateData(List<Object> records) {
        // Common validation logic
```

```
        }

        protected void logProcessingResults(String processType, Integer recordCount,
List<String> errors) {
            Integration_Log__c log = new Integration_Log__c();
            log.Process_Type__c = processType;
            log.Record_Count__c = recordCount;
            log.Error_Count__c = errors.size();
            log.Processing_Time__c = System.now();

            if (!errors.isEmpty()) {
                log.Error_Details__c = String.join(errors, '\n');
            }

            insert log;
        }
    }
}
```

Key Design Principles:
- **Scalability**: Use asynchronous processing and bulk operations
- **Reliability**: Implement retry mechanisms and error handling
- **Monitoring**: Log all operations and send alerts for failures
- **Flexibility**: Use factory patterns for different data types
- **Performance**: Optimize SOQL queries and use platform cache

Q18: How would you implement a multi-tenant data security model?
Answer:

Security Implementation Strategy:

```apex
public with sharing class SecurityManager {

    // Tenant isolation using Record Types and Sharing Rules
    public static void enforceTenantSecurity(List<SObject> records, String operation) {

        String currentUserTenant = getCurrentUserTenant();

        for (SObject record : records) {
            // Enforce tenant boundaries
            if (record.get('Tenant__c') != currentUserTenant && !isSystemAdmin()) {
                throw new SecurityException('Access denied: Cross-tenant access not allowed');
```

```
            }
        }

        // Apply field-level security
        enforceFieldSecurity(records, operation);
    }

    private static String getCurrentUserTenant() {
        User currentUser = [SELECT Tenant__c FROM User WHERE Id = :UserInfo.getUserId()];
        return currentUser.Tenant__c;
    }

    private static Boolean isSystemAdmin() {
        return [SELECT Id FROM Profile WHERE Id = :UserInfo.getProfileId() AND Name =
'System Administrator'].size() > 0;
    }

    private static void enforceFieldSecurity(List<SObject> records, String operation) {

        if (records.isEmpty()) return;

        String objectName = String.valueOf(records[0].getSObjectType());
        Schema.DescribeSObjectResult objectDescribe =
Schema.getGlobalDescribe().get(objectName).getDescribe();

        Map<String, Schema.SObjectField> fieldMap = objectDescribe.fields.getMap();

        for (SObject record : records) {
            Map<String, Object> populatedFields = record.getPopulatedFieldsAsMap();

            for (String fieldName : populatedFields.keySet()) {
                Schema.SObjectField field = fieldMap.get(fieldName);

                if (field != null) {
                    Schema.DescribeFieldResult fieldDescribe = field.getDescribe();

                    switch on operation.toUpperCase() {
                        when 'INSERT', 'UPDATE' {
                            if (!fieldDescribe.isCreateable() || !fieldDescribe.isUpdateable()) {
                                throw new SecurityException('Insufficient permissions to modify field: ' +
fieldName);
                            }
                        }
                        when 'READ' {
```

```
                    if (!fieldDescribe.isAccessible()) {
                        record.put(fieldName, null); // Remove sensitive data
                    }
                }
            }
        }
    }
}

// Custom sharing implementation
public class TenantSharingService {

    public static void shareRecordsWithTenant(List<Id> recordIds, String tenantId, String
accessLevel) {

        List<SObject> sharesToInsert = new List<SObject>();

        // Get all users in the tenant
        List<User> tenantUsers = [SELECT Id FROM User WHERE Tenant__c = :tenantId AND
IsActive = true];

        for (Id recordId : recordIds) {
            String objectName = recordId.getSObjectType().getDescribe().getName();
            String shareObjectName = objectName.replace('__c', '__Share');

            for (User user : tenantUsers) {
                SObject shareRecord =
Schema.getGlobalDescribe().get(shareObjectName).newSObject();
                shareRecord.put('ParentId', recordId);
                shareRecord.put('UserOrGroupId', user.Id);
                shareRecord.put('AccessLevel', accessLevel);
                shareRecord.put('RowCause', 'Manual');

                sharesToInsert.add(shareRecord);
            }
        }

        if (!sharesToInsert.isEmpty()) {
            insert sharesToInsert;
        }
    }
}
```

```

---

## Advanced Topics & Recent Updates

### Q19: What are the latest Salesforce features you should know about?
**Answer:**

**Flow Enhancements (2023-2024):**
- **Reactive Components**: Auto-refresh screen components
- **Flow Orchestrator**: Manage complex multi-flow processes
- **Enhanced Debugging**: Better error handling and logging

**Lightning Web Components:**
- **Lightning Message Service**: Component communication across DOM
- **Wire Adapters**: Enhanced data fetching capabilities
- **Custom Renderers**: Advanced rendering control

**Apex Improvements:**
- **Enhanced Switch Statements**: Pattern matching capabilities
- **Improved Exception Handling**: Better error context
- **Async Apex Enhancements**: Better monitoring and control

```apex
// Modern switch statement example
public String categorizeAccount(Account acc) {
    return switch on acc.Industry {
        when 'Technology', 'Software' => 'Tech Sector';
        when 'Healthcare', 'Pharmaceuticals' => 'Health Sector';
        when 'Banking', 'Insurance' => 'Financial Sector';
        when null => 'Uncategorized';
        when else => 'Other Sector';
    };
}

// Enhanced exception handling
public class ModernExceptionHandling {

    public static void processRecords(List<Account> accounts) {
        try {
            performBusinessLogic(accounts);
        } catch (DmlException e) {
            handleDmlException(e);
```
```

```
        } catch (Exception e) {
            handleGenericException(e, 'processRecords');
        }
    }

    private static void handleDmlException(DmlException e) {
        for (Integer i = 0; i < e.getNumDml(); i++) {
            System.debug('DML Error on record ' + i + ': ' + e.getDmlMessage(i));
            // Log specific error details
            logError('DML_ERROR', e.getDmlMessage(i), e.getDmlId(i));
        }
    }

    private static void handleGenericException(Exception e, String methodName) {
        System.debug('Exception in ' + methodName + ': ' + e.getMessage());
        System.debug('Stack trace: ' + e.getStackTraceString());

        // Enhanced error logging with context
        logError('GENERIC_ERROR', e.getMessage(), methodName);
    }

    private static void logError(String errorType, String message, Object context) {
        // Implementation for comprehensive error logging
    }
}
```

Q20: Explain Einstein Platform Services integration.
Answer:

Einstein Platform Services Integration:

```apex
public class EinsteinVisionService {

    private static final String VISION_URL = 'https://api.einstein.ai/v2/vision';
    private static final String PREDICTION_URL = '/predict';

    public class PredictionResult {
        public String label;
        public Decimal probability;
        public String modelId;
    }
```

```
    public static List<PredictionResult> classifyImage(Blob imageBlob, String modelId) {
        try {
            HttpRequest request = buildVisionRequest(imageBlob, modelId);
            HttpResponse response = new Http().send(request);

            if (response.getStatusCode() == 200) {
                return parseVisionResponse(response.getBody());
            } else {
                throw new EinsteinException('Vision API Error: ' + response.getStatusCode() + ' - ' +
response.getBody());
            }

        } catch (Exception e) {
            System.debug('Einstein Vision Error: ' + e.getMessage());
            throw new EinsteinException('Failed to classify image: ' + e.getMessage());
        }
    }

    private static HttpRequest buildVisionRequest(Blob imageBlob, String modelId) {
        HttpRequest request = new HttpRequest();
        request.setEndpoint(VISION_URL + PREDICTION_URL);
        request.setMethod('POST');
        request.setTimeout(120000);

        // Get Einstein Platform Services access token
        String accessToken = getEinsteinAccessToken();
        request.setHeader('Authorization', 'Bearer ' + accessToken);
        request.setHeader('Cache-Control', 'no-cache');

        // Build multipart form data
        String boundary = '----WebKitFormBoundary7MA4YWxkTrZu0gW';
        request.setHeader('Content-Type', 'multipart/form-data; boundary=' + boundary);

        String body = buildMultipartBody(imageBlob, modelId, boundary);
        request.setBody(body);

        return request;
    }

    private static String buildMultipartBody(Blob imageBlob, String modelId, String boundary) {
        String body = '';

        // Add sampleBase64Content parameter
        body += '--' + boundary + '\r\n';
```

```
        body += 'Content-Disposition: form-data; name="sampleBase64Content"\r\n\r\n';
        body += EncodingUtil.base64Encode(imageBlob) + '\r\n';

        // Add modelId parameter
        body += '--' + boundary + '\r\n';
        body += 'Content-Disposition: form-data; name="modelId"\r\n\r\n';
        body += modelId + '\r\n';

        // Add numResults parameter
        body += '--' + boundary + '\r\n';
        body += 'Content-Disposition: form-data; name="numResults"\r\n\r\n';
        body += '5\r\n';

        body += '--' + boundary + '--\r\n';

        return body;
    }

    private static List<PredictionResult> parseVisionResponse(String responseBody) {
        List<PredictionResult> results = new List<PredictionResult>();

        Map<String, Object> responseMap = (Map<String, Object>)
JSON.deserializeUntyped(responseBody);
        List<Object> probabilities = (List<Object>) responseMap.get('probabilities');

        for (Object prob : probabilities) {
            Map<String, Object> predictionMap = (Map<String, Object>) prob;

            PredictionResult result = new PredictionResult();
            result.label = (String) predictionMap.get('label');
            result.probability = (Decimal) predictionMap.get('probability');
            result.modelId = (String) responseMap.get('modelId');

            results.add(result);
        }

        return results;
    }

    private static String getEinsteinAccessToken() {
        // This would typically retrieve from Named Credential or Custom Setting
        // For demo purposes, showing the structure

        HttpRequest request = new HttpRequest();
```

```
        request.setEndpoint('https://api.einstein.ai/v2/oauth2/token');
        request.setMethod('POST');
        request.setHeader('Content-Type', 'application/x-www-form-urlencoded');
        request.setHeader('Accept', 'application/json');

        String body = 'grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer';
        body += '&assertion=' + generateJWT();
        request.setBody(body);

        HttpResponse response = new Http().send(request);

        if (response.getStatusCode() == 200) {
            Map<String, Object> tokenResponse = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            return (String) tokenResponse.get('access_token');
        } else {
            throw new EinsteinException('Failed to get access token: ' + response.getBody());
        }
    }

    private static String generateJWT() {
        // JWT generation for Einstein Platform Services
        // This is a simplified version - in practice, use proper JWT libraries
        String header = '{"alg":"RS256","typ":"JWT"}';
        String claimSet =
'{"iss":"your-email@example.com","sub":"your-email@example.com","aud":"https://api.einstein.a
i/v2/oauth2/token","exp":' + (System.currentTimeMillis()/1000 + 3600) + '}';

        String encodedHeader = EncodingUtil.base64Encode(Blob.valueOf(header));
        String encodedClaimSet = EncodingUtil.base64Encode(Blob.valueOf(claimSet));

        String token = encodedHeader + '.' + encodedClaimSet;

        // In real implementation, sign with your private key
        // String signature = signWithPrivateKey(token);
        // return token + '.' + signature;

        return token + '.signature_placeholder';
    }

    public class EinsteinException extends Exception {}
}
```

Einstein Platform Services

Advanced LWC Patterns with Einstein Integration

advancedDataTable.html:
```html
<template>
    <lightning-card title="Advanced Data Management" icon-name="custom:custom63">
        <div class="slds-m-around_medium">

            <!-- Filter Controls -->
            <div class="slds-grid slds-gutters slds-m-bottom_medium">
                <div class="slds-col slds-size_1-of-3">
                    <lightning-combobox
                        name="industry"
                        label="Industry Filter"
                        value={selectedIndustry}
                        placeholder="Select Industry"
                        options={industryOptions}
                        onchange={handleIndustryChange}>
                    </lightning-combobox>
                </div>
                <div class="slds-col slds-size_1-of-3">
                    <lightning-input
                        type="number"
                        label="Min Revenue"
                        value={minRevenue}
                        onchange={handleMinRevenueChange}
                        formatter="currency">
                    </lightning-input>
                </div>
                <div class="slds-col slds-size_1-of-3">
                    <div class="slds-m-top_large">
                        <lightning-button
                            variant="brand"
                            label="Refresh Data"
                            onclick={refreshData}
                            disabled={isLoading}>
                        </lightning-button>
                    </div>
                </div>
            </div>
```

```
        <!-- Error Display -->
        <template if:true={error}>
            <div class="slds-notify slds-notify_alert slds-theme_error slds-m-bottom_medium">
                <span class="slds-assistive-text">Error</span>
                <lightning-icon icon-name="utility:error" size="small"
class="slds-m-right_small"></lightning-icon>
                {error.body.message}
                <lightning-button-icon
                    icon-name="utility:close"
                    size="small"
                    variant="bare-inverse"
                    onclick={clearError}
                    class="slds-float_right">
                </lightning-button-icon>
            </div>
        </template>

        <!-- Loading State -->
        <template if:true={isLoading}>
            <div class="slds-text-align_center slds-p-vertical_large">
                <lightning-spinner size="large"></lightning-spinner>
                <p class="slds-m-top_small">Loading data...</p>
            </div>
        </template>

        <!-- Data Table -->
        <template if:false={isLoading}>
            <lightning-datatable
                key-field="Id"
                data={displayedAccounts}
                columns={columns}
                onrowaction={handleRowAction}
                onsave={handleSave}
                draft-values={draftValues}
                hide-checkbox-column="false"
                onrowselection={handleRowSelection}
                show-row-number-column="true">
            </lightning-datatable>

            <!-- Pagination -->
            <div class="slds-m-top_medium slds-grid slds-grid_align-spread">
                <div class="slds-col">
                    <p>Showing {startRecord} to {endRecord} of {totalRecords} records</p>
```

```
                </div>
                <div class="slds-col">
                    <lightning-button-group>
                        <lightning-button
                            label="Previous"
                            onclick={previousPage}
                            disabled={isFirstPage}>
                        </lightning-button>
                        <lightning-button
                            label="Next"
                            onclick={nextPage}
                            disabled={isLastPage}>
                        </lightning-button>
                    </lightning-button-group>
                </div>
            </div>
        </template>

    </div>
  </lightning-card>
</template>
```

advancedDataTable.js:
```javascript
import { LightningElement, track, wire, api } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import { refreshApex } from '@salesforce/apex';
import { updateRecord } from 'lightning/uiRecordApi';
import { getPicklistValues } from 'lightning/uiObjectInfoApi';
import { getObjectInfo } from 'lightning/uiObjectInfoApi';

import getFilteredAccounts from
'@salesforce/apex/AdvancedAccountController.getFilteredAccounts';
import ACCOUNT_OBJECT from '@salesforce/schema/Account';
import INDUSTRY_FIELD from '@salesforce/schema/Account.Industry';

const COLUMNS = [
  {
    label: 'Account Name',
    fieldName: 'Name',
    type: 'text',
    editable: true,
    sortable: true
```

```
    },
    {
        label: 'Industry',
        fieldName: 'Industry',
        type: 'picklistColumn',
        editable: true,
        typeAttributes: {
            placeholder: 'Select Industry',
            options: { fieldName: 'industryOptions' },
            value: { fieldName: 'Industry' }
        }
    },
    {
        label: 'Annual Revenue',
        fieldName: 'AnnualRevenue',
        type: 'currency',
        editable: true,
        cellAttributes: { alignment: 'right' }
    },
    {
        label: 'Phone',
        fieldName: 'Phone',
        type: 'phone',
        editable: true
    },
    {
        label: 'Website',
        fieldName: 'Website',
        type: 'url',
        editable: true
    },
    {
        type: 'action',
        typeAttributes: {
            rowActions: [
                { label: 'View', name: 'view' },
                { label: 'Edit', name: 'edit' },
                { label: 'Clone', name: 'clone' }
            ]
        }
    }
];

export default class AdvancedDataTable extends LightningElement {
```

```
// Public properties
@api recordId;
@api objectApiName = 'Account';

// Tracked properties
@track accounts = [];
@track displayedAccounts = [];
@track error;
@track isLoading = false;
@track selectedIndustry = '';
@track minRevenue = 0;
@track draftValues = [];
@track selectedRows = [];

// Pagination
@track currentPage = 1;
@track pageSize = 10;
@track totalRecords = 0;

// Configuration
columns = COLUMNS;
industryOptions = [];

// Wire results for refreshApex
wiredAccountsResult;
wiredIndustryPicklistResult;

// Wire Object Info
@wire(getObjectInfo, { objectApiName: ACCOUNT_OBJECT })
accountObjectInfo;

// Wire Industry Picklist Values
@wire(getPicklistValues, {
    recordTypeId: '$accountObjectInfo.data.defaultRecordTypeId',
    fieldApiName: INDUSTRY_FIELD
})
wiredIndustryPicklist(result) {
    this.wiredIndustryPicklistResult = result;
    if (result.data) {
        this.industryOptions = [
            { label: 'All Industries', value: '' },
            ...result.data.values.map(item => ({
                label: item.label,
```

```
                value: item.value
            }))
        ];
    } else if (result.error) {
        this.handleError(result.error);
    }
}

// Wire Account Data
@wire(getFilteredAccounts, {
    industry: '$selectedIndustry',
    minRevenue: '$minRevenue',
    limitSize: '$pageSize',
    offset: '$offset'
})
wiredAccounts(result) {
    this.wiredAccountsResult = result;
    this.isLoading = false;

    if (result.data) {
        this.accounts = result.data.accounts || [];
        this.totalRecords = result.data.totalCount || 0;
        this.updateDisplayedAccounts();
        this.error = undefined;
    } else if (result.error) {
        this.handleError(result.error);
        this.accounts = [];
        this.displayedAccounts = [];
    }
}

// Computed Properties
get offset() {
    return (this.currentPage - 1) * this.pageSize;
}

get startRecord() {
    return this.totalRecords > 0 ? this.offset + 1 : 0;
}

get endRecord() {
    return Math.min(this.offset + this.pageSize, this.totalRecords);
}
```

```
get isFirstPage() {
    return this.currentPage === 1;
}

get isLastPage() {
    return this.currentPage >= Math.ceil(this.totalRecords / this.pageSize);
}

// Event Handlers
handleIndustryChange(event) {
    this.selectedIndustry = event.detail.value;
    this.currentPage = 1;
    this.isLoading = true;
}

handleMinRevenueChange(event) {
    this.minRevenue = event.detail.value || 0;
    this.currentPage = 1;
    this.isLoading = true;
}

refreshData() {
    this.isLoading = true;
    this.currentPage = 1;
    return refreshApex(this.wiredAccountsResult);
}

handleRowAction(event) {
    const actionName = event.detail.action.name;
    const row = event.detail.row;

    switch (actionName) {
        case 'view':
            this.navigateToRecord(row.Id, 'view');
            break;
        case 'edit':
            this.navigateToRecord(row.Id, 'edit');
            break;
        case 'clone':
            this.cloneRecord(row);
            break;
    }
}
```

```
async handleSave(event) {
    const updatedFields = event.detail.draftValues;

    try {
        this.isLoading = true;

        // Create promises for each record update
        const updatePromises = updatedFields.map(draft => {
            const fields = { ...draft };
            return updateRecord({ fields });
        });

        // Wait for all updates to complete
        await Promise.all(updatePromises);

        // Show success message
        this.showToast('Success', 'Records updated successfully', 'success');

        // Clear draft values
        this.draftValues = [];

        // Refresh data
        await this.refreshData();

    } catch (error) {
        this.handleError(error);
    } finally {
        this.isLoading = false;
    }
}

handleRowSelection(event) {
    this.selectedRows = event.detail.selectedRows;
}

// Pagination Handlers
previousPage() {
    if (!this.isFirstPage) {
        this.currentPage--;
        this.isLoading = true;
    }
}

nextPage() {
```

```
    if (!this.isLastPage) {
        this.currentPage++;
        this.isLoading = true;
    }
}

// Utility Methods
updateDisplayedAccounts() {
    // Add industry options to each account for picklist editing
    this.displayedAccounts = this.accounts.map(account => ({
        ...account,
        industryOptions: this.industryOptions.filter(option => option.value !== '')
    }));
}

navigateToRecord(recordId, actionName) {
    this[NavigationMixin.Navigate]({
        type: 'standard__recordPage',
        attributes: {
            recordId: recordId,
            actionName: actionName
        }
    });
}

cloneRecord(sourceRecord) {
    // Create a clone with specific fields
    const cloneFields = {
        Name: sourceRecord.Name + ' (Clone)',
        Industry: sourceRecord.Industry,
        Phone: sourceRecord.Phone,
        Website: sourceRecord.Website
    };

    // Navigate to new record page with default values
    this[NavigationMixin.Navigate]({
        type: 'standard__objectPage',
        attributes: {
            objectApiName: 'Account',
            actionName: 'new'
        },
        state: {
            defaultFieldValues: encodeDefaultFieldValues(cloneFields)
        }
```

```
        });
    }

    handleError(error) {
        this.error = error;
        console.error('Component Error:', error);

        let message = 'Unknown error occurred';
        if (error.body && error.body.message) {
            message = error.body.message;
        } else if (error.message) {
            message = error.message;
        }

        this.showToast('Error', message, 'error');
    }

    clearError() {
        this.error = undefined;
    }

    showToast(title, message, variant) {
        const event = new ShowToastEvent({
            title: title,
            message: message,
            variant: variant,
            mode: variant === 'error' ? 'sticky' : 'dismissable'
        });
        this.dispatchEvent(event);
    }
}

// Helper function for navigation
function encodeDefaultFieldValues(fields) {
    return Object.keys(fields)
        .map(key => `${key}=${encodeURIComponent(fields[key])}`)
        .join(',');
}
```

Interview Success Tips

Q22: Common Interview Mistakes to Avoid
Answer:

Technical Mistakes:
1. **Not considering bulkification** - Always write bulk-safe code
2. **Ignoring governor limits** - Understand and optimize for limits
3. **Poor exception handling** - Implement comprehensive error handling
4. **Inefficient SOQL** - Use selective queries with proper indexing
5. **Not using design patterns** - Implement trigger handlers, factory patterns

Communication Mistakes:
1. **Not asking clarifying questions** - Understand requirements fully
2. **Jumping into code immediately** - Discuss approach first
3. **Not explaining your thought process** - Walk through your reasoning
4. **Ignoring edge cases** - Consider error scenarios and data validation

Best Practices to Demonstrate:
- Security awareness (sharing rules, field-level security)
- Performance optimization techniques
- Testable code with proper separation of concerns
- Modern Salesforce features and best practices
- Integration patterns and error handling

Q23: Sample System Architecture Questions
Answer:

Question: "Design a solution for a real estate company that needs to integrate with multiple MLS systems, handle property data synchronization, and provide a mobile app for agents."

Solution Approach:
1. **Data Architecture**: Custom objects for Properties, Listings, Agents
2. **Integration Layer**: REST APIs, Platform Events for real-time updates
3. **Processing**: Batch jobs for bulk sync, Queueable for real-time
4. **Mobile**: Lightning Platform with offline capabilities
5. **Security**: Territory management, sharing rules, field-level security

Key Discussion Points:
- **Scalability**: How to handle millions of property records
- **Performance**: Caching strategies and query optimization
- **Integration**: Error handling and retry mechanisms
- **Security**: Data isolation and access controls
- **Monitoring**: Logging and alerting for system health

Final Preparation Checklist

Technical Readiness
- [ ] Review all governor limits and optimization techniques
- [ ] Practice writing trigger frameworks and bulk-safe code
- [ ] Understand modern LWC patterns and wire adapters
- [ ] Know integration best practices and error handling
- [ ] Be familiar with recent Salesforce updates and features

Interview Strategy
- [ ] Prepare STAR method examples for behavioral questions
- [ ] Practice explaining complex technical concepts simply
- [ ] Have questions ready about the company's Salesforce implementation
- [ ] Prepare examples of challenging problems you've solved
- [ ] Review the job description and align your experience

Code Examples to Master
- [ ] Trigger handler with recursion prevention
- [ ] Batch job with error handling and notifications
- [ ] LWC component with advanced patterns
- [ ] REST API with proper error handling
- [ ] Platform Event implementation
- [ ] Dynamic SOQL utility class

Conclusion

This comprehensive interview kit covers the essential topics for a senior