LeadSuccess API for Salesforce

The **LeadSuccess API for Salesforce** provides access to server-side data structures and procedures via a RESTful API to enable 3rd party developers building customized Export interface from the LeadSuccess System to Salesforce CRM system with standard web techniques and is covering the data created in the original LeadSuccess App in objects compatible to Salesforce objects.

You can find more information on http://www.convey.de/ to learn about further components of the LeadSuccess system.

LeadSucces App

Introduction

The LeadSuccess App allows you to collect data and information about the visitors in an easy, fast and reliable way. It allows you to get the visitor´s data in various ways; in fact, it is possible to scan barcodes, QR-Codes, business card or enter information manually. In addition to this, it is also possible to fill out a customizable questionnaire, take notes and add pictures and sketches to every contact you collect.

Exhibitor Portal

Everything can be managed through the "Exhibitor Portal" which allows you to manage your App users, create your own event-specific questionnaire, check and edit leads and furthermore export your collected leads to different Excel formats.

You will need a valid LeadSuccess admin account to configure the event and app users. App user credentials configured in the "Exhibitor Portal" can be used either with the original LeadSuccess App or for requests with basic authentication to use the API.

You will find additional information about administration of LeadSuccess in the document "User Guide LeadSuccess Mobile – Online Portal"

LeadSuccess app structure

The LeadSuccess App user interface offers the following functional areas to the user:

- **Start Page:**
  Event name, user name, number of contacts, present state, new message flag
- **Capture Barcode**
  Create a new contact based on the visitor's badge barcode
- **Capture Business card**
  Create a new contact based on the visitor's business card photo and save the photo linked to the contact
- **Edit contact manually**
  Create a new contact or edit existing address fields of a selected visitor contact
- **Edit questionnaire**
  Offers a form to answer the questions of the questionnaire based on the questionnaire configuration (multiselect, single select, combo, rating, text notes, date picker, optional questions, mandatory questions). Each questionnaire is linked to a selected contact
- **Create notes**
  Create and edit sketches in SVG format, capture photos, capture audio messages. All notes are linked to a selected contact
- **Edit user state**
  Edit personal user data, toggle the present state, receive and send user messages and

capture a user photo. The data is linked to the employee data of the app user and can be administrated in the Exhibitor Portal

Access protocol

To access this interface is used a subset of ODATA protocol. See ODATA org. ODATA server is based on SAP SQL Anywhere 17 OData implementation. Therefore currently only ODATA version 2.0 syntax can be used. See http://www.odata.org/documentation/odata-version-2-0

LeadSuccess API users will get the **<server-name>** and **<api-name>** to be used for your API-requests together with their login information from convey.

To access this API, HTTPS basic authentication is used. It means user and password has to be transmitted via HTTPS with each request. Since HTTP(S) is a stateless protocol, each request is computed in individual transactions. GET and POST requests return table data as result of the request in the specified format, e.g. JSON. If you use a XMLHTTPRequest alike JavaScript API, use JSON.parse(response.responseText) to transform the result in a JSON object that for automatic data-binding to UI elements within your preferred framework. Some requests will result in URLs to get further data. Due to technical limitations the provided address of these links is a local address and not your API address. Please replace that local address in these URLs to:

```
https://<server-name>/<api-name>/...
```

All parameter data must be transferred in JSON format to the ODATA server. For results the JSON format can be selected with the "$format=json" request option or with the HTTP header “Accept: application/json”, otherwise the server returns the results in XML format. All text data is UTF-8 encoded, except described otherwise for specific data fields. All date-time data is in UTC time zone.

Pay attention to the fact, that the OData request syntax is **case sensitive**!

A user can only keep one transaction active at a time. If several parallel transactions are to be supported, different users must be used for this!

The following request types are supported

Select data

Use requests of type: **GET** to select data from LeadSuccess relations. You should distinguish at least these different types of GET requests:

Select a list of rows

You can select lists of data from API objects with requests like:

GET https://<server-name>/<api-name>/LSA_<table-name>?$format=json

You can use the $filter=(<filter-list>) and $orderby=(<orderby-list>) options to specify restrictions and list order to the request.

Requests of this kind will return an array of rows, that could be handled like this:

```
function xhrSuccess(response) {
```

```
    var obj = JSON.parse(response.responseText);
    var results = obj && obj.d && obj.d.results;
    handleResults(results);
  }
```

The result size maximum to be fetched by one request is limited to 100 rows. To fetch the next row-set, you can use the parameter $skiptoken(<primary-key-value>) in a following GET request to the same relation. To simplify that, you can use the following member in result JSON structure as described above:

```
obj.d.__next
```

Remember to replace the server path in the URL as described above, e.g.:

```
var getNextUrl = function (json) {
    var url = "";
    if (json && json.d) {
        var next = json.d.__next;
        if (next && typeof next === "string") {
            var viewNamePos = next.lastIndexOf("/");
            if (viewNamePos >= 0) {
                url = "https://<server-name>/<api_name>" +
                      next.substr(viewNamePos);
            }
        }
    }
    return url;
}
```

Fetch next list of rows

Use a nextURL given from earlier select request to fetch the next row-set. The returned result is similar to the one of the first select request, including results array and __next member for following row-set, if end of data isn't reached yet.

Select single row

You can use the primary key of each table to select a single row of data with the following request:

```
GET https://<server-name>/<api-name>/LSA_<table-name>(<primary-key-value>)?$format=json
```

Primary keys are always attributes of an integer type.

Requests of this kind will return one row, that could be handled like this:

```
function xhrSuccess(response) {
  var obj = JSON.parse(response.responseText);
  var result = obj && obj.d;
  handleResults(result);
}
```

Procedure call

Use requests of type: **GET** to call procedures. Parameters need to be placed URL-encoded within the URL.

```
var options = {
  type: "GET",
  user: <user-name>,
  password: <password>,
  url: "https://<server-name>/<api-name>/<procedure-name>?<parameters>& $format=json",
};
```

You can build the parameters list like this:

```
<paramaters> = <param1>=<value1>&<param2>=<value2>…;
```

CORS requirements

To support cross-site access-control requests, you should specify the `withCredentials` member of the `XMLHttpRequest` to `true`. You may need to add an option e.g. like this to add the value all your xhr requests, if supported by the `XMLHttpRequest` object:

```
var options = {
  //
  // other options depending from action, see above
  //
  //
  customRequestInitializer: function(req) {
    if (typeof req.withCredentials !== "undefined") {
      req.withCredentials = true;
    }
  }
};
```

Authorization header

To support server-side authorization propagation by some browser clients, like Google Chrome, you should add an “Authorization” header:

```
var options = {
  //
  // other options depending from action, see above
  //
  //
  headers: {
    "Authorization": "Basic " + btoa(<user> + ":" + <password>)
  }
};
```

Resources

Data categories

Based on the given function areas, the frontend needs to handle different data structures to offer the functionality of the LeadSuccess App. This data can be divided in:

- **Static data**
  to be loaded only once after app installation
- **Event data:**
  to be loaded at least once after app installation and updated after administrative changes in the LeadSuccess Exhibitor Portal
- **Application runtime data**
  to be loaded at least once after app installation and updated after interactive changes in the app or in the LeadSuccess Exhibitor Portal
- **Visitor data**
  to be created interactively in the app and to be retrieved via search and list panes

All data is organized in an object structure similar to Salesforce objects. Object contain Salesforce system fields like Id, CreatedDate and LastModifiedDate and object-specific fields. Each Object has a unique Id, that can be referenced as OwnerId field in another object. The LeadSuccess API for Salesforce API offers views called LS_<object-name> or LS_<Procedure-name> to access the data.

Access rights

The LeadSuccess API for Salesforce offers access rights on object-level to the provided views in order to their usage. For static application and runtime data only GET requests are offered. Accessing an object with an unsupported request option will return an exception.

The LeadSuccess API for Salesforce offers access rights on row-level to the provided views. You can only access rows of data in the user context of the currently logged-in user.

Data schema

You can retrieve the full schema data with the following request:

```
GET https://<server-name>/<api-name>/$metadata
```

Static master data

LS_Country

Select the entries of this view to receive s list of CountryCodes and Countries that can be referenced in LeadSuccess API for Salesforce.

Name	Type	Description
LGNTINITLandViewId	Int32	Primary key value
Id	Single-Byte-String(2)	Two letter ISO code. Only single-byte characters are allowed in this field
Country	String(255)	Display text for the country to select

You can use several reference columns to identify country selection for data export.

Application runtime data

Select this data to show information relevant for application runtime.

LS_Event

Select the entries of this view to show information about the events referenced by collected data.

Name	Type	Description
VeranstaltungViewId	Int32	Primary key value.
Id	UUID-String(36)	Universal Unique Identifier. Use this value as reference in objects referencing the **Event** object.
CreatedDate	DateTime	Creation timestamp of the event object
LastModifiedDate	DateTime	Timestamp of last modification of the event object
Subject	String(127)	Display text for the event shown in the app. The value can be edited in LeadSuccess portal
StartDate	Date	Start date of event
EndDate	Date	End date of event
Type	String(255)	Name of event organizer, defaults to value “LeadSuccess”
EventSubtype	String(1000)	Title of the event, usually the name of a trade show
Description	String(255)	A description of the event. The value can be edited in the LeadSuccess portal

LS_User

Select the entries of this view to show information about the users creating or modifying the collected data.

Name	Type	Description
MitarbeiterViewId	Int32	Primary key value
Id	Single-Byte-String(63)	Login name of the user. Only single-byte characters are allowed in this field
FirstName	String(64)	First name of the user. Can be edited in LeadSuccess portal and app
LastName	String(255)	Last name of the user. Can be edited in LeadSuccess portal and app
Email	String(500)	Email address of the user. Can be edited in the LeadSuccess app
Phone	String(64)	Phone number of the user. Can be edited in the LeadSuccess app
MobilePhone	String(64)	Cellphone number of the user. Can be edited in the LeadSuccess app
Street	String(127)	Street address of the user. Can be edited in the LeadSuccess app
PostalCode	String(12)	Postal code / ZIP address of the user. Can be edited in the LeadSuccess app
City	String(127)	City address of the user. Can be edited in the LeadSuccess app
Country	String(255)	Country address of the user. Can be selected in the LeadSuccess app
CountryCode	String(2)	2-character ISO-code of the country address of the user.
Currentstatus	String(32)	Role of the user in LeadSuccess system, e.g. booth staff member or administrator
EventID	UUID-String(36)	Universal Unique Identifier. References the **Event** where the user is currently related to and is able to collect data for in the LeadSuccess app.

Visitor data

LS_Lead

Select the entries of this view to retrieve visitor contact data collected or edited on LeadSuccess App Portal, Kiosk or Service devices or via LeadSuccess API.

Name	Type	Description
KontaktViewId	Int32	Primary key value
Id	UUID-String(36)	Universal Unique Identifier. Use this value as reference in objects referencing the **Lead** object.
CreatedDate	DateTime	Creation timestamp of the lead object.
LastModifiedDate	DateTime	Timestamp of last modification of the lead object
CreatedById	Single-Byte-String(63)	Reference to the **User** who created this **Lead**
LastModifiedById	Single-Byte-String(63)	Reference to the **User** who modified this **Lead** most recently
Salutation	String(32)	Salutation
Suffix	String(80)	Name suffix
FirstName	String(255)	First name
MiddleName	String(64)	Middle name
LastName	String(128)	Last name
Company	String(1024)	Company name
Title	String(80)	Job title
Phone	String(64)	Phone number
MobilePhone	String(64)	Cellphone number
Fax	String(64)	Fax number
Email	String(500)	Email
Website	String(500)	Website
Street	String(128)	Street address
PostalCode	String(12)	Postcode / ZIP of address
City	String(128)	Name of city
Country	String(32)	Name of country
CountryCode	Single-Byte-String(2)	Two letter ISO code. Only single-byte characters are allowed in this field
State	String(128)	Name of federal state
Description	String(4000)	User edited comments or automatically recognized other information that isn't related to the other fields
AttachmentIdList	LONG String: List of UUID-String(36)	Comma-separated list of UUID-Strings referencing **Attachment** objects related to the **Lead** object
SalesArea	String(4000)	User edited or automatically created text to describe the sales area where the lead might be related to. Sales area might be related to address regions via postal code or to specific questionnaire selection
RequestBarcode	String(1000)	Barcode identifier in case of the lead was collected via ticket barcode-scan and is delivered from trade show organizers visitor database
StatusMessage	String(255)	Error message from trade show organizers visitor database in case of failed lead retrieval by ticket barcode-scan

Name	Type	Description
DeviceId	Int32	Identifier of the local app database on the device where the lead was collected. Usually the Identifier of the user's device. This number is shown in the LeadSuccess app and portal as the first number on "Contact list" or "Contact page".
DeviceRecordId	Int32	Identifier of the lead on the local app database on the device where the lead was collected. This number is shown in the LeadSuccess app and portal as the second number on "Contact list" or "Contact page".
SystemModstamp	DateTime	Timestamp of last modification of the lead object by automatic processing on the LeadSuccess server. Even if LastModifiedDate happened earlier on a device that had no online connection to the LeadSuccess server for a while, SystemModstamp will change after new or modified leads are being received from the LeadSuccess server. This value defaults to LastModifiedDate if no automatic processing occurred yet.
EventID	UUID-String(36)	UUID reference to the **Event** where the Lead was collected
IsReviewed	Int32	Value specifying if the lead yet being reviewed in back-office

By adding a $filter query like the following to your GET request, you can do a call to query for only the newly created **Lead** objects that have been added since the last call or whether there are any **Lead** objects that have been processed on the server since a certain date-time because older data from devices has been received subsequently, because devices weren't online all the time:

```
$filter=(SystemModstamp ge cast('2023-11-02T07:00:00Z','Edm.DateTime'))
```

given the certain date-time of 2. Nov. 2023 at 7:00 UTC when your last call happened. Remember to use greater-or-equal to get changes that happened in the same second of your previous call. You have to use the returned date-time in milliseconds to compare with previously received results.

Remember to call for data of all attachments in the returned AttachmentIdList and check the LastModifiedDate of each attachment in case of SystemModstamp has changed in the **Lead** object.

You can use the $orderby option for specific sort order of your result set:

```
$orderby=KontaktViewId
```

e.g. to order the result set by primary key value. Adding `desc` will order the results descending

Please remember to URL-encode spaces %20 and quotes %27 in your GET request.

LS_LeadReport

Select all entries from this view to get the list of collected visitor contact data with related answers to the questionnaire.

Name	Type	Description
KontaktViewId	Int32	Primary key value
Id	UUID-String(36)	Universal Unique Identifier. Use this value as reference in objects referencing the **Lead** object.
CreatedDate	DateTime	Creation timestamp of the lead object.
LastModifiedDate	DateTime	Timestamp of last modification of the lead object
CreatedById	Single-Byte-String(63)	Reference to the **User** who created this **Lead**
LastModifiedById	Single-Byte-String(63)	Reference to the **User** who modified this **Lead** most recently
Salutation	String(32)	Salutation
Suffix	String(80)	Name suffix
FirstName	String(255)	First name
MiddleName	String(64)	Middle name
LastName	String(128)	Last name
Company	String(1024)	Company name
Title	String(80)	Job title
Phone	String(64)	Phone number
MobilePhone	String(64)	Cellphone number
Fax	String(64)	Fax number
Email	String(500)	Email
Website	String(500)	Website
Street	String(128)	Street address
PostalCode	String(12)	Postcode / ZIP of address
City	String(128)	Name of city
Country	String(32)	Name of country
CountryCode	Single-Byte-String(2)	Two letter ISO code. Only single-byte characters are allowed in this field
State	String(128)	Name of federal state
Description	String(4000)	User edited comments or automatically recognized other information that isn't related to the other fields
IsReviewed	Int32	Value specifying if the lead yet being reviewed in back-office
AttachmentIdList	LONG String: List of UUID-String(2147483647)	Comma-separated list of UUID-Strings referencing **Attachment** objects related to the **Lead** object
SalesArea	String(4000)	User edited or automatically created text to describe the sales area where the lead might be related to. Sales area might be related to address regions via postal code or to specific questionnaire selection
RequestBarcode	String(1000)	Barcode identifier in case of the lead was collected via ticket barcode-scan and is delivered from trade show organizers visitor database
StatusMessage	String(255)	Error message from trade show organizers visitor database in case of failed lead retrieval by ticket barcode-scan

Name	Type	Description
DeviceId	Int32	Identifier of the local app database on the device where the lead was collected. Usually the Identifier of the users device. This number is shown in the LeadSuccess app and portal as the first number on “Contact list” or “Contact page”.
DeviceRecordId	Int32	Identifier of the lead on the local app database on the device where the lead was collected. This number is shown in the LeadSuccess app and portal as the second number on “Contact list” or “Contact page”.
SystemModstamp	DateTime	Timestamp of last modification of the lead object by automatic processing on the LeadSuccess server. Even if LastModifiedDate happened earlier on a device that had no online connection to the LeadSuccess server for a while, SystemModstamp will change after new or modified leads are being received from the LeadSuccess server. This value defaults to LastModifiedDate if no automatic processing occurred yet.
EventID	UUID-String(36)	UUID reference to the **Event** where the Lead was collected
Question01	String(2000)	Question text 1
Answers01	String(2000)	Answer to question 1
Text01	String(2000)	Optional text to question 1
…	…	To
Question30	String(2000)	Question text 30
Answers30	String(2000)	Answer to question 30
Text30	String(2000)	Optional text to question 30

LS_AttachmentById

Call this procedure to select any Attachment related to a Lead object, like the PDF file of the questionnaire, the business card image, questionnaire-related or other attached photos, attached sketches or voice-notes. Attachments can be saved as document files of different file types specified via MIME type. Binary data of the file content is Base64 encoded.

Parameter	Type	Description
Id	UUID-String(36)	UUID of the attachment

The procedure will return **LS_Attachment** data, like row select. Direct query string selection of **LS_Attachment** isn't supported at the moment for performance reason.

LS_Attachment

Name	Type	Description
Id	UUID-String(36)	Universal Unique Identifier. Use this value as reference in objects referencing the **Attachment** object.
CreatedDate	DateTime	Creation timestamp of the attachment object.
LastModifiedDate	DateTime	Timestamp of last modification of the attachment object
CreatedById	Single-Byte-String(63)	Reference to the **User** who created this **Attachment**
LastModifiedById	Single-Byte-String(63)	Reference to the **User** who modified this **Attachment** most recently
Name	String(255)	File name of the attachment
Description	String(4000)	Optional description of the attachment. Description is delivered for attachments containing data-protection policies, business card images or questionnaire-related photos.
ContentType	String(255)	MIME-type of the file data
Body	LONG String	Base64 encoded data of the file
BodyLength	Int32	File size
OwnerId	UUID-String(36)	Reference to the **Lead** object where this **Attachment** object is related to

You can use the following GET-request to select a specific **Attachment** object with Id <attachment-id> from the AttachmentIdList of a **Lead** object.:

```
GET https://<server-name>/<api-name>/LS_AttachmentById?Id='<attachment-id>'&$format=json
```

Please remember to URL-encode quotes %27 in your GET request.

Sample API Application

This chapter introduces a reference implementation of a sample application built to showcase the core functionality of the API in a practical and interactive manner. The example is intended as a technical guide and can be used as a template for developing your own integration. It is important to emphasize that this application is not required to access or use the API. Instead, it provides a convenient way to explore real-world usage patterns, inspect the source code, and better understand the expected request/response behavior, data structures, and integration flows. The goal is to reduce implementation effort by offering a concrete starting point for developers.

Additionally, each page includes a link to download a preconfigured Postman collection. This allows developers to experiment with the individual API calls directly in Postman, using various parameters, if they prefer to explore the API in that environment.

General Application Design

Introduction

LeadSuccess API for Salesforce is a solution that provides access to server-side data structures and procedures via a RESTful API. It enables third-party developers to create customized export interfaces from the LeadSuccess system to the Salesforce CRM system using standard web techniques.

The application offers an intuitive user interface for: - Viewing event data - Managing leads and lead reports - Visualizing attachments - Transferring data to Salesforce

The source code can be found in the following GitHub repository:
https://github.com/conveyGmbH/LSAPISFSamples_www

A sample website is available at the following address:
https://lsapisfsamples.convey.de/

Project Structure

The project is organized as follows:

```
LSAPISAMPLES/
├── .github/                    # GitHub configuration
├── .vscode/                    # VS Code configuration
├── api/                        # API endpoints
├── css/                        # Style files
├── docs/                       # Documentation
├── fonts/                      # Fonts
├── images/                     # Images and graphical resources
├── js/                         # JavaScript code
│   ├── config/                 # Configuration files
│   ├── controllers/            # Controllers for different views
```

```
│   ├── services/                # Services for API calls
│   └── utils/                   # Utility functions
├── pages/                       # HTML pages of the application
├── postman/                     # Postman collection for API testing
├── salesforce-backend/          # Backend code for Salesforce integration
├── .gitignore                   # Git configuration
├── config.txt                   # Application configuration
├── index.html                   # Application home page
├── README.md                    # General documentation
└── staticwebapp.config.json     # Hosting configuration
```

Application Login

To access the application, the user must provide their login credentials.

Login Process:

1. Open the application in your browser
2. Enter the following information on the login screen:
   - Server name (**serverName**)
   - API name (**apiName**)
   - Username (**userName**)
   - Password (**password**)
3. Click on the "**Login**" button

```
// Login code example
async function login() {

    const serverName = document.getElementById("serverName").value.trim();
    const apiName = document.getElementById("apiName").value.trim();
    const userName = document.getElementById("userName").value.trim();
    const password = document.getElementById("password").value.trim();

    if (!serverName || !apiName || !userName || !password) {
      return displayError("Please fill in all fields.");
    }

    try {
      const credentials = btoa(`${userName}:${password}`);
      saveSessionData(serverName, apiName, credentials);
      const apiService = new ApiService(serverName, apiName);
      const response = await apiService.request("GET", "");

      if (response) {
        window.location.href = "/pages/display.html";
      }
    } catch (error) {
      console.error("Error during connection:", error);
      displayError("Error during connection. Please try again.");
```

```
    }
  }
```

Interface Navigation

After logging in, the main interface consists of several sections:

1. **Header**: Contains the LeadSuccess logo and the logout button
2. **Sub-header**: Displays the application title and a link to the documentation
3. **Entity Selector**: Allows you to choose the entity to display (LS_Country, LS_User, LS_Event)
4. **Filters**: Options to filter the displayed data
5. **Data Table**: Display of data with sortable columns
6. **Action Buttons**: Buttons to interact with the selected data

Navigation Between Pages

The application uses the *headerController.js* controller to manage navigation between different pages:

```
// headerController.js - Logo navigation
function handleClickLogo() {
    const logoDiv = document.querySelector('.logo');

    if (logoDiv) {
        logoDiv.style.cursor = 'pointer';

        logoDiv.addEventListener('click', () => {
            const currentPage = window.location.pathname.split('/').pop();
            if (currentPage !== 'display.html') {
                const baseUrl = getBaseUrl();
                window.location.href = `${baseUrl}/display.html`;
            }
        });
    }
}
```

Main Application Components

Data Management

Selecting an Event

Before you can view Leads or Lead Reports, you must first select an Event in the main interface.

4. In the entity selector, choose "**LS_Event**"
5. The table displays the list of available events
6. Click on a row to select an event
7. Once the event is selected, the "**View Leads**" and "**View Lead Reports**" buttons are activated

```
// Event selection code example
function handleRowClick(item, event) {
  const row = event.currentTarget;
  const tbody = document.querySelector('tbody');

  if (row.classList.contains('selected')) {
    row.classList.remove('selected');
    selectedEventId = null;
    sessionStorage.removeItem('selectedEventId');
    updateButtonState(false);
  } else {
    const previouslySelected = tbody.querySelector('tr.selected');
    if (previouslySelected) {
      previouslySelected.classList.remove('selected');
    }

    row.classList.add('selected');
    selectedEventId = item.Id;
    sessionStorage.setItem('selectedEventId', selectedEventId);

    if (currentEntity === ACTIVATING_ENTITY) {
      updateButtonState(true);
    } else {
      updateButtonState(false);
    }
  }
}
```

Viewing Leads

After selecting an Event, you can access the list of Leads associated with that event.

1. Click on the "**View Leads**" button
2. The application displays the list of Leads for the selected event
3. You can:
   - Filter Leads using the filtering options
   - Sort Leads by clicking on column headers
   - View attachments by selecting a Lead and then clicking on "**Show Attachment**"
   - Transfer a Lead to Salesforce by clicking on "Transfer to Salesforce"

Viewing Lead Reports

Lead Reports provide detailed information about Leads, including questionnaire responses.

1. Click on the "**View Lead Reports**" button
2. The application displays the list of Lead Reports for the selected event
3. The functionality is similar to that of viewing Leads

Attachment Management

LeadSuccess API for Salesforce allows you to view attachments associated with Leads.

Supported Attachment Types:

- Images (PNG, JPEG, SVG)
- PDF documents
- Audio files
- Video files
- Other file formats

Viewing Attachments:

1. Select a Lead or Lead Report from the list
2. If attachments are available, the "**Show Attachment**" button is activated
3. Click on "Show Attachment" to display the attachments
4. The application displays the attachments as tabs
5. Click on a tab to view the corresponding attachment
6. Use the "Download" button to download the attachment

```
// Code example for displaying attachments
function displayAttachment(attachment) {

  const attachmentContainer = document.getElementById('attachmentContainer');
  const fileNameElement = document.getElementById('fileName');
  const downloadButton = document.getElementById('downloadButton');

  if (!attachment) {
    attachmentContainer.innerHTML = '<div class="no-data"><p>No attachment
available</p></div>';
    fileNameElement.textContent = 'No file';
    downloadButton.style.display = 'none';
    return;
  }

  // Attachment properties

  const fileName = attachment.Name || 'attachment';
  const fileType = attachment.ContentType;
  const base64Data = attachment.Body;
  const fileSize = attachment.BodyLength;

  // Update file name display
  fileNameElement.textContent = fileName;
  downloadButton.style.display = 'inline-flex';
  downloadButton.disabled = false;

  if (!base64Data || !fileType) {
    attachmentContainer.innerHTML = '<div class="no-data"><p>Missing data for this
attachment</p></div>';
    return;
  }

  try {
```

```
    const dataUrl = `data:${fileType};base64,${base64Data}`;

    // Configure download button
    downloadButton.onclick = () => {
      const a = document.createElement('a');
      a.href = dataUrl;
      a.download = fileName;
      document.body.appendChild(a);
      a.click();
      document.body.removeChild(a);
    };

    // Display based on file type
    if (fileType === 'image/svg+xml' || fileName.toLowerCase().endsWith('.svg')) {
      // Code for SVG files
    } else if (fileType.startsWith('image/')) {
      // Code for images
      attachmentContainer.innerHTML = `<img src="${dataUrl}" alt="${fileName}"
style="max-width: 100%; max-height: 500px;" />`;
    } else if (fileType === 'application/pdf') {
      // Code for PDFs
      attachmentContainer.innerHTML = `
        <iframe
          src="${dataUrl}#view=Fit&scrollbar=0"
          class="pdf-viewer"
          type="application/pdf"
          style="width: 100%; height: 100%; border: none;"
        >
         <p>Your browser does not support displaying PDFs. <a href="${dataUrl}"
download="${fileName}">Download the PDF</a> to view it.</p>
        </iframe>`;
    } else if (fileType.startsWith('audio/')) {
      // Code for audio
      attachmentContainer.innerHTML = `
        <audio controls style="width: 100%;">
          <source src="${dataUrl}" type="${fileType}">
          Your browser does not support this audio element.
        </audio>`;
    } else if (fileType.startsWith('video/')) {
      // Code for video
      attachmentContainer.innerHTML = `
        <video controls style="width: 100%; max-height: 500px;">
          <source src="${dataUrl}" type="${fileType}">
          Your browser does not support this video element.
        </video>`;
    } else {
      // Other file types
      attachmentContainer.innerHTML = `
        <div class="no-data">
          <p>${fileName} (${(fileSize / 1024).toFixed(2)} KB)</p>
          <p>Preview not available for this file type (${fileType})</p>
          <p>Use the download button to open the file</p>
        </div>`;
    }
  } catch (error) {
```

```
    console.error("Error displaying attachment:", error);
    showError(`Error displaying attachment: ${error.message}`);
    attachmentContainer.innerHTML = `<div class="no-data"><p>Error displaying
attachment: ${error.message}</p></div>`;
  }
}
```

LeadSuccess API Management

Data Structure

For the complete data structure details, please refer to the existing LeadSuccess API for Salesforce data schema documentation.

You can retrieve the full schema data with the following request:

GET https://<server-name>/<api-name>/$metadata

Important Note: The OData request syntax is case sensitive! This is a common issue that clients encounter when working with the API.

A key aspect of the implementation is that the column headers displayed in the sample site interface exactly match the attribute names in the JSON response from API requests. This 1:1 mapping follows the OData schema, making it intuitive to understand the data structure.

Additionally, the attribute names provided in the JSON response from API requests correspond to the attribute names of the associated Salesforce objects, which simplifies the data transfer process.

Example Requests and Responses

LS_Country

```
Request: GET https://lstest.convey.de/apisftest/LS_Country?$format=json

Response:
{
  "d": {
    "results": [
      {
        "__metadata": {
          "uri": "http://localhost:8091/odata_apisf/LS_Country('AC')",
          "type": "LeadSuccessAPISFOData.LS_Country"
        },
        "Id": "AC",
        "LGNTINITLandViewId": 247,
        "Country": "Ascension"
      }
```

```
      // More country entries...
    ]
  }
}
```

LS_User

Request: GET https://lstest.convey.de/apisftest/LS_User?$format=json

```
Response:
{
  "d": {
    "results": [
      {
        "__metadata": {
          "uri":
"http://localhost:8091/odata_apisf/LS_User('admin1%40512.leadsuccess.de')",
          "type": "LeadSuccessAPISFOData.LS_User"
        },
        "Id": "admin1@512.leadsuccess.de",
        "MitarbeiterViewId": 1244,
        "FirstName": "Admin",
        "LastName": "IBM API Test",
        "Email": null,
        "Phone": null,
        "MobilePhone": null,
        "Street": null,
        "PostalCode": null,
        "City": null,
        "Country": null,
        "CountryCode": null,
        "CurrentStatus": "CustomerAdmin",
        "EventId": "a098de14-34e9-4170-9776-34e3d94c272a"
      }
      // More user entries...
    ]
  }
}
```

LS_Event

Request:
CopyGET https://lstest.convey.de/apisftest/LS_Event?$format=json

```
Response:{
  "d": {
    "results": [
      {
        "__metadata": {
          "uri": "http://localhost:8091/odata_apisf/LS_Event('9b1763d8-1c4e-47ce-8903-
41b74b0720e9')",
          "type": "LeadSuccessAPISFOData.LS_Event"
```

```
        },
        "Id": "9b1763d8-1c4e-47ce-8903-41b74b0720e9",
        "VeranstaltungViewId": 317,
        "CreatedDate": "/Date(1537353034106)/",
        "LastModifiedDate": "/Date(1723646756169)/",
        "Subject": "API Test",
        "StartDate": "/Date(1546300800000)/",
        "EndDate": "/Date(1956441600000)/",
        "Type": "Test Veranstalter LS-TEST",
        "EventSubtype": "API Test Accounts",
        "Description": null
      }
      // More event entries...
    ]
  }
}
```


LeadSuccess API for Salesforce uses the OData protocol to access data. The main entities are:

1. **LS_Country**: Information about countries
   - Id: Two-letter ISO code
   - Country: Country name
2. **LS_Event**: Information about events
   - Id: Universal Unique Identifier (UUID)
   - Subject: Event name
   - StartDate: Start date
   - EndDate: End date
   - Type: Event type
   - EventSubtype: Event subtype
   - Description: Event description
3. **LS_User**: Information about users
   - Id: User login name
   - FirstName: First name
   - LastName: Last name
   - Email: Email address
   - Phone: Phone number
   - EventId: Reference to the event to which the user is attached
4. **LS_Lead**: Information about leads
   - Id: Universal Unique Identifier (UUID)
   - FirstName: First name
   - LastName: Last name
   - Company: Company
   - Email: Email address
   - Phone: Phone number
   - Country: Country

- AttachmentIdList: List of attachments
- EventId: Reference to the event

5. **LS_LeadReport**: Detailed information about leads with questionnaire responses
   - Same fields as LS_Lead
   - Question01 to Question30: Questionnaire questions
   - Answers01 to Answers30: Answers to questions
   - Text01 to Text30: Optional text for each question
6. **LS_Attachment**: Information about attachments
   - Id: Universal Unique Identifier (UUID)
   - Name: File name
   - ContentType: MIME type
   - Body: Base64-encoded file data
   - BodyLength: File size
   - OwnerId: Reference to the Lead to which the attachment is associated

ApiService.js - Main Communication Service

The ApiService service is the cornerstone of communication between the application and the LeadSuccess API. It handles HTTP requests, authentication, and response processing.

Architecture and Key Features

```
// apiService.js
export default class ApiService {
  constructor(serverName, apiName) {
    this.serverName = serverName;
    this.apiName = apiName;
    this.credentials = sessionStorage.getItem("credentials") || null;
  }

  // Main method for making API requests
  async request(method, endpoint, data = null) {
    // ...
  }

  // Method to get the next page URL (pagination)
  getNextUrl(data) {
    // ...
  }

  // Method to fetch the next page of results
  async fetchNextRows(nextUrl) {
    // ...
  }

  // Method to log out of the application
```

```
  logout() {
    // ...
  }

  // Error handling methods
  handleHttpError(status, errorData = {}) {
    // ...
  }

  handleNetworkError(error) {
    // ...
  }

  showError(message, shake = false) {
    // ...
  }
}
```

Main Method: Request

The request method is the heart of the ApiService service. It handles all communication with the API.

```
async request(method, endpoint, data = null) {
  const errorElement = document.getElementById("errorMessage");
  if (errorElement) errorElement.style.display = "none";

  try {
    // Check that credentials are available
    if (!this.credentials) {
      throw new Error("No credentials found");
    }

    // Prepare HTTP headers with basic authentication
    const headers = new Headers({
      Accept: "application/json",
      Authorization: `Basic ${this.credentials}`,
    });

    // Add Content-Type header for methods other than GET
    if (method !== "GET") {
      headers.append("Content-Type", "application/json");
    }

    // Request configuration
    const config = {
      method,
      headers,
      credentials: "same-origin",
    };

    // Add request body for methods with data
    if (data) {
      config.body = JSON.stringify(data);
    }
```

```
    // Build the complete URL
    const url = `https://${this.serverName}/${this.apiName}/${endpoint}`;

    // Execute the request
    const response = await fetch(url, config);

    // Handle HTTP errors
    if (!response.ok) {
      const errorData = await response.json().catch(() => ({}));
      this.handleHttpError(response.status, errorData);
      return null;
    }

    // Parse and return the JSON response
    return await response.json();
  } catch (error) {
    // Handle network errors
    this.handleNetworkError(error);
    return null;
  }
}
```

Pagination Handling

LeadSuccess API uses pagination to handle large datasets. The ApiService service offers specific methods to facilitate this pagination.

```
// Method to extract the next page URL from the response
getNextUrl(data) {
  let url = "";
  if (data && data.d) {
    const next = data.d.__next;

    if (next && typeof next === "string") {
      const viewNamePos = next.lastIndexOf("/");

      if (viewNamePos >= 0) {
        // Build the complete URL for pagination
        url =
`https://${this.serverName}/${this.apiName}${next.substring(viewNamePos)}`;
      }
    }
  }
  return url;
}

// Method to fetch the next page of results
async fetchNextRows(nextUrl) {
  const errorElement = document.getElementById("errorMessage");
  if (errorElement) errorElement.style.display = "none";

  try {
    const headers = new Headers({
```

```
      Accept: "application/json",
      Authorization: `Basic ${this.credentials}`,
    });

    const config = {
      method: "GET",
      headers,
      credentials: "same-origin",
    };

    console.log("Fetching next rows with URL:", nextUrl);
    const response = await fetch(nextUrl, config);

    if (!response.ok) {
      const errorData = await response.json().catch(() => ({}));
      this.handleHttpError(response.status, errorData);
      return null;
    }

    return await response.json();
  } catch (error) {
    this.handleNetworkError(error);
    return null;
  }
}
```

Salesforce Transfer

LeadSuccess API for Salesforce offers robust integration with Salesforce CRM, allowing you to transfer Lead data directly to your Salesforce instance. This section details the entire authentication, connection, and transfer process.

Salesforce Integration Architecture

The Salesforce integration relies on three main components:

1. **Front-end (JavaScript)**: Handles the user interface, client-side authentication and API calls
2. **Backend (Node.js)**: Serves as a secure intermediary between the application and Salesforce
3. **Salesforce API**: REST API for creating Leads in Salesforce

The interaction between these components uses the OAuth 2.0 flow for secure authentication.

Salesforce Authentication Process

Authentication with Salesforce follows the standard OAuth 2.0 flow:

1. **Initialization**: The client requests an authentication URL from the backend
2. **Authentication**: The user is redirected to the Salesforce login page
3. **Authorization**: The user authorizes the application to access their Salesforce account

4. **Token Exchange**: Salesforce returns an authorization code that the backend exchanges for an access token
5. **Session**: The backend generates a unique session token that is returned to the client

```
// salesforceService.js - Method to get the authentication URL
async getAuthUrl() {
  try {
    const response = await fetch(`${this.apiBaseUrl}/salesforce/auth`, {
      method: 'GET',
      headers: {
        'Accept': 'application/json',
        'Cache-Control': 'no-cache, no-store, must-revalidate'
      }
    });

    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }

    const data = await response.json();
    console.log('Auth URL received');
    return data.authUrl;
  } catch (error) {
    console.error('Error getting auth URL:', error);
    throw error;
  }
}
```

Lead Transfer Process

The complete process of transferring a Lead to Salesforce includes the following steps:

1. **Select a Lead**: Select a Lead or Lead Report from the list
2. **Initiate Transfer**: Click on the “Transfer to Salesforce” button
3. **Connection Check**: The application checks if you are already connected to Salesforce
4. **Authentication**: If necessary, a Salesforce authentication window opens
5. **Data Preparation**: The application prepares and validates the Lead data
6. **Duplicate Check**: The system checks if the Lead already exists in Salesforce
7. **Lead Transfer**: The data is sent to Salesforce via the backend
8. **Confirmation**: The application displays the transfer result with the Salesforce ID

```
// displayLeadTransferController.js - Complete transfer function
async function continueTransferWithToken() {
  const transferStatus = document.getElementById('transferStatus');
  const transferBtn = document.getElementById('transferToSalesforceBtn');

  try {
    // Display transfer status
    transferStatus.innerHTML = `
      <div class="transfer-pending">
```

```
          <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0
   0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-
   linecap="round" stroke-linejoin="round">
      <circle cx="12" cy="12" r="10"/>
      <polyline points="12 6 12 12 16 14"/>
    </svg>
    Lead transfer in progress...
  </div>
`;

// Prepare Lead data
const leadDataToSend = {...selectedLeadData};

// Clean invalid data
delete leadDataToSend.State; // Avoids Salesforce validation issues
for (let field in leadDataToSend) {
  if (
    leadDataToSend[field] === 'N/A' ||
    leadDataToSend[field] === 'null' ||
    leadDataToSend[field] === 'undefined' ||
    leadDataToSend[field] === null ||
    leadDataToSend[field] === undefined
  ) {
    // Remove fields with invalid values
    delete leadDataToSend[field];
  }
}

// Handle required Salesforce fields
if (!leadDataToSend.LastName || leadDataToSend.LastName === 'N/A') {
  leadDataToSend.LastName = 'Unknown';
}
if (!leadDataToSend.Company || leadDataToSend.Company === 'N/A') {
  leadDataToSend.Company = 'Unknown';
}
if (leadDataToSend.Country === 'N/A') {
  delete leadDataToSend.Country;
}

// Call the transfer API
const response = await fetch(`${appConfig.apiBaseUrl}/direct-lead-transfer`, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    sessionToken,
    leadData: leadDataToSend
  })
});

const result = await response.json();

// Handle authentication errors
if (!response.ok) {
```

```
      // If the session has expired, restart authentication
      if (response.status === 401) {
        localStorage.removeItem('sf_session_token');
        sessionToken = null;
        handleTransferButtonClick();
        return;
      }
      throw new Error(result.message || `Error ${response.status}`);
    }

    // Display successful transfer result
    transferStatus.innerHTML = `
      <div class="status-success">
        <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24
24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-
linejoin="round">
          <path d="M22 11.08V12a10 10 0 1 1-5.93-9.14"/>
          <polyline points="22 4 12 14.01 9 11.01"/>
        </svg>
        Lead successfully transferred to Salesforce
      </div>
      <div class="status-details">
        <p><strong>Salesforce ID:</strong> ${result.leadId}</p>
        <p><strong>Status:</strong> ${result.status}</p>
        <p><strong>Message:</strong> ${result.message}</p>
      </div>
    `;

    // Update connection status
    updateConnectionStatus('connected', 'Connected to Salesforce');
  } catch (error) {
    console.error('Error during transfer:', error);

    // Display errors
    transferStatus.innerHTML = `
      <div class="status-error">
        <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24
24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-
linejoin="round">
          <circle cx="12" cy="12" r="10"/>
          <line x1="15" y1="9" x2="9" y2="15"/>
          <line x1="9" y1="9" x2="15" y2="15"/>
        </svg>
        Transfer failed: ${error.message || 'Unknown error'}
      </div>
    `;
  } finally {
    // Reset button state
    isTransferInProgress = false;
    transferBtn.disabled = false;
    transferBtn.innerHTML = `
      <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0 24
24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-
linejoin="round">
        <path d="M12 5v14M19 12l-7 7-7-7"/>
```

```
      </svg>
      Transfer to Salesforce
    `;
  }
}
```

Server-Side Implementation (server.js)

The backend handles secure interaction with Salesforce via the jsforce module, and implements the following endpoints:

1. Authentication URL Generation (/api/salesforce/auth)

```
// server.js - Authentication URL generation endpoint

apiRouter.get('/salesforce/auth', (req, res) => {

  try {
    const authUrl =
`${SF_LOGIN_URL}/services/oauth2/authorize?response_type=code&client_id=${SF_CLIENT_ID
}&redirect_uri=${encodeURIComponent(SF_REDIRECT_URI)}&scope=api%20id%20web%20refresh_t
oken`;
    res.json({ authUrl });
  } catch (error) {
    console.error('Error generating auth URL:', error);
    res.status(500).json({ error: 'Failed to generate authorization URL' });
  }
});
```

2. OAuth Callback (/api/oauth2/callback)

Processes the Salesforce response after authentication, exchanges the code for a token, and stores the session information.

3. Session Check (/api/salesforce/session-check)

```
// server.js - Session check endpoint

apiRouter.get('/salesforce/session-check', (req, res) => {
  const sessionToken = req.headers['x-session-token'];

  if (!sessionToken || !tokenStore.has(sessionToken)) {
    return res.status(401).json({
      success: false,
      message: 'Invalid or expired session'
    });
  }

  // Check session expiration (1 hour)
  const session = tokenStore.get(sessionToken);
  const now = Date.now();
```

```
  if (now - session.timestamp > 3600000) {
    tokenStore.delete(sessionToken);
    return res.status(401).json({
      success: false,
      message: 'Session expired'
    });
  }

  return res.json({
    success: true,
    message: 'Session valid'
  });
});
```

4. Lead Transfer (/api/direct-lead-transfer)

Main endpoint that receives Lead data, checks for potential duplicates, and creates the Lead in Salesforce.

```
// server.js - Extract from the transfer endpoint
// Create Lead in Salesforce
try {
  const result = await conn.sobject('Lead').create(sfLeadData);

  if (result.success) {
    console.log('Lead created successfully, ID:', result.id);
    return res.json({
      success: true,
      leadId: result.id,
      status: 'Transferred',
      message: 'Lead successfully transferred to Salesforce'
    });
  } else {
    console.error('Lead creation failed:', result.errors);
    return res.status(400).json({
      success: false,
      message: `Failed to create lead: ${result.errors.join(', ')}`
    });
  }
} catch (sfError) {
  console.error('Salesforce API error:', sfError.message);
  return res.status(400).json({
    success: false,
    message: `Salesforce error: ${sfError.message}`
  });
}
```

Salesforce Configuration

For the integration to work, you must configure a connected app in Salesforce with the following settings:

1. **Application Name**: LeadSuccess API Integration
2. **OAuth Callback URL**: Full URL of your callback endpoint (e.g., https://yourdomain.com/api/oauth2/callback)
3. **Required OAuth Scopes**: api, id, web, refresh_token

The following environment variables must be configured on the server:

SF_CLIENT_ID=your_client_id
SF_CLIENT_SECRET=your_client_secret
SF_LOGIN_URL=https://login.salesforce.com

For Salesforce sandbox environments, use https://test.salesforce.com as SF_LOGIN_URL.

Transfer Error Handling

The application handles several types of errors that can occur during transfer:

1. **Authentication Errors**: Session expiration, invalid tokens
2. **Lead Duplicates**: Automatic detection of existing Leads by email
3. **Validation Errors**: Missing or invalid data (e.g., unsupported country)
4. **Salesforce API Errors**: Issues with the Salesforce API (quotas, permissions)
5. **Network Errors**: Connection issues between the client, backend, and Salesforce

SalesforceService.js - Salesforce Integration Service

The SalesforceService service is specifically designed to handle integration with Salesforce CRM. It manages authentication, connection status checking, and data transfer to Salesforce.

Architecture and Key Features

```
// salesforceService.js
import { appConfig } from "../config/salesforceConfig.js";

class SalesforceService {
  constructor() {
    this.apiBaseUrl = appConfig.apiBaseUrl;
    this.sessionToken = localStorage.getItem('sf_session_token') || null;
  }

  // Check Salesforce connection status
  async checkConnection() {
    // ...
  }

  // Get Salesforce authentication URL
  async getAuthUrl() {
    // ...
  }
```

```
  // Session token management
  setSessionToken(token) {
    // ...
  }

  // Transfer a lead to Salesforce
  async transferLead(leadData) {
    // ...
  }

  // Logout from Salesforce
  async logout() {
    // ...
  }
}

export default SalesforceService;
```

Connection Check

```
const response = await fetch(`${this.apiBaseUrl}/salesforce/connection-status`, {
      method: 'GET',
      headers: {
        'Accept': 'application/json',
        'Cache-Control': 'no-cache, no-store, must-revalidate',
        'X-Session-Token': this.sessionToken || ''
      }
    });

    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }

    const status = await response.json();

    return status;
  } catch (error) {
    console.error('Connection check error:', error);
    return { connected: false, error: error.message };
  }
}
```

Salesforce Authentication

The getAuthUrl method allows you to get the Salesforce authentication URL to start the OAuth authentication process.

```
async getAuthUrl() {

  try {
    console.log('Requesting Salesforce auth URL...');
    const response = await fetch(`${this.apiBaseUrl}/salesforce/auth`, {
      method: 'GET',
```

```
      headers: {
        'Accept': 'application/json',
        'Cache-Control': 'no-cache, no-store, must-revalidate'
      }
    });

    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }

    const data = await response.json();
    console.log('Auth URL received');
    return data.authUrl;
  } catch (error) {
    console.error('Error getting auth URL:', error);
    throw error;
  }
}
```

Lead Transfer to Salesforce

The transfer Lead method is the main function that handles sending Lead data to Salesforce.

```
async transferLead(leadData) {
  try {
    // Check for session token presence
    if (!this.sessionToken) {
      console.error("Error: No session token available");
      throw new Error('Not connected to Salesforce. Please connect first.');
    }

    // Prepare request options
    const requestOptions = {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
        'Cache-Control': 'no-cache, no-store, must-revalidate',
        'X-Session-Token': this.sessionToken
      },
      body: JSON.stringify(leadData)
    };

    try {
      // Execute the request
      const response = await fetch(`${this.apiBaseUrl}/salesforce/transfer-lead`,
requestOptions);

      let resultData;

      // Process response with robust error handling
      try {
        const responseText = await response.text();
        console.log('Response text:', responseText);
```

```
        try {
          resultData = JSON.parse(responseText);
          console.log('JSON response parsed:', resultData);
        } catch (parseError) {
          console.error('JSON parsing error:', parseError);
          throw new Error(`Non-JSON response: ${responseText}`);
        }
      } catch (textError) {
        console.error('Error reading response text:', textError);
        throw new Error('Unable to read server response');
      }

      // Handle HTTP errors
      if (!response.ok) {
        console.error('Non-OK HTTP response:', response.status, resultData);

        // Special handling for expired sessions
        if (response.status === 401) {
          console.log('Session expired, removing token');
          this.setSessionToken(null);
        }

        throw new Error(resultData.message || `HTTP error: ${response.status}`);
      }

      // Return data on success
      console.log('Transfer successful:', resultData);
      return resultData;
    } catch (fetchError) {
      console.error('Fetch error:', fetchError);
      throw fetchError;
    }
  } catch (error) {
    console.error('Error in transferLead:', error);
    console.error('Stack trace:', error.stack);
    throw error;
  } finally {
    console.log('===== END TRANSFERLEAD =====');
  }
}
```

Session Token Management

The setSessionToken method handles storing and removing the session token in localStorage.

```
setSessionToken(token) {
  this.sessionToken = token;
  if (token) {
    localStorage.setItem('sf_session_token', token);
    console.log('Session token saved:', token.substring(0, 6) + '...');
  } else {
    localStorage.removeItem('sf_session_token');
    console.log('Session token cleared');
```

```
  }
}
```

Salesforce Logout

The logout method allows you to properly log out of Salesforce.

```
async logout() {
  try {
    console.log('Logging out from Salesforce...');

    if (!this.sessionToken) {
      return { success: true, message: 'Already logged out' };
    }

    const response = await fetch(`${this.apiBaseUrl}/salesforce/logout`, {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Cache-Control': 'no-cache, no-store, must-revalidate',
        'X-Session-Token': this.sessionToken
      }
    });

    // Remove session token regardless of outcome
    this.setSessionToken(null);

    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }

    const result = await response.json();
    return result;
  } catch (error) {
    console.error('Logout error:', error);
    // Remove token even in case of error
    this.setSessionToken(null);
    throw error;
  }
}
```

Postman Collection

LeadSuccess API for Salesforce provides a complete Postman collection to facilitate testing and integration with the API. This collection contains all available endpoints, with preconfigured

examples and automated tests.

Getting the Postman Collection

There are two methods to get the Postman collection:

Method 1: From the Application Interface

6. Log in to the LeadSuccess API application
7. Click on the “Test API in Postman” button in the application header
8. A modal window displays with information about the collection
9. Click on “Download Collection” to download the collection JSON file

```
// postman-integration.js - Collection download button
document.getElementById('downloadBtn').addEventListener('click', function() {
    // Path to the collection JSON file
const jsonFileUrl = `${window.location.origin}/postman/LeadSuccess-API-
Collection.json`;


    // Create download link
    const downloadLink = document.createElement('a');
    downloadLink.href = jsonFileUrl;
    downloadLink.download = 'LeadSuccess-API-Collection.json';
    document.body.appendChild(downloadLink);
    downloadLink.click();
    document.body.removeChild(downloadLink);

    // Change the modal content after download
    // ...
});
```

Method 2: Direct Access to the File

The collection file is available at the following address:

https://[your-domain]/postman/LeadSuccess-API-Collection.json

Postman Collection Structure

The collection is organized hierarchically to reflect the API structure:

```
LeadSuccess API Collection/
├── Authentication/
│   └── Login
├── Event Management/
│   ├── Get All Events
│   └── Get Event By ID
├── Lead Management/
│   ├── Get Leads by Event
│   ├── Get Lead by ID
```

```
│   └── Filter Leads
├── Lead Report Management/
│   ├── Get Lead Reports by Event
│   └── Get Lead Report by ID
├── Attachment Management/
│   └── Get Attachment by ID
└── Salesforce Integration/
    ├── Get Auth URL
    ├── Check Session
    └── Transfer Lead
```

Environment Variables Configuration

To use the collection effectively, you must configure a Postman environment with the following variables:

Variable	Description	Example
serverName	LeadSuccess server name	api.leadsuccess.com
apiName	API name	LeadSuccess-API
username	Username	user123
password	Password	password123
auth	Authentication token (automatically generated)	*automatically generated*
eventId	Event ID to filter Leads	550e8400-e29b-41d4-a716-446655440000
leadId	ID of a specific Lead	6b86b273-eb8a-4b7c-8c3d-cfc23b9a6000
attachmentId	ID of an attachment	d4735e3a-5b4a-4a8e-9b1d-dccebf0bc000
sfSessionToken	Salesforce session token	*generated during authentication*

Pre-Request Scripts and Automated Tests

The collection includes scripts to automate certain tasks:

Pre-Request Script for Authentication

Collection JSON File Example

Here is an excerpt from the collection JSON file to illustrate its structure:

```
{
  "info": {
    "_postman_id": "a1b2c3d4-e5f6-7890-a1b2-c3d4e5f67890",
    "name": "LeadSuccess API Collection",
    "description": "Collection for testing the LeadSuccess API for Salesforce",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
```

```
"item": [
  {
    "name": "Authentication",
    "item": [
      {
        "name": "Login",
        "request": {
          "method": "GET",
          "header": [
            {
              "key": "Authorization",
              "value": "Basic {{auth}}",
              "type": "text"
            }
          ],
          "url": {
            "raw": "https://{{serverName}}/{{apiName}}/?$format=json",
            "protocol": "https",
            "host": ["{{serverName}}"],
            "path": ["{{apiName}}", ""],
            "query": [
              {
                "key": "$format",
                "value": "json"
              }
            ]
          },
          "description": "Authenticate with the LeadSuccess API"
        },
        "response": []
      }
    ]
  },
  {
    "name": "Event Management",
    "item": [
      {
        "name": "Get All Events",
        "request": {
          "method": "GET",
          "header": [
            {
              "key": "Authorization",
              "value": "Basic {{auth}}",
              "type": "text"
            }
          ],
          "url": {
            "raw": "https://{{serverName}}/{{apiName}}/LS_Event?$format=json",
            "protocol": "https",
            "host": ["{{serverName}}"],
            "path": ["{{apiName}}", "LS_Event"],
            "query": [
              {
                "key": "$format",
                "value": "json"
              }
            ]
          },
          "description": "Get all events"
        },
        "response": []
      }
```

```
      ]
    }
    // ... other endpoints ...
  ]
}
```

Using the Collection

1. Import the JSON file into Postman (File > Import)
2. Create a new environment (Environment) and configure the required variables
3. Select the created environment from the dropdown list in the top right
4. Start by running the “Login” request in the “Authentication” section
5. Once authenticated, you can test the other endpoints

You can also use Postman’s “Runner” function to run an automated test suite on all endpoints.

Utils Helper

The helper.js module provides a set of essential utility functions used throughout the application to manage date formatting, array sorting, filter handling, and other common functionalities.

Date Formatting and Manipulation

```
// Date formatting for display
export function formatDate(dateString) {
  if (!dateString) return 'N/A';

  // Handle specific "/Date(timestamp)/" format
  if (typeof dateString === 'string' && dateString.includes('/Date(')) {
    const timestamp = dateString.match(/\/Date\((\d+)\)\//);
    if (timestamp && timestamp[1]) {
      const date = new Date(parseInt(timestamp[1]));
      return date.toISOString().split('T')[0]; // YYYY-MM-DD format
    }
  }

  // Handle standard dates
  try {
    const date = new Date(dateString);
    if (isNaN(date.getTime())) return dateString;
    return date.toISOString().split('T')[0];
  } catch (e) {
    return dateString;
```

```
  }
}

// Parse dates in different formats (maximum flexibility)
export function parseDate(dateString) {
  if (!dateString) return null;

  // dd.mm.yyyy format (European format)
  if (/^\d{2}\.\d{2}\.\d{4}$/.test(dateString)) {
    const parts = dateString.split('.');
    const date = new Date(parseInt(parts[2]), parseInt(parts[1])-1,
parseInt(parts[0]));
    return date;
  }

  // yyyy-mm-dd format (ISO format)
  if (/^\d{4}-\d{2}-\d{2}$/.test(dateString)) {
    return new Date(dateString);
  }

  // dd/mm/yyyy format
  if (/^\d{2}\/\d{2}\/\d{4}$/.test(dateString)) {
    const parts = dateString.split('/');
    return new Date(parseInt(parts[2]), parseInt(parts[1])-1, parseInt(parts[0]));
  }

  // General parsing attempt
  const date = new Date(dateString);
  if (!isNaN(date.getTime())) {
    return date;
  }

  return null;
}

// Date formatting for OData queries
export function formatDateForOData(date) {
  if (!(date instanceof Date) || isNaN(date)) {
    console.error("Invalid date:", date);
    return null;
  }

  const year = date.getFullYear();
  const month = (date.getMonth() + 1).toString().padStart(2, '0');
  const day = date.getDate().toString().padStart(2, '0');
  return `${year}-${month}-${day}`;
}
```

Data Table Management

```
// Table sorting
export function sortTable(index, th) {
  let sortAsc = !th.classList.contains('asc');
  const tableRows = document.querySelectorAll('tbody tr');

  [...tableRows].sort((a, b) => {
```

```
    let firstRow = a.querySelectorAll('td')[index].textContent.toLowerCase();
    let secondRow = b.querySelectorAll('td')[index].textContent.toLowerCase();
    return sortAsc ? (firstRow > secondRow ? 1 : -1) : (firstRow < secondRow ? 1 : -
1);
  }).forEach(sortedRow => document.querySelector('tbody').appendChild(sortedRow));

  th.classList.toggle('asc', sortAsc);
  th.classList.toggle('desc', !sortAsc);
}

// Table and message clearing
export function clearTable() {
  const tableHead = document.getElementById('tableHead');
  const tableBody = document.getElementById('tableBody');
  if (tableHead) tableHead.innerHTML = '';
  if (tableBody) tableBody.innerHTML = '';
  const noDataMessage = document.getElementById('noDataMessage');
  if (noDataMessage) noDataMessage.textContent = '';
  const searchInput = document.getElementById('search');
  if (searchInput) searchInput.value = '';
}
```

OData Query Functions

```
// Escape values for OData queries
export function escapeODataValue(value) {
  return value.replace(/'/g, "''");
}

// Reset filters
export function resetFilters(entity, fields) {
  localStorage.removeItem(`${entity}_Filters`);
  fields.forEach(field => {
    const input = document.getElementById(`filter-${field}`);
    if (input) {
      input.value = '';
    }
  });
  clearTable();
  const noDataMessage = document.getElementById('noDataMessage');
  noDataMessage.textContent = 'Filters have been reset. Please enter new values and
click "Apply Filters".';
}
```

Pagination Management

```
// Pagination setup
export function setupPagination(apiService, displayDataFunction) {
  let nextUrl = '';

  // Function to update next URL
  function updateNextUrl(data) {
    nextUrl = apiService.getNextUrl(data);
    const nextButton = document.getElementById('nextButton');
    if (nextButton) {
      nextButton.disabled = !nextUrl;
    }
```

```
  return nextUrl;
}

// Function to load next rows
async function loadNextRows() {
  if (!nextUrl) {
    console.error('No next URL found.');
    return;
  }

  const nextButton = document.getElementById('nextButton');
  if (nextButton) {
    nextButton.textContent = 'Loading...';
    nextButton.disabled = true;
  }

  try {
    const data = await apiService.fetchNextRows(nextUrl);

    if (data && data.d && data.d.results && data.d.results.length > 0) {
      // Display data
      displayDataFunction(data.d.results, true);

      // Update next URL
      updateNextUrl(data);
    } else {
      nextUrl = '';
      if (nextButton) {
        nextButton.disabled = true;
      }
    }
  } catch (error) {
    console.error("Error loading next rows:", error);
  } finally {
    if (nextButton) {
      nextButton.textContent = 'Next';
      nextButton.disabled = !nextUrl;
    }
  }
}

// Function to initialize pagination buttons
function initPagination() {
  const nextButton = document.getElementById('nextButton');
  if (nextButton) {
    nextButton.addEventListener('click', loadNextRows);
    nextButton.disabled = true;
  }
}

// Return an object with pagination functions
return {
  updateNextUrl,
  loadNextRows,
  initPagination
```

```
  };
}
```

Error codes

For all requests, a result code and possibly a textual error message is returned.

The error message serves to describe the error for a developer as exactly as possible. Ideally, it should be saved in case of error in a log file. It is not intended to be displayed to an end user (exhibitor).

The result code is roughly based on the HTTP status codes and is intended as the basis for an automatic response to specific errors. The three-digit numeric error code can be followed by a detail error code with a dot.

The rough classification of the error is made possible by the first digit of the result code:

• "**2**" for success

• "**4**" for client-side errors, e.g. wrong or missing parameters

• "**5**" for server-side errors

Common error codes used by the API:

• "**400**" Bad Request: Incorrect request, e.g. missing mandatory parameter

• "**404**" Not Found: No matching record was found. If appropriate, the type of missing data record is specified as the detail error code:

• "**500**" Internal Server Error: There is a problem on the server, e.g. missing documents. Here a LeadSuccess administrator should be contacted. The plain text message helps to localize the problem!

Please note that in addition to the application error codes defined here, further error messages of the overlying protocol levels (**ODATA, web server**) can occur.

Version Overview:

Version Manual	Version Database	Date	Changes
0.1	8.2.24	2023-11-02	1st pre-release of LeadSuccess API for Salesforce manual
0.2	8.2.4	2023-11-02	Added some more description about retrieval of subsequently changed leads
0.3	8.7.00	2024-11-22	Added: **LS_LeadReport**, IsReviewed for **LS_Lead**
0.4	8.7.03	2025-04-04	Added technical docu for sample website
			.
			.
			.