<https://www.coursera.org/learn/building-no-code-apps-with-appsheet-foundations/home/welcome>

**Building No-Code Apps with AppSheet**

# Fundamentals of no-code app development

# What is no-code app development?

# It's the process of developing apps without writing any code.

# SaaS-based no-code platforms empower businesses to create software exponentially faster and with less money than a traditional code-based approach.

# No-code development VS low-code development,

# No coding VS Some amount of coding is required.

# Non-technical professionals use it to build applications without writing any code VS Traditional developers use it to simplify and expedite coding.

# Why should you use no-code app development? Benefits are

# No-code platforms provide a faster time to deployment.

# No-code platforms enhanced productivity.

# Less IT backlog

# Improve security

# Cleaner data

# Less paper (paper based systems)

# Greater Mobility

# Improved communication.

# Reduced operational expenditures

# Who uses no-code app development platforms?

# Everyone (technical or non-technical uses).

# 3 types of workers are typically involved in no-code app development.

# 1.Process owners - are non-technical or technical employees, such as project managers and IT developers, who are in a position to spearhead app development.

# 2. Decision makers - are executives who have the vision and power to approve no-code app development.

# 3. End users - are employees who will benefit from using apps on a daily basis.

# What is Google Cloud AppSheet product

* + With AppSheet, anyone can make an app.
  + It can be mobile or desktop-based and can serve many different functions.
  + It’s a no-code platform that enables organizations to develop and launch applications at scale.
  + How AppSheet works.
    - It connects with services across the entire Google ecosystem.

Graphical user interface, application, Word

Description automatically generated

* + - It can be developed to implement various business functions that

Graphical user interface, application

Description automatically generated

* + - 1. Collect data including images, barcodes, and signatures.
      2. Manage and process the data by applying business logic and
      3. Share insights into the data by generating reports, dashboards, and notifications.
    - It can be deployed and used externally or internally in your organization.

Graphical user interface, application

Description automatically generated

* + - It supports a wide **variety of data sources**, including spreadsheets and databases that can be used with your app.
    - It enables you to implement data and integration policies, security and app design, and content authoring, which allows you to establish a set of governance practices through the app development lifecycle.
    - Using AppSheet, process owners in different organizations departments can create apps in a consistent manner following best practices.
    - Users interact with your apps using mobile phones, tablets, or desktops to retrieve and update data.
  + AppSheet capabilities include
    - Build rich feature-filled applications.
    - Various types of views including tables, galleries, charts, and maps.
    - Forums and other user controls to capture user data,
    - Branding, color themes, and localization features.
  + Security features
    - Like secure sign-on via Google, Microsoft, Dropbox, and other providers
    - Users, groups, and sensitive data management
  + Process automation features
    - Create basic automation to send email, SMS, and push notifications
    - Implement advanced automation workflows.
  + AI capabilities included
    - Optical character recognition(OCR)
    - Machine learning.
  + Data source access and integration features
    - Access data sources in Google Workspace, Office 365, Dropbox, and other providers and
    - Integrate with Cloud databases and SaaS providers.
  + Management, governance, and reporting features
    - Manage the app user life cycle
    - Track user activity, usage, and error reporting
* AppSheet UseCases. What can you actually build with no-code platforms?
  + No-code platforms are completely open-ended,  and let you design feature-rich apps from existing data sources.
  + No-code platforms provide versatility to app creators and users.
  + Inventory/Asset Tracker App.
    - Tracking manually using spreadsheets or paper is time-consuming, and error-prone.
    - Build an asset tracker app to track inventory in real-time, and avoid confusion, and unnecessary delays.
  + Customer service.
    - Build a variety of front line and customer Experience apps for product safety alerts,  appointment management, check-in, and check-out.
  + HR management.
    - Build customized apps that are tailored to your HR needs in the areas of document management, event coordination, new hire.
  + Various operational processes are
    - Office automation, customer engagement, and relationship management,
    - field services and logistics, accounting and finance,
    - Project management.
* Getting Started with AppSheet
  + Signing in to appsheet.com/Account/Login
  + Creating your first app
    - Use the AppSheet UI which includes an editor to build and configure your app.
    - There are 3 ways you can build your app using the AppSheet UI.

Graphical user interface, application

Description automatically generated

* + - 1. The 1st option allows you to start by defining your app details and connecting an initial dataset from your provider account.
      2. With the 2nd option, if you have ideas for your app but no existing data, you can use natural language to describe your app functionality and have the data schema and source automatically generated in your connected account.
      3. The 3rd option allows you to start with a sample app. In the AppSheet editor, you have access to 50 or more publicly-available templates that help provide a foundation for common app use cases across a variety of business functions.
  + How to create an app (1st) by connecting to an existing Google Sheet as the data source.
    - Broken down this process into 6 steps.
      1. prepare your data, use spreadsheets and other documents
         * Provide meaningful header columns in bold in the first row.
         * Ensure that the data type format is consistent.

Graphical user interface, application, table

Description automatically generated

* + - 1. Connect your data to AppSheet, one of 3 ways to connect.
         * 1st option, Add “New Data Source” from the My Account tab.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* + - * + 2nd option, Add "New App" from the My App page.

A picture containing shape

Description automatically generated

* + - * + 3rd option, from Google Sheets by clicking "Extensions", "AppSheet", "Create an app".
      1. After connected your data, add capabilities to your app by using the AppSheet editor. These capabilities include
         * interacting with data via the app,
         * performing actions by using app behaviors,
         * viewing data in the app through different types of views,
         * running automated processes in response to events, and much more.
         * AppSheet editor menu items review

Graphical user interface, text, application

Description automatically generated

* + - * + Info - see and manage the technical details of your app.
        + Data - define how AppSheet is connected to your data.
        + UX - create and customize views along with the look and feel of your app.
        + Behavior - create buttons for navigating in the app, initiating external links, and triggering data workflows.
        + Automation - define automated processes to run in response to certain events.
        + Security – define security features for your app such as requiring users to sign in.
        + Intelligence - incorporate machine learning into your app such as sentiment analysis, predictive analytics, and optical character recognition.
        + Users - share your app and manage who has access to it.
        + Manage - app versions, monitor usage and performance, and much more.
        + Save button - Make sure you click it so your edits are saved.
      1. Define how your connected data will be used
         * Go to the Data tab.
         * By default, the editor lists (customer details) the data source or table that was used to create the app.

Graphical user interface, application, Teams

Description automatically generated

* + - * + Click on "New Table", to add additional tables to your app,
        + To access a list of columns that make up the table click on the "Columns" tab.

Table

Description automatically generated

Show? will or not display columns value in the apps

Type specify type of data is in each column.

Pencil icon modify the properties of the column to change how the app interacts with the data that's stored in the column.

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application

Description automatically generated

* + - 1. Create views and customize the app’s look and feel from UX menu
         * Use ready-to-use view templates that you can customize easily.
         * Each page you see in your app is known as a view.
         * AppSheet may suggest views based on the data that is connected to the app.

Graphical user interface, text, application

Description automatically generated

* + - * + Click "New View".

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* + - 1. Create custom actions for your app.
         * From The Behavior tab, set up custom buttons and actions that can be used in a wide variety of ways

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

* + - * + The Workflow and Reports tabs let you set up automation such as sending emails or SMS text when data is added or updated.
        + Workflows are automations that are triggered when certain actions occur,
        + Reports are automations that run on periodic schedules.
        + Finally, if you want users to be able to use your app when it is not connected to the Internet, you can enable offline use in the Offline/Sync section.
    - Testing your app
      1. The simplest and easiest way to test during development is to use the app preview.

Graphical user interface, text, application, chat or text message

Description automatically generated

* + - 1. The preview in the editor sidebar shows what the latest version of your app will look like on a phone oriented vertically.
      2. Test it on your mobile devices. Go to the User's tab and in the user section, type the email addresses for the people you want to share your app with.

Graphical user interface, text, application, email

Description automatically generated

* + - Finally, you can officially deploy it which will activate all the services in your app.
    - Help.appsheet.com/en, community.appsheet.com.
* Using data sources in your app
  + Understand the different types of data sources, grouped into three categories;

Graphical user interface, application, table

Description automatically generated

* + Connect AppSheet to your data in one of three ways.

Graphical user interface, text, application, chat or text message

Description automatically generated

* + - Add a new data source from the My Account app.
  + Configure your app to use the data source, Click New Table from Tables.

Graphical user interface, text, application, chat or text message

Description automatically generated Graphical user interface, text, application, email

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

* + - After the table is added , configure the column structure, data validation, set other column types and properties for use in the app
  + Defining the data structure in your app
    - AppSheet automatically infers your table column structure including the name, type, and other column attributes for use by the app.

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

* + - There are 35 different data types
      1. Enum which allows to select 1 value from the list of allowed values,
      2. EnumList which allows to select 0 or multiple values from the list of allowed values,
      3. address which is used to specify a postal or street addres can be plotted on a map.
      4. Help.appsheet.com/en/articles/1013271-column-types-diving-deeper
    - The following column properties.
      1. Key: you must have a key column  that uniquely identifies a row in the table
      2. Label: Whether the column name is displayed in the app. One image and non-image column can have this setting.  Chooses important column.
      3. Formula: enables you to provide a calculation to be applied to the column data whenever the user changes a value in a form.
      4. Show: allows you to display or hide the column data in the app.
      5. Editable: enable or disable the user from editing a column's value in the app.
      6. Require: enforces a rule that the column must have valid data in all rows.
      7. You can also set additional column properties in the AppSheet editor.
      8. initial value : To provide an initial value for new rows added, set the value.
      9. display name : To display the column header in the app
      10. Description: To provide a description of the column's purpose or role,
      11. Search: To enable this column for indexing in-app search results,
      12. Scan: To enable barcode scanning in the app using the user's device camera,
      13. NFC: To enable near-field communication scanning of NFC tags using the user's device,
      14. PII: To prevent personally identifiable information (PII) data from being retained in system logs,
    - AppSheet will add **virtual columns** automatically to your table structure.

A picture containing table

Description automatically generated

* + - 1. when a table is first added to the app or when the table structure is regenerated.
      2. doesn't correspond to an actual column in the underlying spreadsheet or table
      3. value is automatically computed via an app formula expression;
      4. values are not stored anywhere so they won't show up in your spreadsheet,
      5. they do however affect your app's functionality behaving much like an ordinary column.
      6. These columns are usually added as a convenience based on common use.
      7. Examples of automatically added virtual columns are row number which computes a unique number for each row, computed name which concatenates a First Name and Last Name column.

Graphical user interface

Description automatically generated

* + - 1. A common use case for virtual columns is to combine other columns in the app.
      2. Logo

         Description automatically generated
      3. To implement this, use the app formula, concatenate First Name, Last Name.
      4. Construct complex conditions using app formulas.
    - Images or videos in your app.
      1. To display images in your app, set the image/thumbnail datatype of your columns
      2. To display videos, set the column type as video,

Graphical user interface, application

Description automatically generated

* + - 1. The actual value that is stored in a column is either a publicly accessible URL or a filename or path to the location of the image (must be stored in your Cloud file system not own computer).

Graphical user interface, application, table, Excel

Description automatically generated

* + - 1. If you specify an image file name in your spreadsheet, the file should be in the same folder location as your spreadsheet.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

* + - 1. Image file names can be specified relative to the location of the spreadsheet.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

* + - click "Regenerate structure"
      1. When changes are made to the column structure of the underlying data source, you must regenerate the column structure for your app.
      2. Decide to remove/add/rearrange columns. With these changes made to the table structure, you may see an error in the app when it syncs with a data source.
  + Using more than one table in your app, Reference between tables

Diagram

Description automatically generated Diagram

Description automatically generated

* + - When an app uses multiple tables, create connections (references) between the tables.
    - Reference can be used to
      1. represent relationships.
      2. Easily retrieve information from a related row in another table.
      3. Navigate from one row to another.
      4. Create dynamic dropdowns
      5. Show related lists of items
      6. referenced columns can be
    - To create a reference, you add a column of type ref to a table and specify the name of the sauce or related table that this column refers to.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + - A ref column always stores the key column value of the reference row in the other table.
    - Graphical user interface

      Description automatically generated
    - Whenever a ref column is created either by the system or by you, the app automatically creates a reverse reference virtual list column in the reference table.
    - This reverse reference column is virtual and will not exist in the underlying sheet or table.
    - Graphical user interface, application

      Description automatically generated
    - A reverse reference enables the app to
      1. navigate from one row of a table to all of its related row,
      2. easily display a row of a table, along with all of its related rows in another table.
      3. Compute aggregates like the count of a customer's orders or the total dollar value of a customer's orders.
    - The system added reverse reference column is a list of the key column values of the related rows.
    - Graphical user interface, application

      Description automatically generated
    - For example, the reverse reference column in the customers table for customer and Adams would contain a list of the order IEDs that exists in the related orders table for this customer.
* Defining the UX of your app
  + click, "New View.", to create a new view. The new view form enables you to define the view.
  + “View name” is a unique name,
  + “For this data” - select the table whose data will be used by the view,
  + the type of view, currently supports 11 different types

Graphical user interface

Description automatically generated

|  |  |  |
| --- | --- | --- |
| * The calendar view. * You can display the calendar as a singular or a dashboard view. * Currently supports integration with Google Calendar which you can use as a data source | * Deck View * It also has an option to select a layer of data to display as a secondary view. * This nested view provides a way to visualize lists of related data created by building references between tables. | * The deck and table views summarize the data as a scrollable list of row. |
| * Gallery is the default view of data that has an image or thumbnail image column. * Can control the size of the images displayed in the app.  |  |  | | --- | --- | |  |  | | * Detail view. | * Google Map view * Lat-long data is represented as comma-separated. * The complete address including the city-state if any and country. * For performance reasons, only a limited number of pins can be displayed at the same time. |
| * An onboarding view is a special view that is dedicated to onboarding new users of your app. * The onboarding view is backed by a table that has columns for the image, title and two short descriptions. * The values from those columns are displayed in the view as shown. | * A form view. * AppSheet automatically generates forms for the tables that have been added to your app. * The order of each form field can be customized. | * The card view is a new way to visualize data more easily via cards that dynamically fill the device screen and enable users to quickly view details and initiate related actions. * The card view also includes an intuitive layout method to customize each field that is displayed on the card |
| * The dashboard view allows you to show a multiple views in one place in the app. |  | * The chart view |

* + its position from where the user accesses it in the app and other view options.
  + Each type of view has type specific view.
    - Display options - how the view is presented in the app's main menu and in the navigation bar along the bottom of the app display.
    - Display options include the following properties.
      1. A display name : set the text to refer to the view.  If not set, the view name is used.
      2. Set an expression that evaluates through text to set the views display name.
      3. An icon - displayed with the view's display name. Custom icons are not allowed.
      4. Every view must have an icon. If not is selected, a type-specific default will be used.
      5. Show if is a yes/no expression that if provided determines whether the view is included as a navigation element in its designated position. If the expression evaluates to true, the view will be included.

Graphical user interface, application

Description automatically generated

* + - Each view type also has type-specific behavior options that allow you to tailor the behavior of the view when the user interacts with it.

A picture containing timeline

Description automatically generated

* + Primary views are accessed via the bottom bar of the app and are used most often.
    - Frequently used views that are accessed from the bottom navigation bar in the app.
    - AppSheet automatically generates primary views based on the tables.
    - The UI that AppSheet creates for your app using these views is essentially CRUD based.

Diagram

Description automatically generated with medium confidence

* + System views are automatically generated views for your app.
    - Can configurable. A picture containing graphical user interface

      Description automatically generated

Graphical user interface, text, application

Description automatically generated Graphical user interface, application, table

Description automatically generated

* + - System views include menu views.
      1. Accessed from the top left menu in the app, and generally include views that are only used occasionally by the app user.
    - Reference views, is a view that is only used when drilling down into your data.
      1. When an app users selects or edits a row from a table view, they are not taken to any of the top level views that appear in either the menu or the buttons at the bottom of your app.
      2. Instead, the user is taken to a ref view.
      3. A ref view is a view that is only used when drilling down into data editing or adding data or displaying a row via ref column.
      4. The system automatically generates reference views for each of the tables.
      5. In the example, the Items\_details table has a item details view of type detail. Because the table allows additions and updates, there is also a Items\_Form view of type form.
      6. You can modify the system generated views to easily change the look and feel.
      7. You can also explicitly define ref views by creating a new view and setting its position to ref.
      8. Any user-defined ref view takes precedence over a corresponding system defined ref view.
  + Customizing the UX of your app
    - Configuring the appearance of your app, from the brand tab under the UX section.
      1. Set your app's color theme and primary color.
      2. Themes control the appearance of the header, footer, text, and icons in your app.

Graphical user interface, text

Description automatically generated

* + - 1. The app logo icon - users will click to launch app from the device's home screen.
      2. The launch image is shown when the application is initially launched and when sync is in progress.
      3. The background image forms a backdrop for all pages in the app.
      4. Change the color style of the header and footer

Chart, bubble chart

Description automatically generated

* + Control the display of data in your app from the format rules tab under the UX section.
    - To implement this, create format rules for the data tables used by your app.
    - specify the Rule name, the data table “For this data” to which this rule applies, an optional “If this condition is true” that must be satisfied for the format rule to take effect.
    - “Format these columns and actions” - A list of table columns to which the rule is applied, a set of visual and textual format settings that include an icon to show alongside the rows of data that match the rule condition.
    - Text and highlight color, font size and text attributes like bold italic, etc.
    - Graphical user interface, text, application, email

      Description automatically generatedChart

      Description automatically generated with low confidence
  + Tailoring the app’s user experience from The options tab under the UX section
    - The options on this tab apply to the entire app.
    - The starting view option, identifies the view that the user sees when the app is launched after installation.
    - If the start with about option is enabled, the app's about page is displayed when the app is launched.
    - The show name and email in the side menu option, allows you to display the apps user's name and email address in the main menu in the app, if the app requires sign-in.
    - Other options in this tab like disabling app sharing, setting forms styles that control how user input forms are displayed, and image upload sizes to name a few.

Graphical user interface, application, Teams

Description automatically generated Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated Graphical user interface, text, application

Description automatically generated

* + Changing system buttons and notifications when app viewed from The localize tab under the UX
    - It’s usually sufficient to address many of the app's localization requirements.
    - Can configure locale support including locale currency using the AppSheet supported Cloud providers locale settings.
    - <https://help.appsheet.com/en/articles/961395-locale-support-in-appsheet>

Table

Description automatically generated

* + Customize the app's about page.
    - Every app should have a page that describes the application.
    - To customize the About page, from the properties tab under info in the navigation menu.
    - Specify various properties of the app that include a unique short name or handle for the app, the version of the app, a short one-line description of the app.
    - Under the information for app user's section, specify user guidance, a how-to information for users of the app and about URL to a website that provides more information about the app, the app's privacy policy and the terms and conditions for users of the app.
    - The about page is accessed from the main menu in the app.
    - It can be configurable

Graphical user interface, text, application, email

Description automatically generated Graphical user interface

Description automatically generated Graphical user interface, text, application

Description automatically generated

* App Publshing
  + App Deployment Process, From the deploy tap under the manage section of the navigation menu
    - 1. Run a deployment check to look for any errors or warnings reported for your app before publishing it.

Graphical user interface, text, application, email

Description automatically generated

* + - * To run a check, to see the stage of app's deployment.
      * The app is in the prototype stage, if it is not deployed, click on run deployment, check to run the deployment check.
    - 2. Fix all reported errors and resolve any warnings,

Graphical user interface, text, application

Description automatically generated

* + - * The output of the deployment check contains the status of warning or error.

Graphical user interface, text, application, email

Description automatically generated

* + - * <https://www.appsheet.com/support/errormessages#ERROR_PLAN_CHECK>

Text

Description automatically generated with medium confidence

* + - * After you've fixed any issues, you should re run the deployment check.
      * If no more issues are reported, you can move the app to the deployed state.
    - 3. Deploy the app

Graphical user interface, text, application

Description automatically generated

* + - * When you deploy your app, app changes the state to deployed.

Graphical user interface, text, application, email

Description automatically generated

* + - * You have the option to return it to its prototype state if further changes to the app are required, you can also undeploy your app.
      * If you want to underplay and permanently remove an app, you can simply delete it.
      * To delete an app, navigate to the my apps page in the app sheet UI.

Graphical user interface, application

Description automatically generated

* + - * To remove an app user navigate to the users page in the app

Graphical user interface, text, application, chat or text message

Description automatically generated

* + - * Deployment process, 2 different ways to deploy and publish your app,
      * 1. Instant deployment
        + If you're building an app for your team or employees.
        + With instant deployment, apps that you build are hosted by the AppSheet Hosting app installed on your mobile device.
        + When you deploy and run your app, it will appear to run on its own, but it's actually hosted by the AppSheet Hosting app.
        + This is similar to a webpage that is hosted in a web browser.
        + Publishing an AppSheet app is as simple as clicking a link in an email

Graphical user interface, text, application

Description automatically generated

* + - * + Other place to get the install link

Graphical user interface, text, application

Description automatically generated

* + - * + Send an app installing link by email.
        + When a user clicks the Install Link on their mobile device, the user is first asked to install the AppSheet Hosting app on their device.
        + If already present, Clicking on the app icon launches the AppSheet app.
        + The benefits of instant deployment

it's easy and simple to implement and use,

Changes to the app are immediately available to the app user.

* + - * 2. traditional deployment.
        + You submit the app separately to an app store, iTunes store for iOS, and Google Play Store for Android.
        + Consider this model if the app store discovery and distribution model is important to you, and you are aiming for large scale consumer adoption of your app.
        + 1st test your app using instant deployment before the traditional deployment.
        + The traditional deployment process involves these steps.

1. create the app package or white-label version of your app.

From the deploy tab under the manage section.

Graphical user interface, text, application, email

Description automatically generated

2. submit the app package to the appropriate app store.

3. go through the app approval process mandated by the app Store.

* + - * + It create standalone native versions of your AppSheet app for Android and iOS.
        + So your apps can be distributed as a direct download from the Android or iOS app stores without the need to download the AppSheet Hosting app.
        + Whether or not you make use of this deployment path, you can also enable the white-label option in order to remove AppSheet branding and better establish your own brand within the app.
        + Graphical user interface, text, application, email

          Description automatically generated
  + 4. test the app as an end user.
    - Good idea to test your app on your own device before sharing it with your users.
    - The easiest way to test your app on a device is to first install the app sheet hosting app on your device.
    - Graphical user interface, text, application

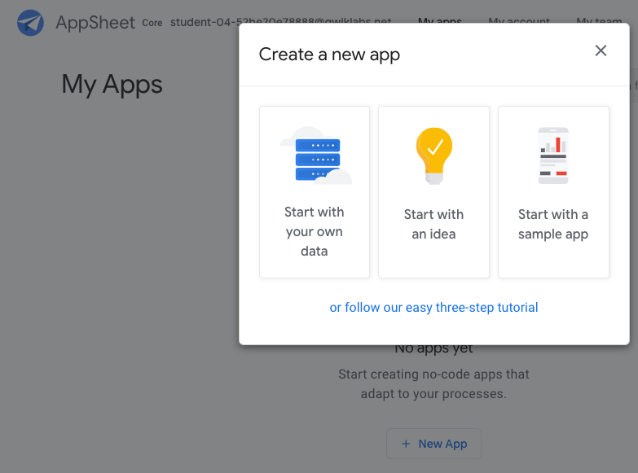
      Description automatically generated
    - Sign in, then it starts an app gallery, which itself is an app created by app sheet.
    - The app gallery lets you browse and run your own apps and apps that have been shared with you on your mobile device.
    - To test the app on your device, tap and run it from the app gallery, verify that the app functions as expected on your device.
    - There are three primary views in the app gallery,
      * Recent - which displays the apps most recently accessed by the user sorted into categories of today this week and earlier,
      * Shared with me - Which displays apps that the user has been invited to use either via direct email invitation or via domain permission and
      * Created by me - which displays apps the user has created. This view is only visible if the user has created an application.
  + Sharing your app with your end-users – the link through Email, Website, SMS.
    - From the Users tab under the Users section, add user to share your app
    - By clicking the Share icon in the top bar of the AppSheet UI.
    - Graphical user interface, text, application, chat or text message

      Description automatically generated
    - Finally you can also choose to share your app using the in-app share button
    - The share button can be disabled by users option from the Options tab
    - Graphical user interface, text, application

      Description automatically generated
    - how to install the app on an Android device using app shortcuts.
    - On Android devices, users can install app shortcuts from AppSheet's app gallery without needing to use the installation link.
    - To use this, the app user should download the AppSheet hosting app from the app store.
    - Graphical user interface, text, application

      Description automatically generated

# DEM0 1 Lab 1 Creating a Basic App sing AppSheet

* 1. You are now signed in to AppSheet.
  2. 
  3. Click on the X dialog to view the AppSheet MyApps page.
  4. The MyApps page be empty since you do not have any apps yet.

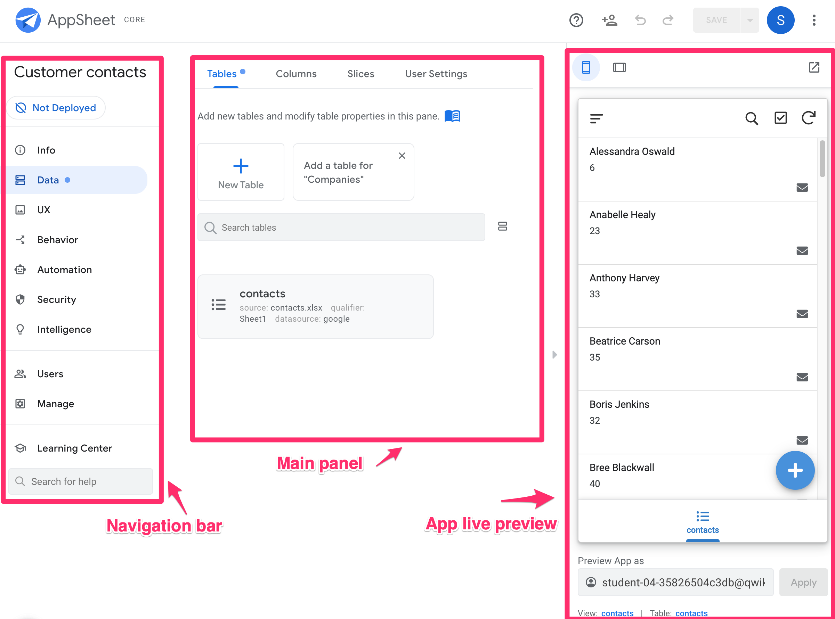
## Task 1. Create your first AppSheet app

* 1. There are 3 ways you can build your app using AppSheet:
     1. Connect to a data source.
     2. Develop an app from an idea.
     3. Use a template.

### **Create the app**

* + 1. On the **My Apps** page in the AppSheet UI, click **New App** to create an app.
    2. In the **Create a new app** dialog, click **Start with your own data**.
    3. In the same dialog for **App name**, type **Customer Contacts** to name your app.
    4. For **Category**, select **Customer Engagement** from the list.
  1. **Select data for your app**
     1. In the same dialog, click **Choose your data**.
     2. In the file picker, select the contacts spreadsheet from Google Drive.
  2. Click **Start customizing**, or close the **Welcome to your app** pop-up.
     1. You are now in the AppSheet editor UI. The UI also provides a live preview of the app running in an emulator, which you can use to test the app's functions.

## Task 2. Preview the app

* 1. AppSheet has created your new app and connected it to the contacts spreadsheet that provides data for the app. The contacts app is a live, functional prototype that is connected to the contacts sheet on Google Drive.
  2. 
  3. The AppSheet editor consists of 3 main areas:
     1. **Navigation bar:** Implement the app's functionality, including its data source connections and schema, the user experience (UX), and other behaviors. You will explore some of these topics in subsequent labs of this course.
     2. **Main panel:** Configure settings for the app to implement its functionality, based on the option you select in the left navigation bar.
     3. **App live preview:** View and test changes to your app's functionality as you design and implement features in your app.

### **View contact information**

* + 1. In the app live preview, scroll through the list of contacts that are displayed in the app.
    2. AppSheet automatically sorts the data by first and last name.
    3. Click the name of any contact to view of that contact's.

### **Modify contact information,** Click **Edit** pencil to edit and click **Save**.

* + 1. Verify that the updated info on the contact details page in the app.
    2. Click back arrow to return to the contacts list page.

### **Add a new contact,** click **Add** plus to add a new contact.

* + 1. In the contact form, specify the following: ID, Email, Name, Phone. Click **Save**.
  1. On the contacts list page in the app, click **Search** search.
* Task 3. Synchronize app data
  1. All the changes you made in the previous task are local within the app and are automatically synced to the underlying data source used by the app.
  2. If not sync, Click **Sync** sync to synchronize the changes to the Google sheet on Google Drive.

### **Verify the data changes,** navigate to [https://drive.google.com](https://drive.google.com/). Open the **contacts** sheet.

# DEMO 2 Configuring Data for your App using AppSheet

# Task 1. Create the app

<https://www.appsheet.com/Template/AppDef?appName=Lab1-CustomerContacts-3856613&copy=1>

1. On the **Clone your App** form, specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **App name** | Customer Contacts |

1. Click **Start Customizing** to go to the AppSheet editor.

You can also access the app from the **My apps** page in the AppSheet UI under **Prototype Apps**.

## Task 2. Set up your app with a second data source

Our Customer contacts app currently uses a Google sheet containing sample person contact information.

In this task, you add a second Google sheet that contains information about companies to the app.

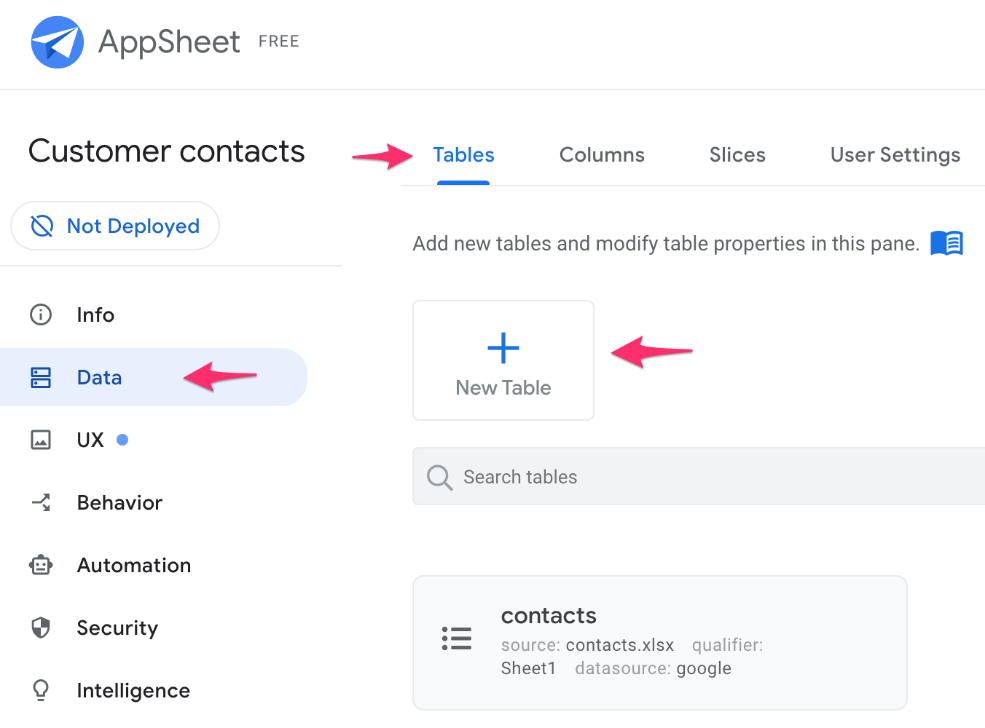
### **Select data for your app**

AppSheet refers to data files used in your app as tables. A table is a description of the rows and columns in your spreadsheet. While all of the data is stored in your spreadsheet, this description becomes part of your app definition.

Adding a table to your app is usually one of the first steps involved in creating an app.

To add additional tables after you have created your app, perform the following steps:

1. In the AppSheet UI, use the left navigation bar and go to **Data > Tables**. The previously added contacts table is listed.

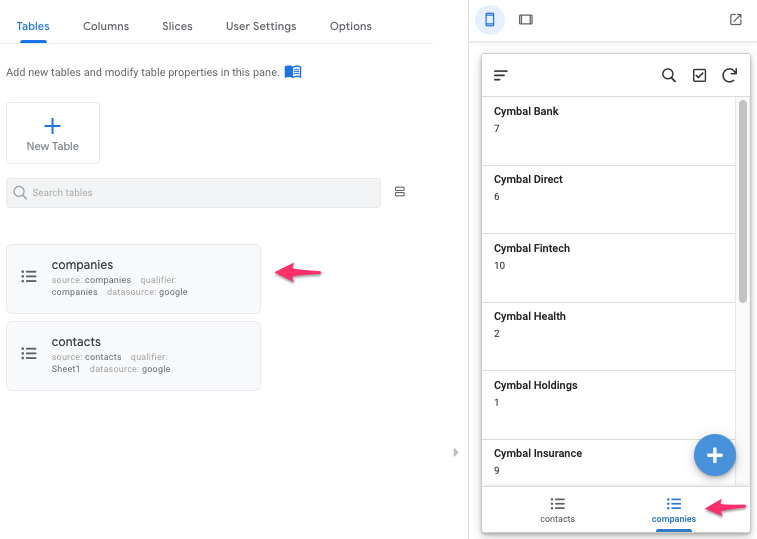


1. To add a second data source to your app, click **New Table**.
2. In the **Get data from** form, click **Sheets on Google Drive**.
3. In the file picker, select the companies sheet from the **My Drive** folder, and click **Select**. This sheet was automatically provisioned for the lab.
4. In the **Create a new table** form, leave the default settings and click **Add this table**.

The settings in this dialog allow you to select a specific worksheet from the spreadsheet or a different type of data source; and lets you allow or disallow modifications to the data by users of the app.

1. AppSheet adds the companies table to your app. This table contains information about various sample companies.

Preview the data from the new table using the app live preview feature in AppSheet.



## Task 3. Configure your app's data structure

When a data source is added as a table for your app, AppSheet reads each column header to define the column structure of the app. You need a column header in your data source for each column in which you store data.

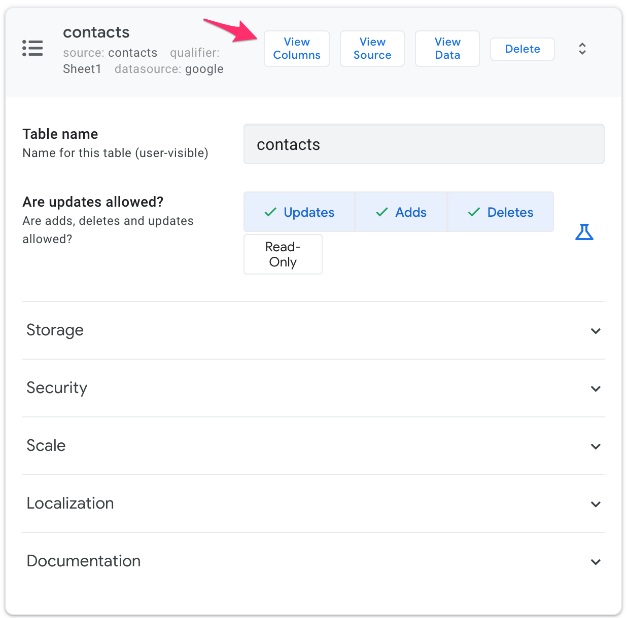
In this task, you inspect and if needed update the columns' type and properties (the default structure) that AppSheet has inferred and set for the app.

### **Update column structure for: contacts**

1. In the AppSheet editor left navigation menu, select **Data** and then click the **Columns** tab at the top.

The columns tab displays an expanded view of the tables that have been added to the app.

1. To inspect the structure of the columns in the contacts table, click the contacts panel to view its configuration.
2. Click **View Columns** to view the column structure of the table.



1. The table below lists the columns and some of their properties as inferred by AppSheet.

Determine if you need to change the type or other properties of each column (Scroll to the right in the AppSheet editor to view all the column properties).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Type** | **Key?** | **Label?** | **Formula** | **Show?** | **Editable?** | **Require?** |
| \_RowNumber(System generated virtual column) | Number |  |  |  |  |  |  |
| ID | Number | (checked) |  |  | checked | checked | checked |
| Email Address | Email |  |  |  | checked | checked | checked |
| First Name | Name |  |  |  | checked | checked |  |
| Last Name | Name |  |  |  | checked | checked |  |
| Phone | Text |  |  |  | checked | checked |  |
| \_ComputedName(System generated virtual column) | Name |  | (checked) | CONCATENATE([First Name]," ",[Last Name]) | checked |  |  |

Which of the above columns' properties do you think needs to be updated?

This generally depends on your app's requirements. For the purposes of this lab, the highlighted columns in the table above are candidates whose properties should be changed.

1. Given that this is a person contact app, there must be valid data in all cells of the First Name column, so the **Require?** property for this column should be checked.
2. Notice that AppSheet has set the type property of the **Phone** column as **text**.

This allows the column to contain a single line of text. Since this column is intended to store a valid phone number, change the type to **Phone** using the drop-down list.

1. Click **Save** to save your column configuration changes.

### **Update column structure for: companies**

Follow the same process to update the structure of the columns of the companies table, where required.

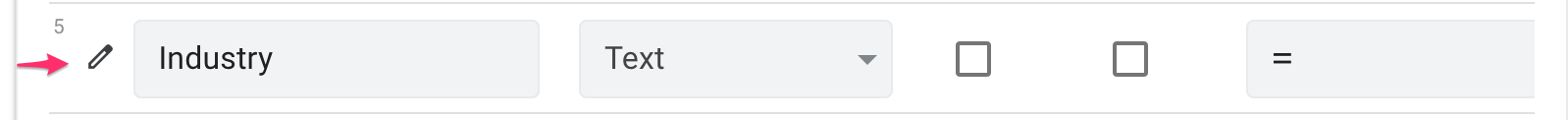
1. Perform this step by referring to the highlighted items in the table below as a guide to update the column properties:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Type** | **Key?** | **Label?** | **Formula** | **Show?** | **Editable?** | **Require?** |
| \_RowNumber(System generated virtual column) | Number |  |  |  |  |  |  |
| ID | Number | (checked) |  |  | checked | checked | checked |
| Phone | Text |  |  |  | checked | checked |  |
| Company Name | Name |  |  |  | checked | checked |  |
| Industry | Text |  |  |  | checked | checked |  |
| Business Address | Address |  |  |  | checked | checked |  |
| Shipping Information | LongText |  | (checked) |  | checked | checked |  |

1. Repeat the previous step for each of the highlighted columns in the table above, except the Industry column which is updated in the next step.
2. AppSheet supports the Enumerated Type for columns. Columns of these types are constrained to having one or more allowed values from a fixed list.

In this step, you change the **Industry** column's **type** property to use the Enum (single value) or EnumList (multiple values) type.

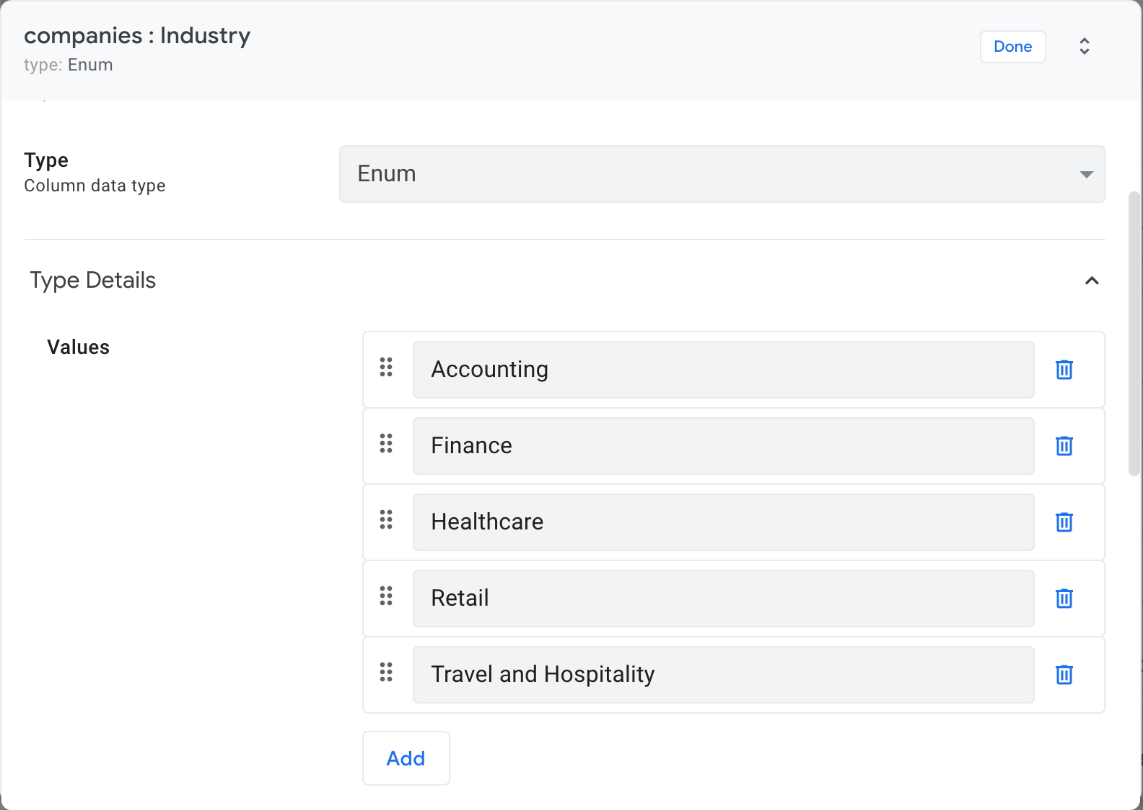
Click the pencil icon to the left of the Industry column.



On the column details form, specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**(type or select) |
| Type | Enum |
| Type Details  Values | Accounting  Finance  Healthcare  Retail  Travel and Hospitality |

Click **Add** for each enum value to be added to the list of allowed values.



**Note for information only:**

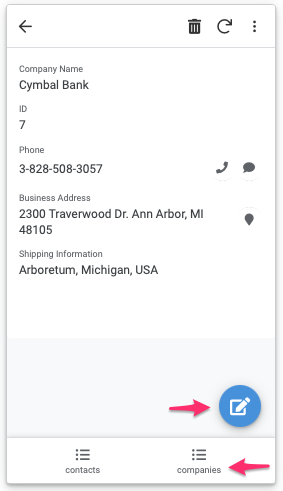
The **Allow other values** option enable users to enter any value they wish in addition to the allowed values displayed in the dropdown list.

The **Auto-complete other values** option makes it easier for the user to choose from the set of previously entered values and helps to ensure that all entries are submitted in the same way avoiding typos.

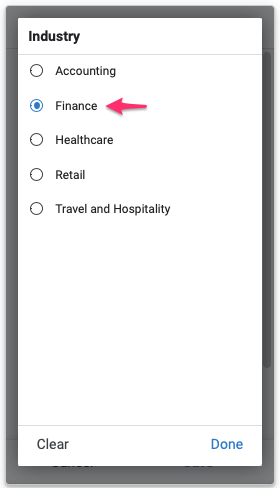
By setting the **Input mode**, you can control whether the values are displayed as buttons arranged naturally or as a vertical stack, or as a dropdown set of radio buttons in the app.

1. After all the values are entered, click **Done** in the form.
2. Click **Save** to save the app configuration changes.
3. Preview the change in the live app preview:

Select the companies table, and click on any one of the companies from the list. Click the pencil pencil icon to edit the company record.



Scroll down to select the **Industry** column using the dropdown. The **enum** values are displayed. Select any of the values.



Click **Save** to save the data changes.

Refer to the documentation for more information on [AppSheet column types](https://help.appsheet.com/en/articles/1013271-column-types-diving-deeper).

## Task 4. Regenerate your app's data structure

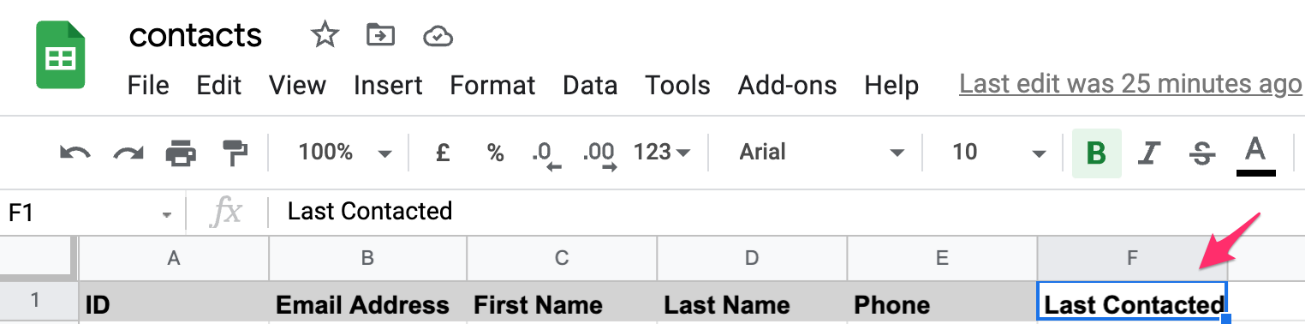
AppSheet reads column metadata from your data source to define the column structure of the app. For spreadsheets, AppSheet uses the column headers to derive this information. Every time you modify the columns in the spreadsheet, you need to regenerate the column structure within the app, or AppSheet won't know how to locate the columns to read and write data and your app will stop functioning.

In this task, you regenerate your app's data structure after adding a new column to the sheet on Google drive.

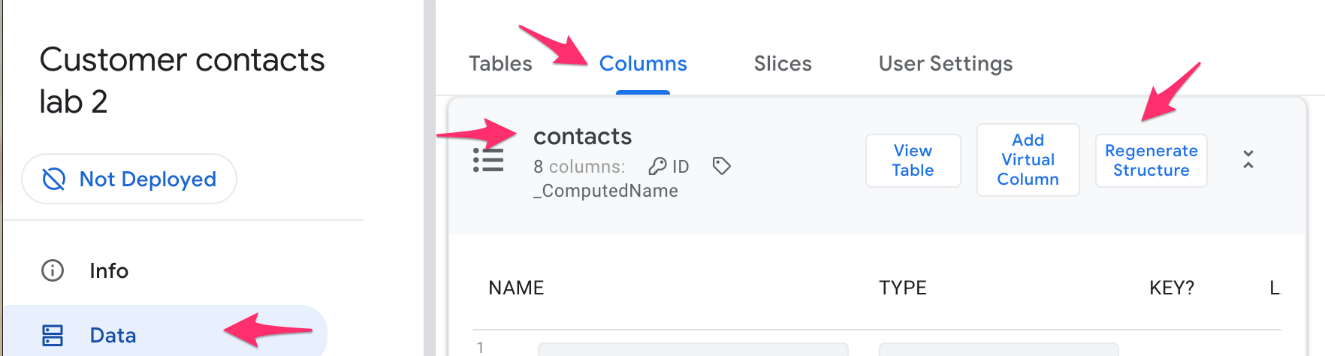
### **Add a new column to the contacts sheet**

1. Navigate to the **Data > Tables** in the AppSheet editor and expand the contacts table definition.
2. Click **View Source**. AppSheet opens the contacts sheet from Google Drive in a separate browser tab.
3. Add a new column header to the contacts sheet in row 1, column F with a value of: **Last Contacted**.

We will use this new column to store the last contacted date and time when the person was contacted.



1. Switch to the AppSheet editor in your browser and navigate to **Data > Columns**.
2. Expand the contacts table if not already expanded.
3. Click **Regenerate Structure**.



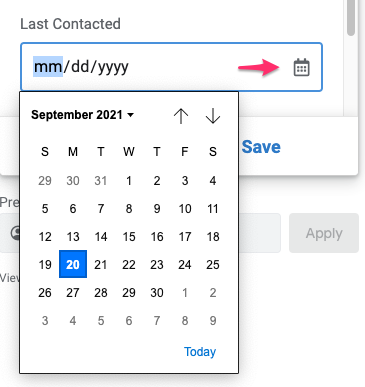
1. Click **Are you sure?**

AppSheet regenerates the column structure for the contacts table, and re-syncs the app in the live preview.

1. AppSheet also infers the type of the new column which may not be the intended column type. Using the **Type** dropdown list, change the type of the **Last Contacted** column to **Date**.
2. Since it is not mandatory for every contact to have a value for this column, scroll the column attributes to the right and verify that the **Require?** attribute is unchecked. If it is checked, then select it to uncheck the attribute.
3. Click **Save** to save your app configuration changes.

### **Preview the new column in the app**

1. In the app live preview, select one of the contacts from the list.
2. Click the pencil pencil icon to edit the contact.
3. Scroll to the bottom and click the calendar icon to set the Last Contacted value for the contact.



**Note**

In some browsers, the calendar widget may not open. If this occurs, enter the date manually in the edit box.

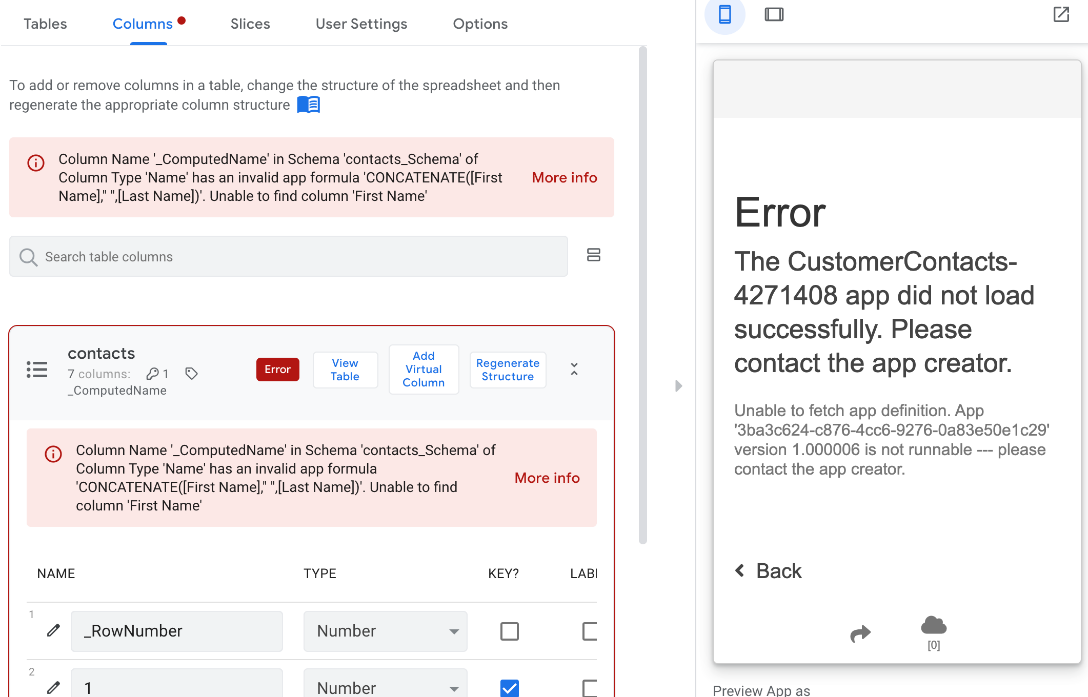
1. Click **Save** in the app preview to save the data for this contact.
2. View the contacts sheet on Google drive to verify that the contact that was updated using the app reflects the updated value in the Last Contacted column.

### **Simulate column headers error**

AppSheet automatically extracts structure from a spreadsheet so that your app can be generated. Errors with spreadsheet structure typically occur when AppSheet is unable to properly identify the column headers.

1. Simulate this condition in the contacts sheet by deleting the first header row from the sheet on Google Drive.
2. Navigate back to the **Data > Columns** tab in the AppSheet editor, and expand the contacts table definition.
3. Click **Regenerate structure**, then **Are you sure?** to confirm.

AppSheet reports the error as shown below:



1. To resolve the error, undo the deletion of the header row in the sheet by clicking the undo arrow.
2. As in the earlier step, regenerate the data structure by clicking **Regenerate structure**, then **Are you sure?** to confirm.

The error should now be resolved.

## Task 5. Create relationships between tables

The person contacts in the contacts sheet are likely employed at the companies whose information is stored in the companies sheet.

This implies that there is a relationship between the two tables. Appsheet allows you to define references between rows in related tables using a special column type called **Ref**.

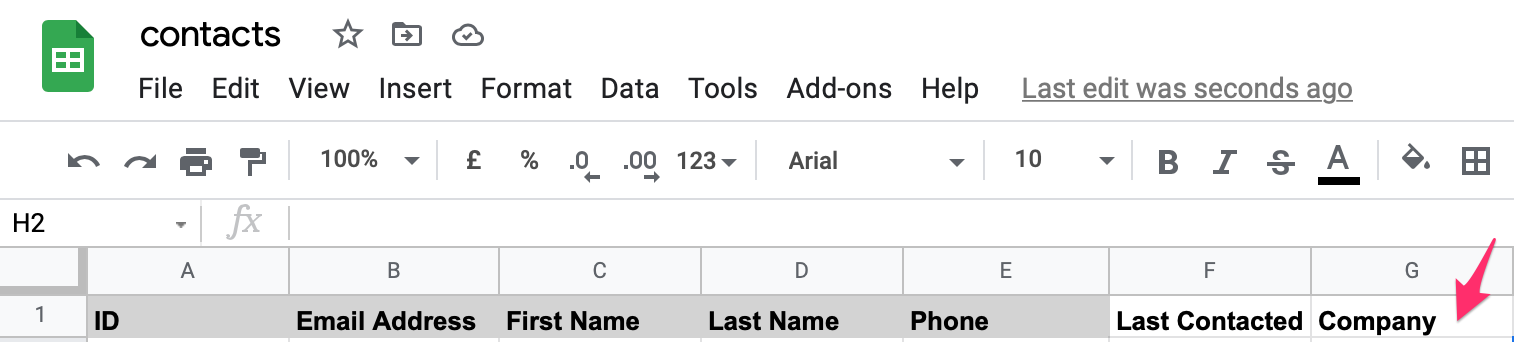
In this task, you use the **Ref** column type to define the relationship between rows in the two tables.

Read the AppSheet documentation to learn more about the [Ref column type](https://help.appsheet.com/en/articles/961426-references-between-tables).

### **Set up the reference column in the contacts sheet**

1. In the contacts sheet on Google drive add a new column header in row 1, column G with a value of: **Company**.

This new column will be used to store the ID of the company that the contact is associated with.

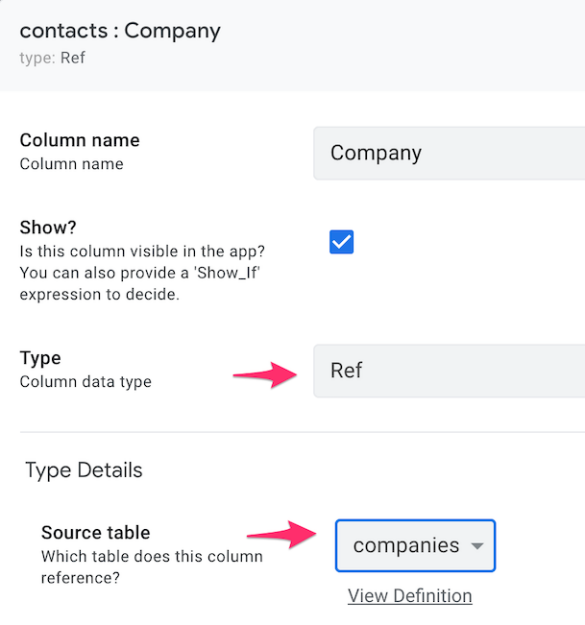


1. Switch to the AppSheet editor in your browser and navigate to **Data > Columns**.
2. Expand the contacts table if not already expanded.
3. Click **Regenerate Structure**, then click **Are you sure?**

AppSheet regenerates the column structure of the contacts table and adds the new Company column to the list of columns.

1. Click the pencil icon to the left of the Company column to edit its properties.
2. On the column details form, specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**(type or select) |
| Type | Ref |
| Source table | companies |

1. This changes the type of the column to be a reference that refers to the companies table.
2. 
3. When contacts are added or updated in the app to include the company that is associated with the contact, AppSheet automatically stores the Company ID in this column in the contacts sheet.
4. Click **Done** to save the changes to the column properties.
5. Finally, click **Save** in the AppSheet editor to save the changes and refresh the app in the live preview.

### **Preview the Company column in the app**

1. In the app live preview, select one of the contacts from the contacts list.
2. Click the pencil pencil icon to edit the contact.
3. Scroll to the bottom and select a Company from the drop-down list of companies. AppSheet has automatically populated the list from the companies table.
4. Click **Save** in the app preview to save the data for this contact.
5. View the contacts sheet on Google drive to verify that the contact that was updated using the app, contains the ID of the company that was selected.

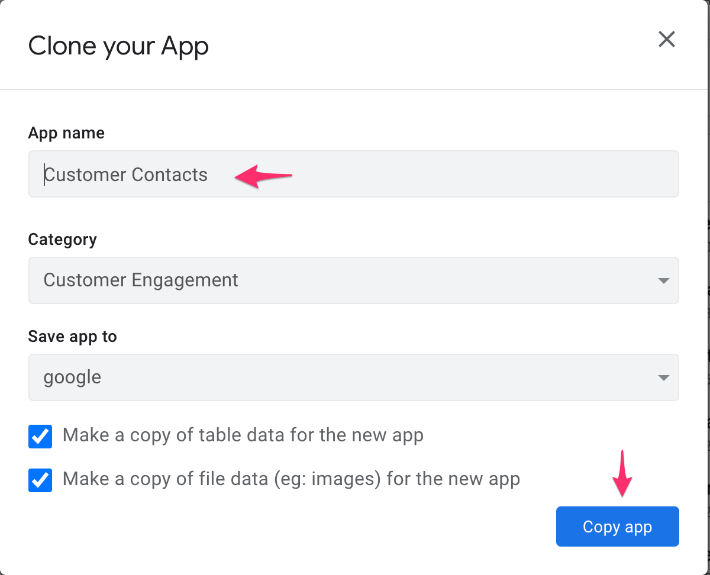
**Demo 3 Implementing the User Experience for your App using AppSheet**

## Task 1. Create the app

1. Click the link to copy the Customer Contacts app to your AppSheet account:

<https://www.appsheet.com/Template/AppDef?appName=Lab2-CustomerContacts-3856613&copy=1>

1. On the **Clone your App** form, for **App name**, type **Customer Contacts** and leave the remaining settings as their defaults.



1. Click **Copy app**.
2. Click **Start Customizing** to go to the AppSheet editor.

## Task 2. Customize the contacts primary view

1. Navigate to the **UX > Views** tab in the AppSheet editor.
2. Under **Primary Views**, click on the contacts view to expand its configuration.

Note that its view type is set to **card**.

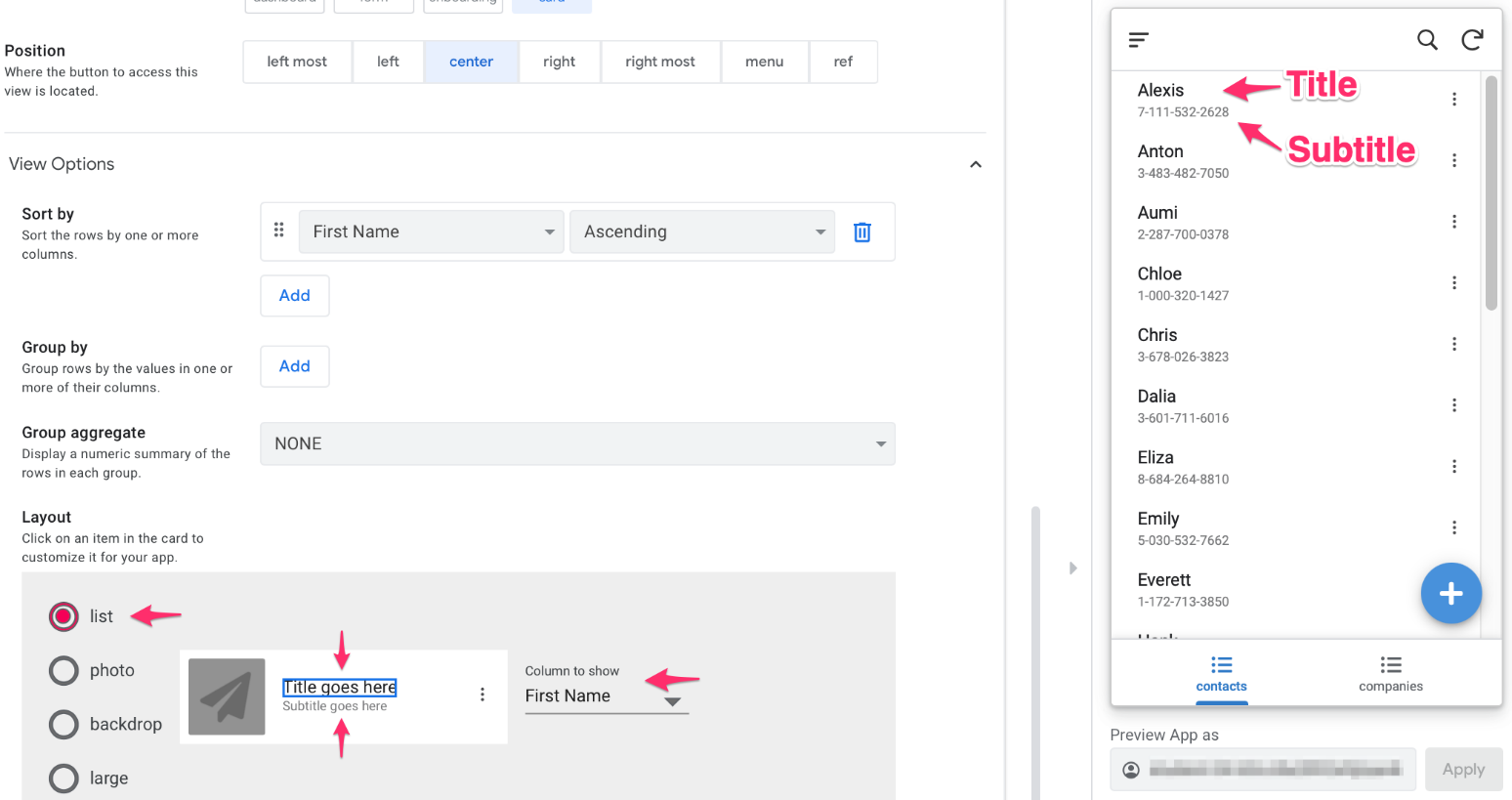
1. Expand **View Options** if not already expanded.
2. Note that the sort order is by the First Name column in the contacts table. Scroll to the **Layout** section. The **list** option is selected by default.

The layout controls how the different items on the card are laid out in the view.

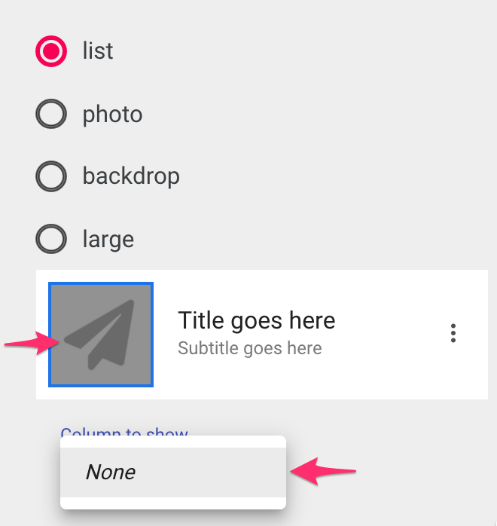
Click **Title goes here**. The **Column to show** indicates the name of the column whose value is displayed as the title of the card.

In this case, verify that it is the First Name column from the contacts table.

1. Similarly, click **Subtitle goes here**. Verify that the **Column to show** indicates the Phone column whose value is displayed as the subtitle on the card.



1. Click the airplane icon to set the photo column.



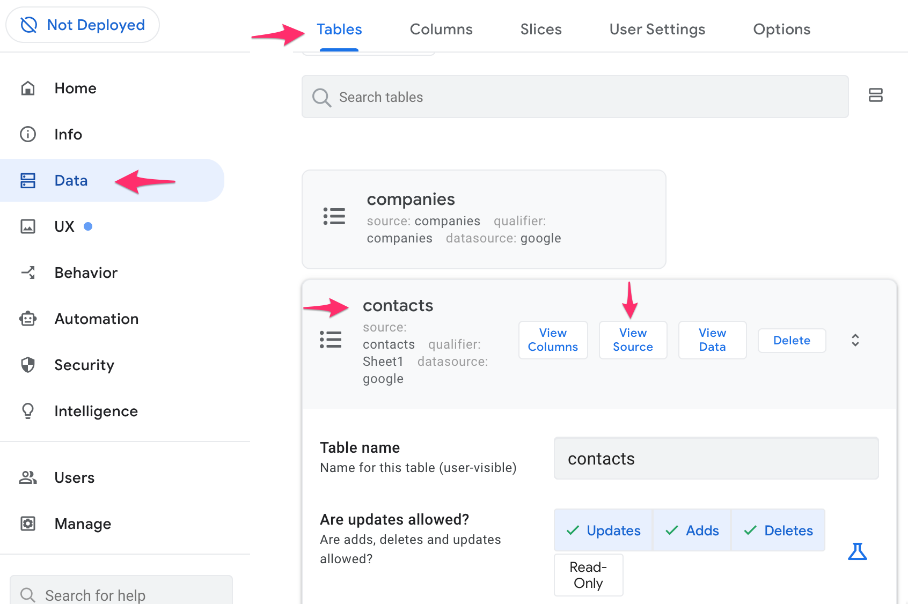
Note that AppSheet could not identify a column in the contacts table that contains image data. This is because the Google sheet used by the table does not contain such a column.

### **Use a photo column in the list layout**

1. Lets add a column to our contacts Google sheet to store the photo of a contact. To do this, navigate to the **Data > Tables** tab in the AppSheet editor.

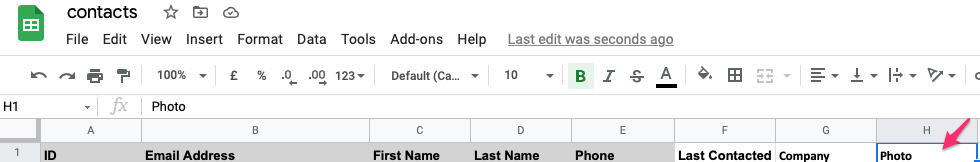
Click **contacts** to expand the table configuration.

1. Click **View Source**.



AppSheet opens the data source, the contacts Google sheet in a separate window or tab in the browser.

1. Add the new column header: **Photo** in column H of the contacts sheet.



This column will contain a publicly accessible URL to the photo image of the contact person.

**Note**

This column can also contain a folder path and name of a file on Google drive that contains the image. The file must be located in the same or relative path location as the spreadsheet and must be publicly accessible.

1. Switch to the AppSheet editor in the other browser tab and navigate to the **Data > Columns** tab. Expand the contacts table column configuration.
2. Since the photo column was added to the Google sheet, we need to let AppSheet know of the new column structure of the contacts table. Click **Regenerate Structure**.

Click **Are you sure?** to confirm.

AppSheet automatically infers the correct column type of Image for the new photo column.

### **Store the image data**

You now have a place to store image URLs or image filenames in the table that is used by your app.

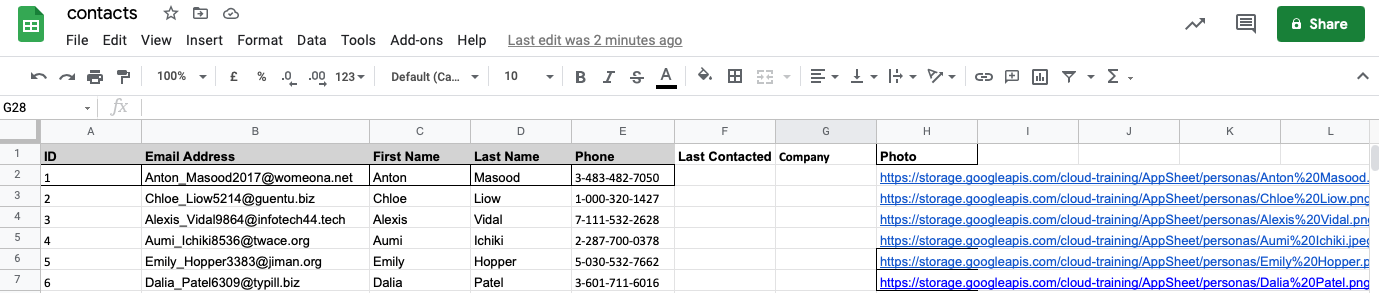
In this step, you provide the app with access to the image files using publicly hosted URLs.

1. Open the link below in a separate browser tab:

<https://docs.google.com/spreadsheets/d/1eBLeNWgMgBGhmtYBQXOUD68tzGWRfX7tDiCVWgQCzrs/edit?usp=sharing>

1. For some rows in the contacts sheet, update the **Photo** column with the URLs to the images.

Copy the image URLs from the Photo column in the Photos sheet opened in the previous step and paste them in the corresponding rows with the matching **ID** values in the contacts sheet.



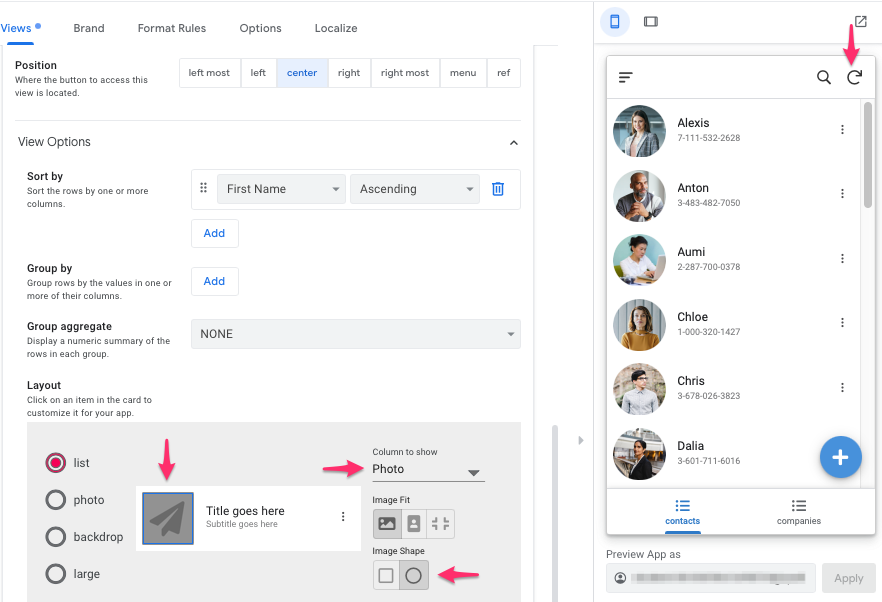
**Note**

In the lab environment not all the contacts will have photo image urls and some images may be duplicated.

### **Update the contacts list layout to use the photo column**

1. Navigate back to the **UX > Views** tab and scroll to the **Layout** section of the contacts view configuration.
2. With the **list** layout selected, click the airplane icon.
3. For **Column to show**, select **Photo** from the drop-down list.
4. For **Image shape**, select the circle icon.
5. Click **Save** to save your changes to the app configuration.

AppSheet updates the app in the live preview to show the photos of the contacts whose images were updated in the Google sheet.



1. If the live app preview does not update automatically, you may need to sync the app by clicking on the sync icon.

## Task 3. Customize the companies primary view

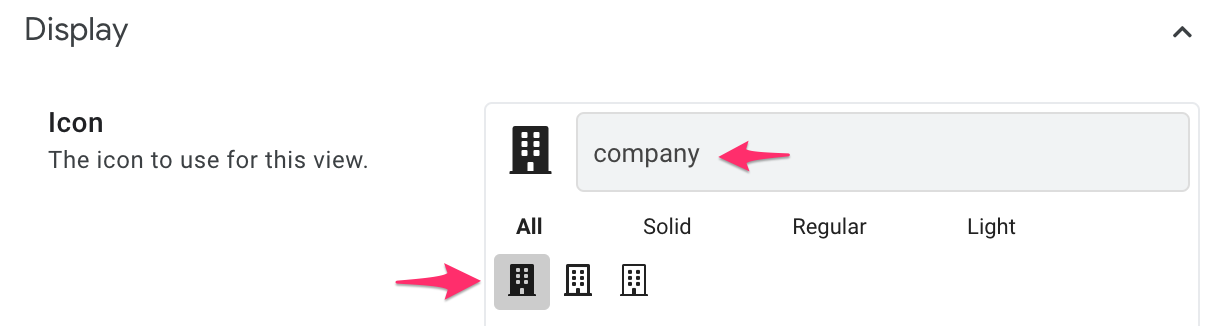
### **Change the companies view type and layout**

1. In the AppSheet editor, navigate to **UX > Views** and select the companies primary view to expand its configuration.
2. Under **View options**, change the view type to **card** by selecting it.
3. Scroll to the **Layout** section under **View Options**, and select **photo**.
4. Select the small airplane icon. For **Column to show**, select **Business Address**.
5. Select the large airplane icon. For **Column to show**, select **Business Address**.
6. Click **Save**.

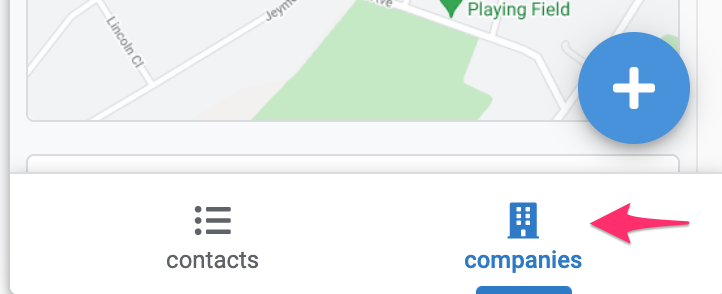
The app now displays a map of the business address location for each company.

### **Change the companies view icon**

1. In the same form, scroll down and expand the **Display** options.
2. In the **Icon** search box, type **company**, and select any of the icons from the icons search results list.



1. Click **Save**.
2. The companies view icon in the app's bottom navigation bar now displays the new icon.



## Task 4. Using the Map view

The map view is a great way to display locations within your app. Depending on your needs you can display location data, drop pins, or provide directions with your app.

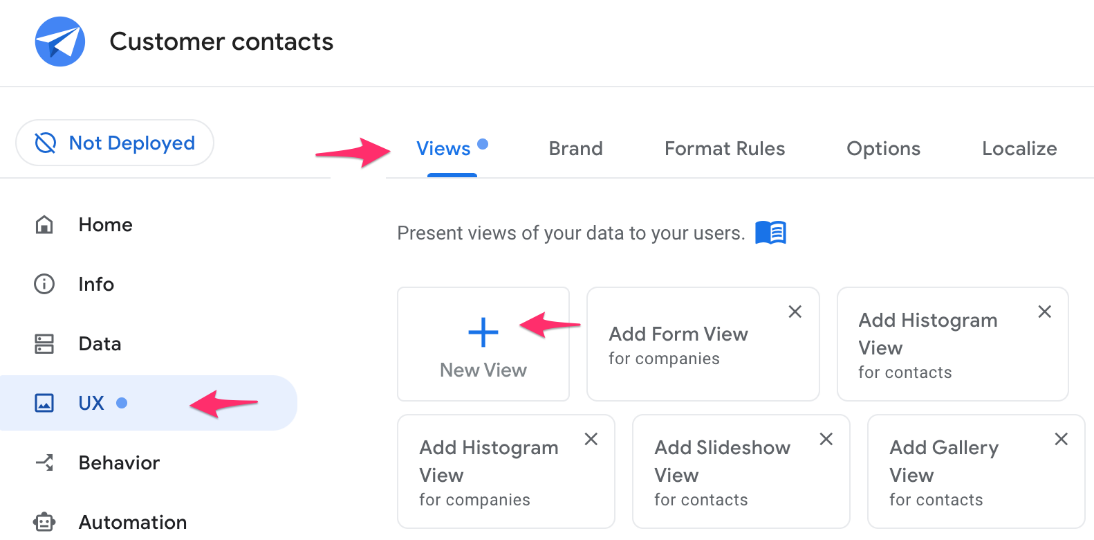
Location data points can come from addresses, XY (coordinates of a point) or LatLong (GPS) information.

You can also use either Google maps to display your mapped locations using addresses and LatLong information, or a custom image to map XY points.

In this task, you use a Map view to display the business address location of companies on a map in the app.

### **Create a Map view**

1. Navigate to **UX > Views** in the AppSheet editor.
2. To add a new view in the Customer contacts app, click **New View**.

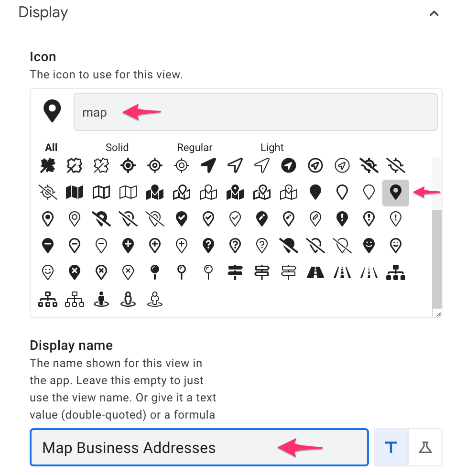


1. In the **New View** form, specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value** (type or select) |
| **View name** | Business Addresses |
| **For this data** | companies |
| **View type** | map |
| **Position** | menu |
| **View options**  **Map column** | Business Address |

1. This view will map data from the Business Addresses column in the companies table.
2. The user will be able to access this view from the app's main menu.
3. In the same form, expand the **Display** options.
4. In the **Icon** search box, type **map**, and select any of the map icons from the list.
5. For **Display name**, type **Map Business Addresses**.

This will be the name displayed for this item in the main menu.



1. Click **Save** to save the new view.

### **Preview the map view**

1. To test and preview the new map view, in the live app preview click the app's main menu icon to display the menu.
2. Verify that the **Business Addresses** menu item is included on the main menu.
3. Select the **Business Addresses** menu item from the main menu to map all the business addresses from the companies table.
4. Select any of the pinned locations in the map. AppSheet automatically displays details about the company at that location.

## Task 5. Using forms in your app

Apps usually need input from the user at some point as the user interacts with the app. To collect user input, AppSheet supports the **Form** view type.

In this task, you use a form view in the app to enable the user to enter product information in order to ship products to a company.

### **Add the shipTo data source to your app**

The form view must have a place to store the user entered data from the app. Before you create the view, you must create or use a table to store this information.

1. To add a table to your app, navigate to **Data > Tables** in the AppSheet editor, and click **New Table**.
2. In the **Get data from** dialog, click **Sheets on Google Drive**.
3. In the file picker, select the shipTo Google sheet from your drive folder and click **Select**. This sheet was automatically provisioned for the lab.
4. In the **Create a new table** dialog, click **Add This Table**.

AppSheet adds the table to your app. You can now create and use views for this table in the app.

### **View and update the shipTo table column structure**

1. In the AppSheet editor, navigate to **Data > Tables** and expand the **shipTo** table configuration.
2. Click **View Columns**. AppSheet has automatically inferred the types of the columns from the sheet.

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| ID | Text | Unique ID of the table |
| Product Name | Name | Name of the product being shipped |
| Quantity | Number | Number of units of the product being shipped |
| Date | Date | Current date of the request |
| Status | Text | Status of the request |
| Company | Number | ID of the company to which the product is being shipped |

1. In the column structure for the **ID** column scroll the properties and deselect the **Show?** and **Search?** properties.

The **ID** column was selected as the **key** for the table and is automatically populated with system generated unique values using the UNIQUEID() expression in the **INITIAL VALUE** property.

As this column contains random system generated values, it can be hidden and made unsearchable for the app user.

Learn more about using [keys in AppSheet](https://help.appsheet.com/en/articles/1023086-what-is-a-key).

1. Click the pencil icon to the left of the **Company** column to open the column property edit form.
2. The **Company** column refers to the company to which items are being shipped, so its type should be set to **Ref**.

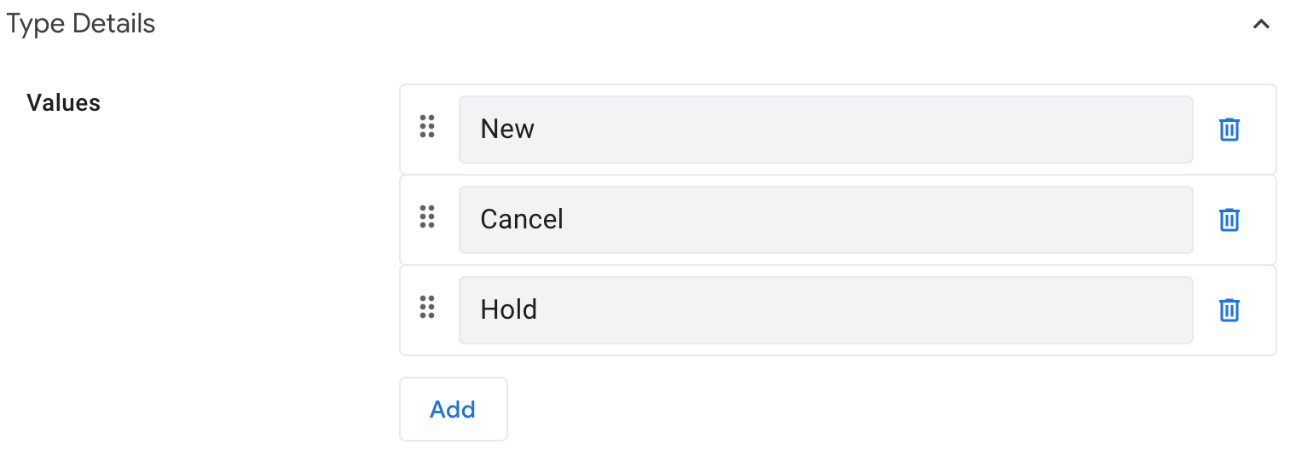
In this form specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**(type or select) |
| Type | Ref |
| Source table | companies |

1. Click **Done** to close the column property edit form.
2. The **Status** column should be populated from a fixed set of values and not contain free text entered by the user.

To change the type of the **Status** column, click the pencil icon to the left of the column.

1. In the column edit dialog, change the type to **Enum**.
2. Under **Type Details**, for **Values**, click **Add** and type **New**.
3. Repeat the previous step to add additional Status values: **Cancel**, **Hold**.



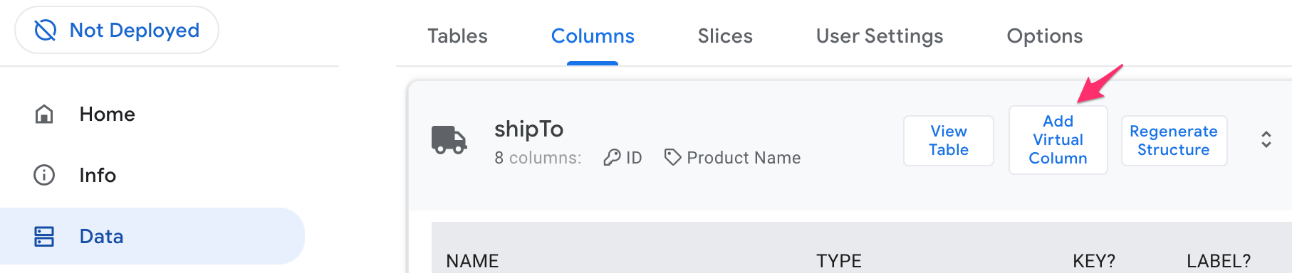
1. Click **Done**.
2. Click **Save** to save your changes.

### **Add a virtual column to the shipTo table**

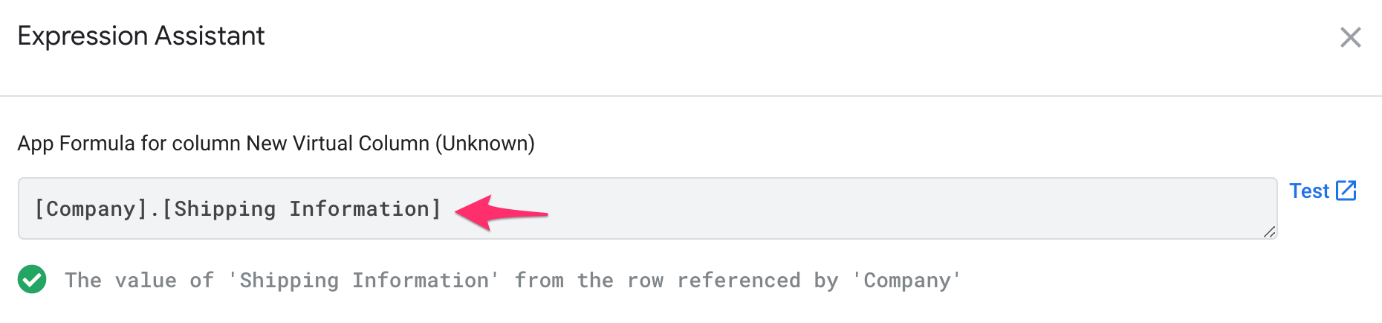
It would be nice to also include a company's shipping information whenever the **shipTo** form is displayed in the app. Note that the **Shipping Information** column in the companies table already stores this information.

We can use a virtual column in the **shipTo** table to implement this capability. A virtual column is not an actual column in the underlying sheet. Instead, it is automatically computed via an app formula expression.

1. In the **shipTo** table's columns configuration, click **Add Virtual Column**.



1. In the **New Virtual Column** dialog, for **Column name**, type **Shipping Info**.
2. Click in the **App formula** box to open the **Expression Assistant**.
3. In the **Expression Assistant** dialog, for **App Formula for column New Virtual Column**, type **[Company].[Shipping Information]**.



**Note**

Enter the expression as shown above, including the period (.) between the two column names.

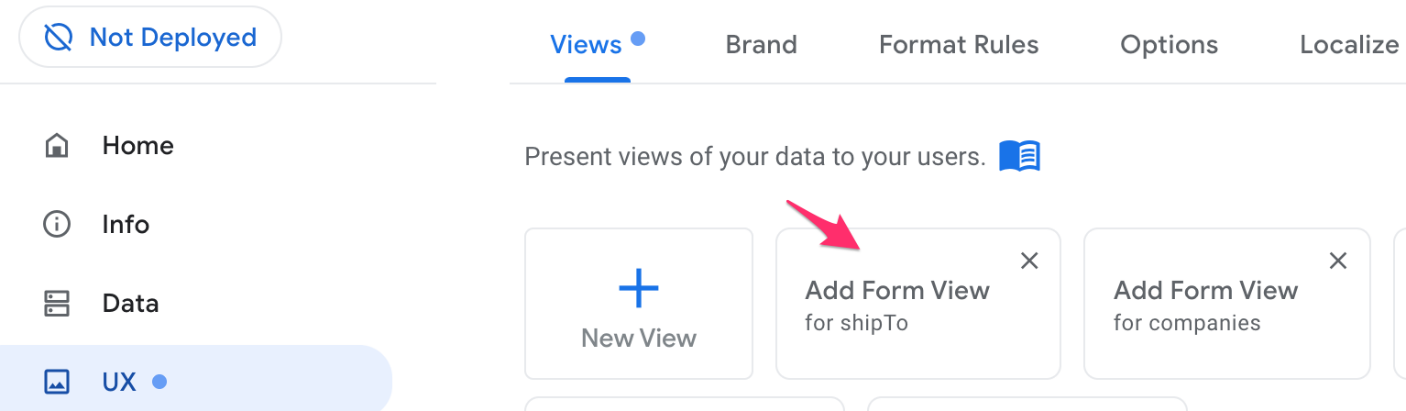
The expression evaluates to the value of the **Shipping Information** column of the row in the companies table that is referenced by the company ID value in the **Company** column in the shipTo table.

1. Click **Save** in the **Expression Assistant**.
2. Click **Done** in the **New Virtual Column** dialog.
3. Click **Save** to save the configuration changes.

The virtual **Shipping Info** column will now automatically be used in the form view that you will create in the next task.

### **Create the shipTo form view**

1. To create the **shipTo** form view, navigate to **UX > Views** in the AppSheet editor, and click **Add Form View for shipTo**.



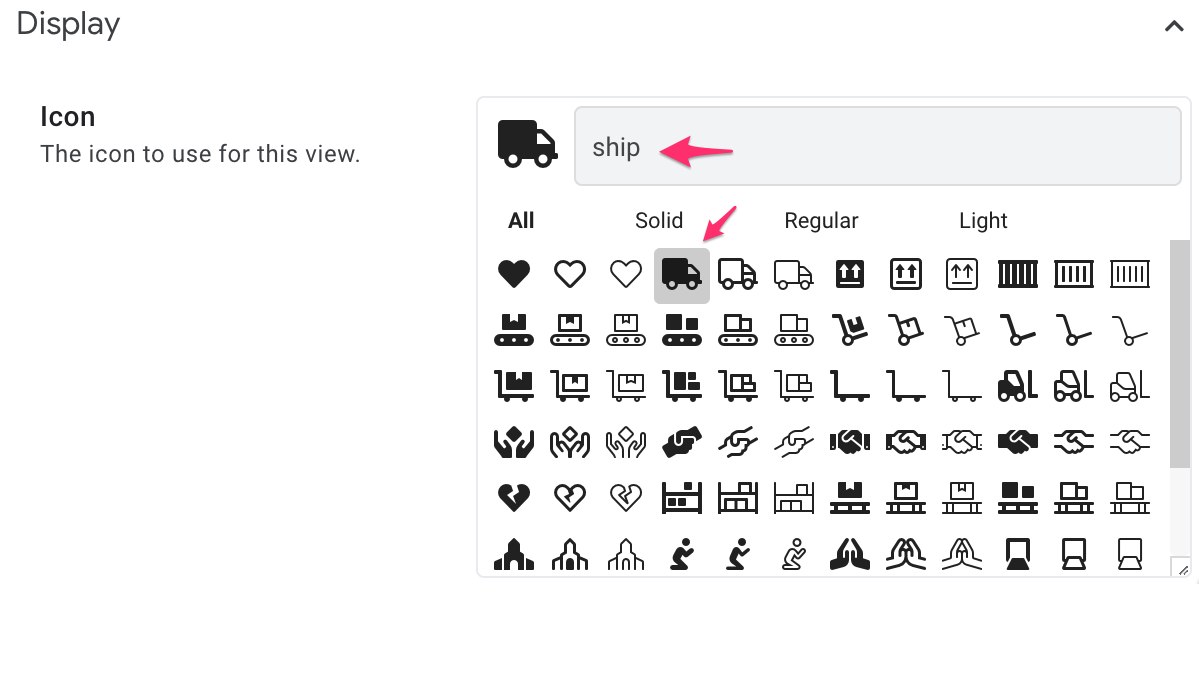
**Note**

If this form view button is not available, click **New View** to add the form view.

1. In the **New View** form specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value** (type or select) |
| View name | ShipTo |
| For this data | shipTo |
| View type | form |
| Position | right |

1. Under **View Options**, for **Form style**, select **Side-by-side**. This style lays out the input fields with their labels alongside the field's UI component.
2. Under **View Options**, for **Finish view**, select **shipTo\_Detail**. This sets the view that the app will display after the user submits this form.
3. Scroll the form and expand the **Display** section.
4. For **Icon**, type ship in the search box.
5. Select any one of the appropriate icons. This icon will be displayed in the center position of the app's bottom navigation bar.



1. Click **Save** to save your changes in the AppSheet editor.

### **Test the ShipTo form**

Follow the steps below to preview and test the new form.

1. If the live app preview is currently displaying the **ShipTo** form, click **Cancel** in the app's bottom navigation bar.

We will test the user experience starting from the initial view in the app.

1. Note the new **ShipTo** view icon on the app's bottom navigation bar. Click **companies** in the app's bottom navigation bar.
2. Click the **ShipTo** icon to open the form.
3. In the form specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**(type or select) |
| Product Name | Widget 1 |
| Quantity | 2 |
| Status | New |
| Company | Any company from the list |

1. Note that the app automatically displays the Shipping Info based on the selected company.
2. Click **Save** to submit the form and save the new shipping entry to the shipTo table.

### **Verify the entry in the Google sheet**

1. As a final verification step, switch to your browser tab to view the shipTo Google sheet on drive.
2. Verify that the new shipping entry has been added to the sheet.

Click Check my progress to verify the objective.

Using forms in your app

Check my progress

## Task 6. Customize the look and feel of your app

In the final task of this lab, you customize your app's look and feel by implementing a few branding changes.

### **Change theme and color**

1. To change the app's theme and color, navigate to the **UX > Brand** tab in the AppSheet editor.
2. Switch between the **light** and **dark** themes by clicking on each theme. Select one of the themes that best suits your preference.
3. You can change the primary colors used for text, buttons and icons in the app by selecting one of the colors from the list of colors available, or create a custom color of your choice.

Select any one of the colors to customize the look of your app.

### **Change logo and images**

1. To change the app's logo, select one from the list.
2. To change the launch image of the app, select one from the list.
3. To change the background image of the app, select one from the list.

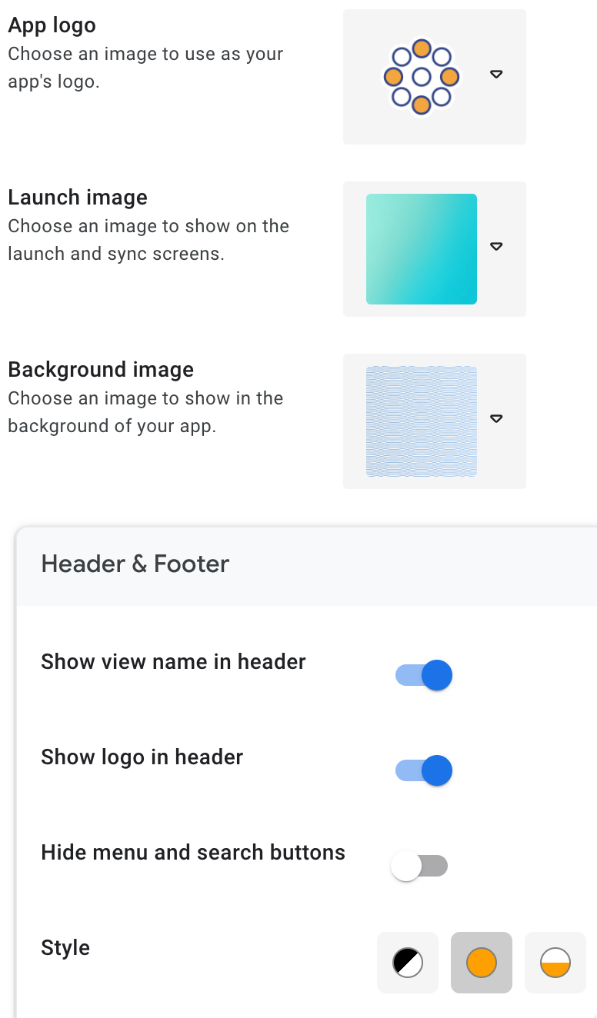
**Note**

For the app's logo and images, you can also provide custom logos and images by hosting them on your cloud providers file storage service and providing the path to the files.

### **Customize header and footer**

You can choose to display the app's logo and view names in the app's header, hide the app's menu and search buttons, and change the color style of the header and footer.

1. Click **Show view name in header** to enable this feature.
2. Click **Show logo in header** to enable this feature.
3. For style, select the full color style option.



1. Click **Save** to save your changes.
2. Preview your branding changes in the live app preview.

# Demo 4 Publishing your App using AppSheet

## Task 1. Create the app

1. Click the link to copy the Customer contacts app to your AppSheet account:

<https://www.appsheet.com/Template/AppDef?appName=Lab3-CustomerContacts-3856613&copy=1>

1. On the **Clone your App** form, specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**(type or select) |
| App name | Customer Contacts |

1. Click **Copy app**.
2. Click **Start Customizing** to go to the AppSheet editor.

Your app is set up with the original contacts and companies data sources, and you can now continue to build out the app's functionality.

Click Check my progress to verify the objective.

Create the app

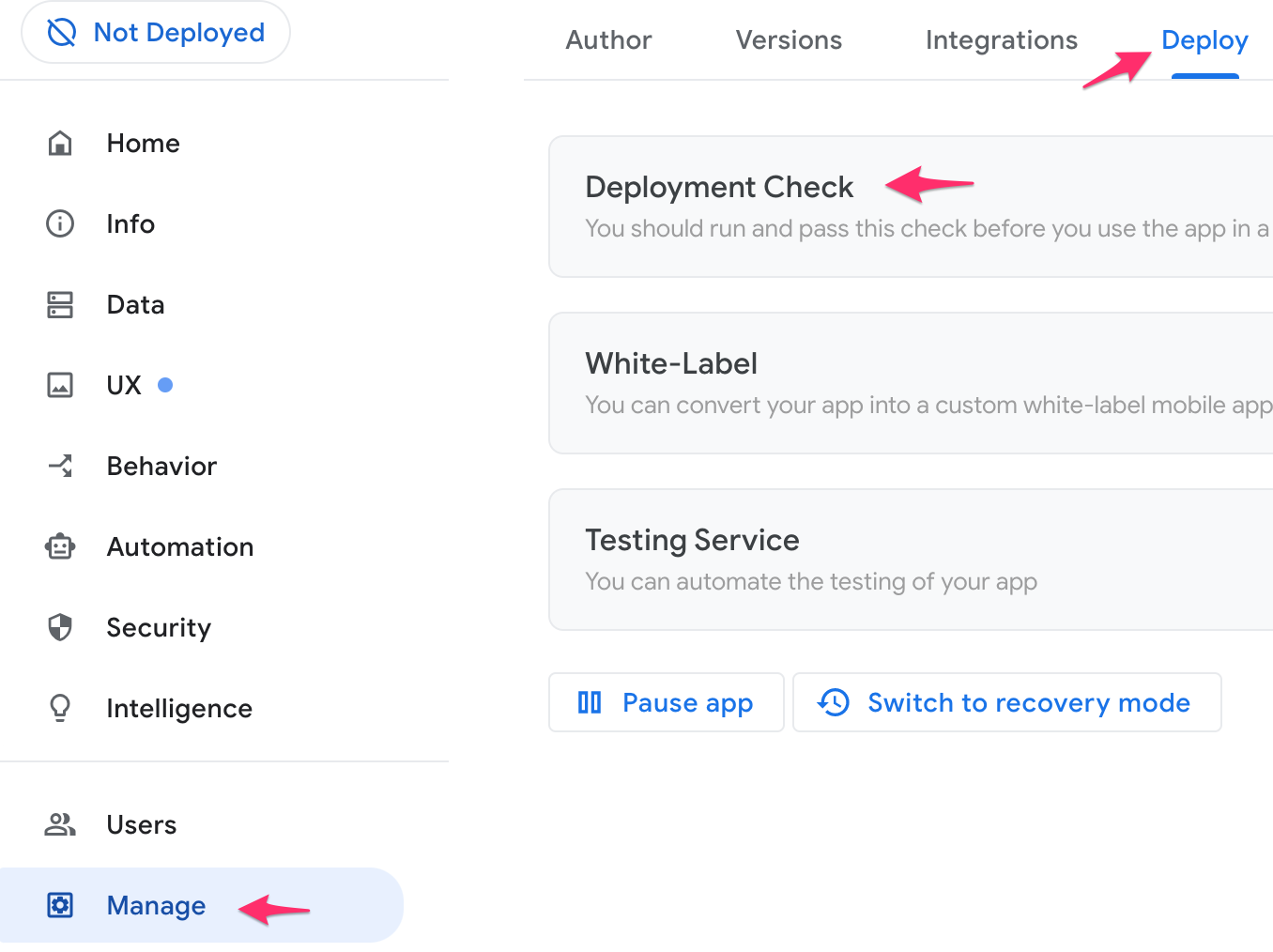
Check my progress

## Task 2. Check your app for errors or warnings

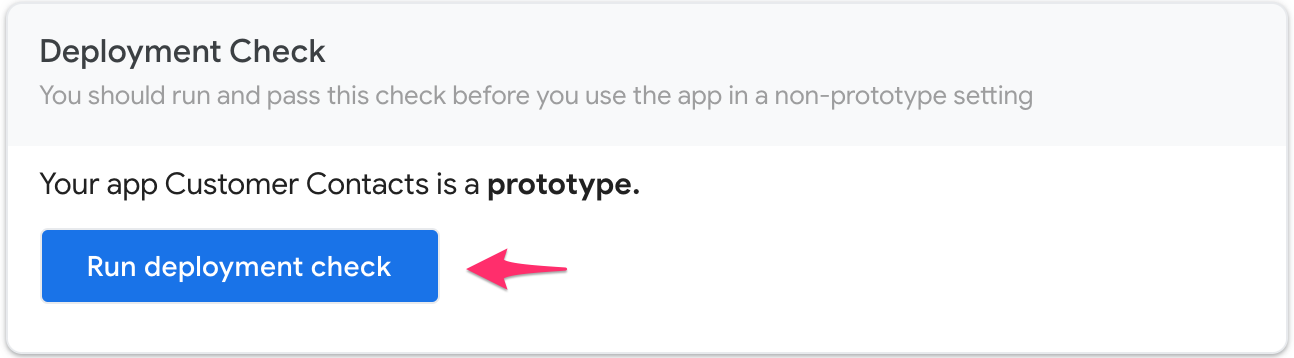
In this task, you check your app for any errors or warnings by running a deployment check.

### **Run a deployment check**

1. To run a deployment check on your app, navigate to the **Manage > Deploy** tab in the AppSheet UI.
2. Click **Deployment Check**.



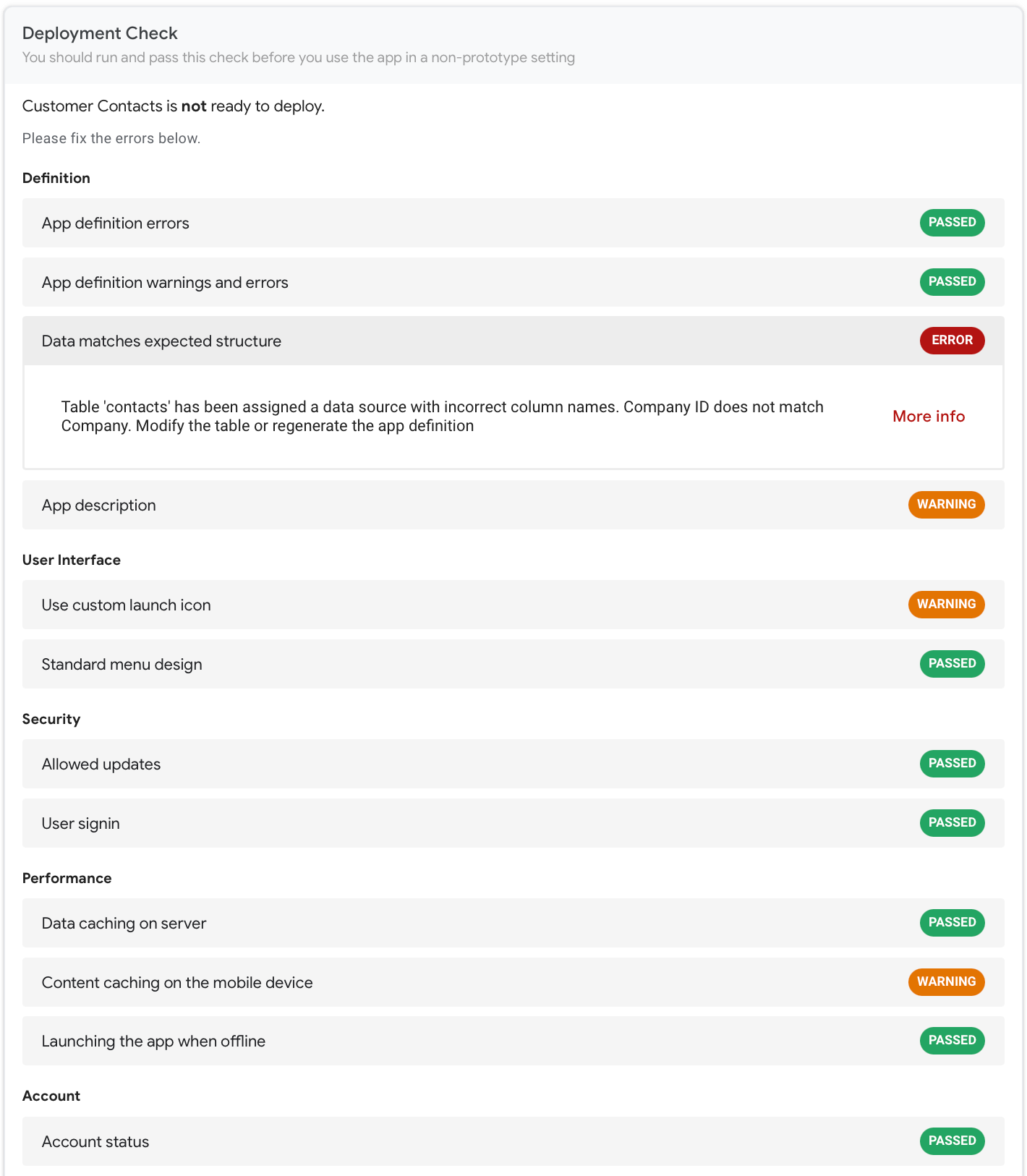
1. Click **Run Deployment Check**.



The output of the deployment check lists any errors or warnings that you should fix, before deploying the app.

### **Inspect the deployment check report**

1. View the report to look for any errors or warnings from the deployment check. Scroll to see the full list of issues.



The report contains a few errors or warnings. You fix these issues in the next task.

## Task 3. Fix any errors and warnings from the deployment check

In this task, you review the details of each error or warning as reported in the output of the deployment check and fix them.

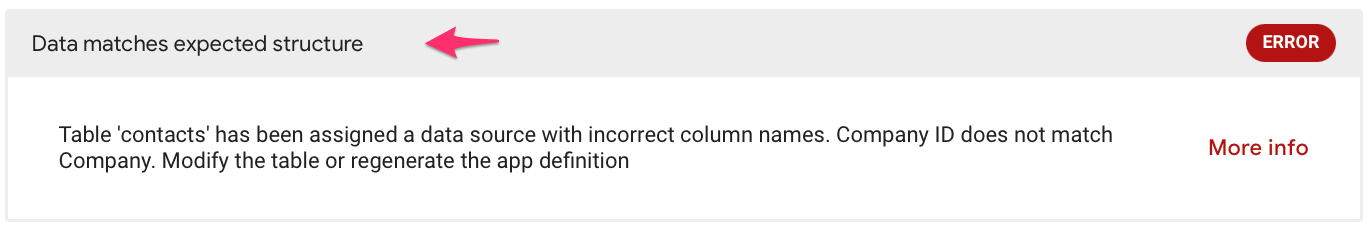
**Note**

Warnings do not prevent you from deploying your app, but you must fix any errors that are reported. It is a good practice to review all warnings and resolve them if possible.

### **Fix the Data structure error**

1. Click the Data matches expected structure error.

The section expands to provide more details on the error.



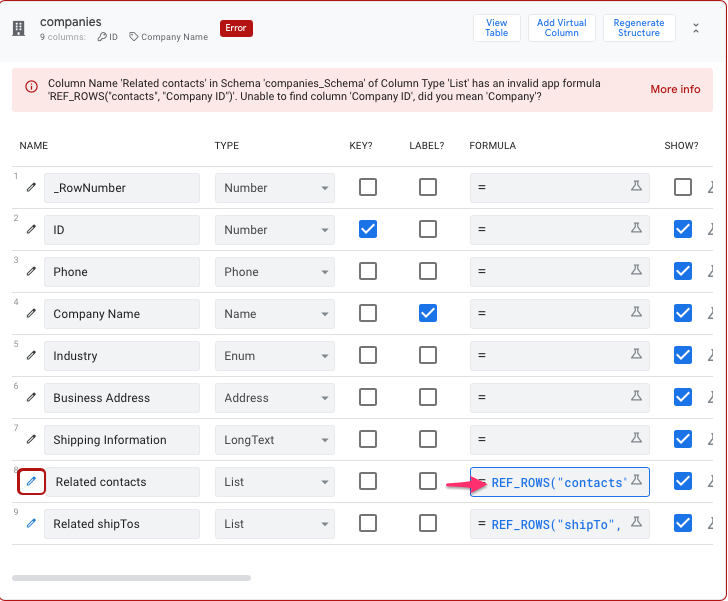
1. Click **More info** to view more details about this error.

From the description of the error, you can see that there is a mismatch between the name of the Company ID column in the app definition and the name of the column Company in the contacts Google sheet.

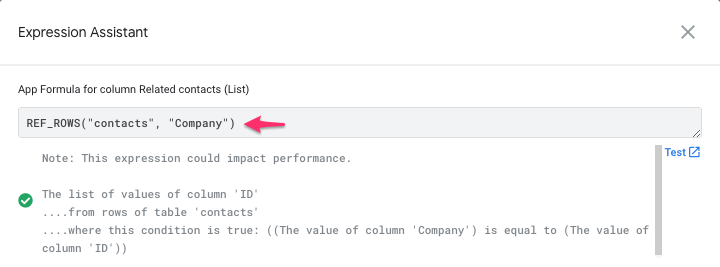
1. To fix this error, navigate to the **Data > Columns** section in the AppSheet editor, and click **contacts** to open the table definition.
2. Edit the name of the Company ID column and update it to Company.
3. The Company column is also used in the app formula expression for the Related contacts reverse reference column in the Companies table, which must be updated.

In **Data > Columns**, click **companies** to open the table definition.

1. Click into the **Formula** field of the Related contacts column definition to bring up the Expression Assistant.



1. Edit the **App Formula** expression to update the column name to Company.



1. Click **Save** in the Expression Assistant.
2. Click **Save** to save your app changes.
3. Graphical user interface, application

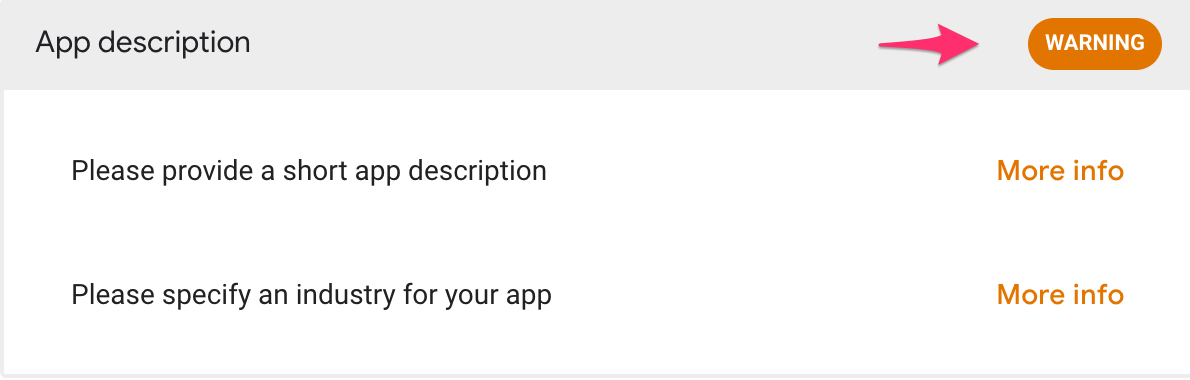
   Description automatically generated

### **Fix the App description warning**

It is a good practice to include a short description of your app.

1. Click the App description warning.

The section expands to provide more details on the warning.



Click **More info** to get more details on the issues reported that include the likely cause and resolution steps.

1. To fix the App description warning, navigate to the **Info >> Properties** tab in the AppSheet UI.
2. Click **App Properties**.
3. In the **App Properties** form, specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**(type or select) |
| **Short Description** | An app to manage all of your customers and contacts in one place. |
| **Industry** | Business Services |

1. Click **Save** to save your changes.

### **Fix the Custom launch icon warning**

Your app is already configured to use one of the logo icons provided by AppSheet. This warning is to remind you to change the logo to your own custom icon.

1. Click the Use custom app launch icon warning.

The section expands to provide more details on the warning.

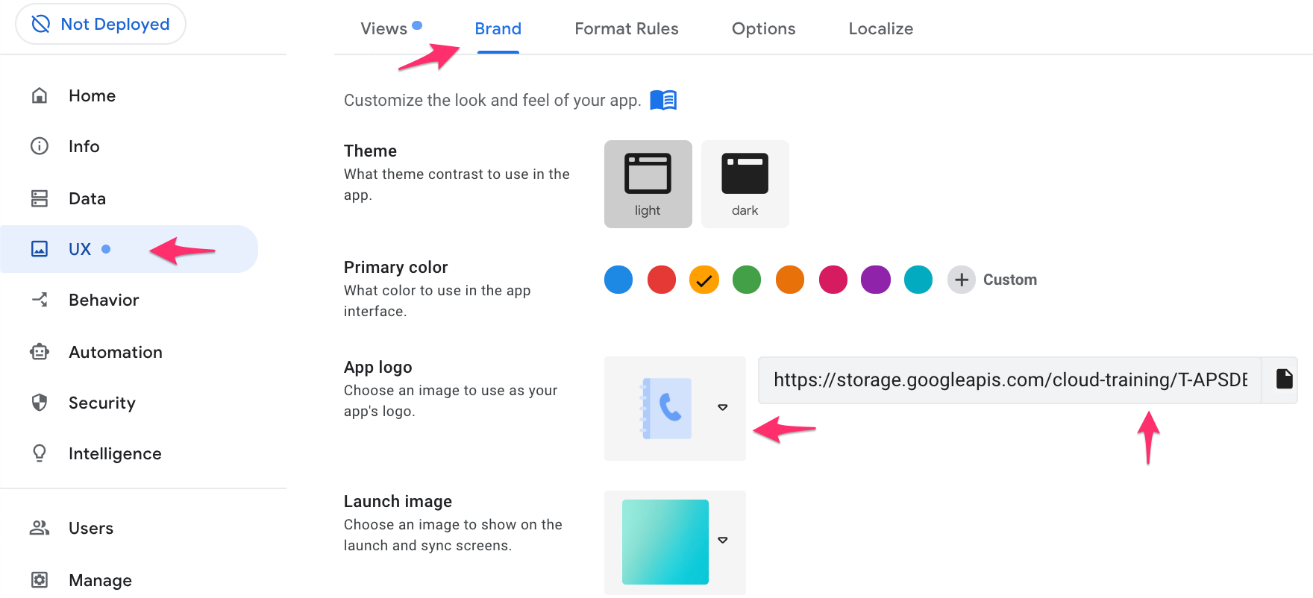
Optionally, click **More info** to get more details on the issues reported that include the likely cause and resolution steps.

1. To provide a custom app launch icon for your app, navigate to the **UX >> Brand** tab in the AppSheet UI.
2. For **App logo**, select **Custom** from the dropdown list.
3. Copy and paste the url below in the edit box to the right of the logo:

https://storage.googleapis.com/cloud-training/T-APSDEV-B/app\_logo.png

Copied!

content\_copy



1. Click **Save** to save your changes.

### **Fix the Content caching on the mobile device warning**

This warning indicates that your app could benefit from offline device caching of images and documents.

1. Click the Content caching on the mobile device warning.

The section expands to provide more details on the warning.

Optionally, click **More info** to get more details on the issues reported that include the likely cause and resolution steps.

1. To enable this option, navigate to the **Behavior >> Offline/Sync** tab in the AppSheet UI.

Scroll down to the **Offline Use** section.

1. Enable the **Store content for offline use** option.



Learn more about [offline content caching](https://intercom.help/appsheet/en/articles/953714-offline-sync-the-essentials) at the AppSheet documentation site.

1. Click **Save** to save your changes.

## Task 4. Re-run the deployment check

Once you have fixed all errors and fixed or reviewed any warnings, you should run the deployment check again.

### **Run the deployment check**

1. Navigate to the **Manage > Deploy** tab in the AppSheet UI.
2. If the deployment check report from the previous run is open, click **Continue editing**, otherwise click on the **Deployment Check** panel to expand it.
3. Click **Run Deployment Check**.

The output of the deployment check should contain no errors or warnings. We can now go ahead and deploy the app in the next task.

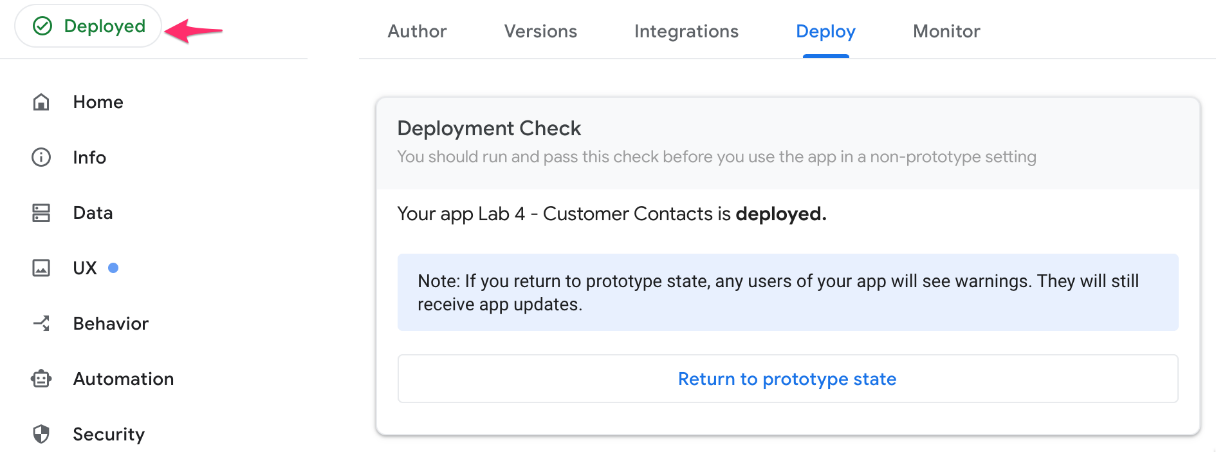
## Task 5. Deploy and test the app

In this task you deploy the app and test it outside of the AppSheet UI directly within a browser.

### **Deploy the app**

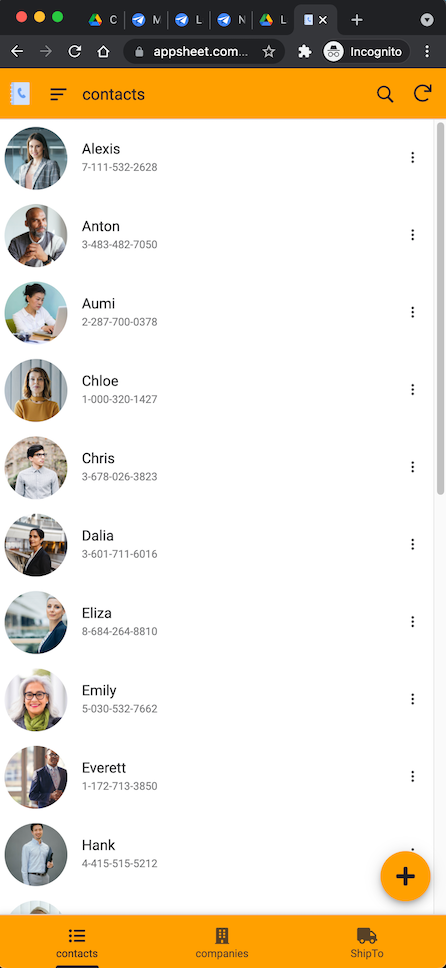
* In the **Deployment Check** panel, click **Move app to deployed state**.

AppSheet now deploys your app and changes the app's state to Deployed.



### **Test the app**

1. In the AppSheet UI, navigate to the **Users >> Links** tab.
2. Copy the URL in the **Browser Link** box.
3. Navigate to that url in a separate browser window. The app should load in the new browser window.



1. Test out the features of the app from the desktop browser to make sure all the functionality works as expected.
2. From the desktop browser that is running the app, click the **ShipTo** icon to open the shipping form.
3. In the form specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**(type or select) |
| Product Name | Test Product |
| Quantity | 4 |
| Status | New |
| Company | Any company from the list |

1. Verify that the app automatically displays the Shipping Info based on the selected company.
2. Click **Save** to submit the form and save the new shipping entry to the shipTo table.

Click Check my progress to verify the objective.

Deploy and test the app

Check my progress

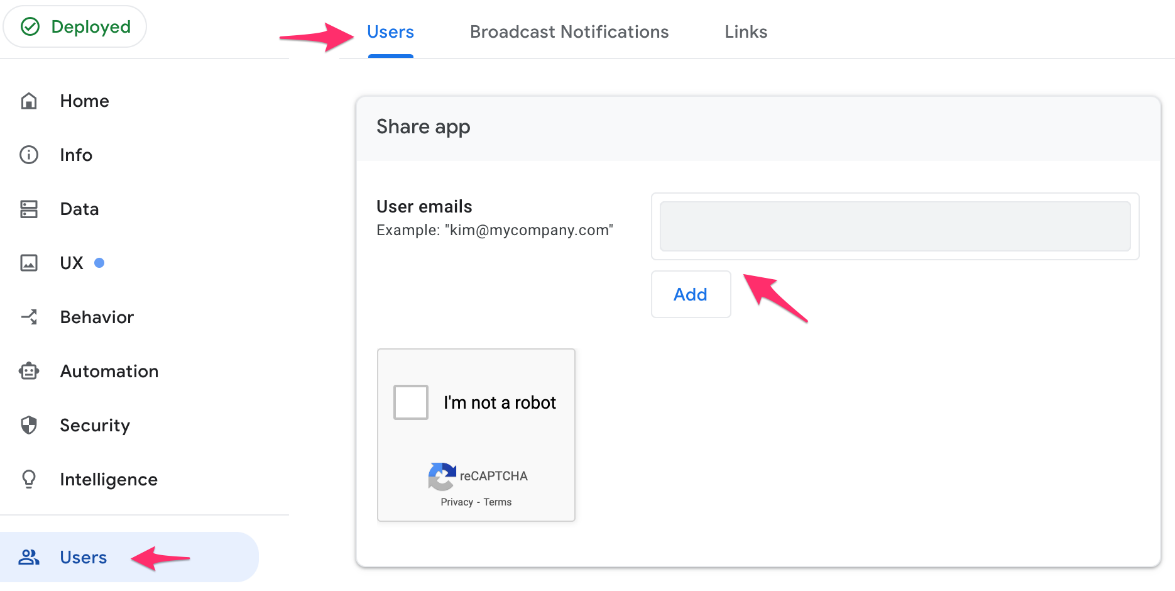
## Task 6. Sharing your app

Once you have fully tested your app, you can share the app with yourself or any user so that they can download and start using the app.

### **Add users**

The first step in sharing your app with users is to add them as users of your app.

1. To add a user to your app, navigate to the **Users >> Users** tab or click the add user icon in the AppSheet UI.



1. In the **Share app** form, for **User emails** type a valid email address.

Type your own valid email address so you can run the app as a user on your device in the next task.

1. Click **Add**.
2. Optionally repeat the two steps above to add any additional user email addresses.
3. Check the **reCaptcha** box to verify that you are not a robot.
4. Optionally, update the invite message.
5. Click **Add users and send invite**.

**Note**

The reCaptcha verification has an expiry time, so you may have to check it again if it times out.

### **Verify receipt of the email**

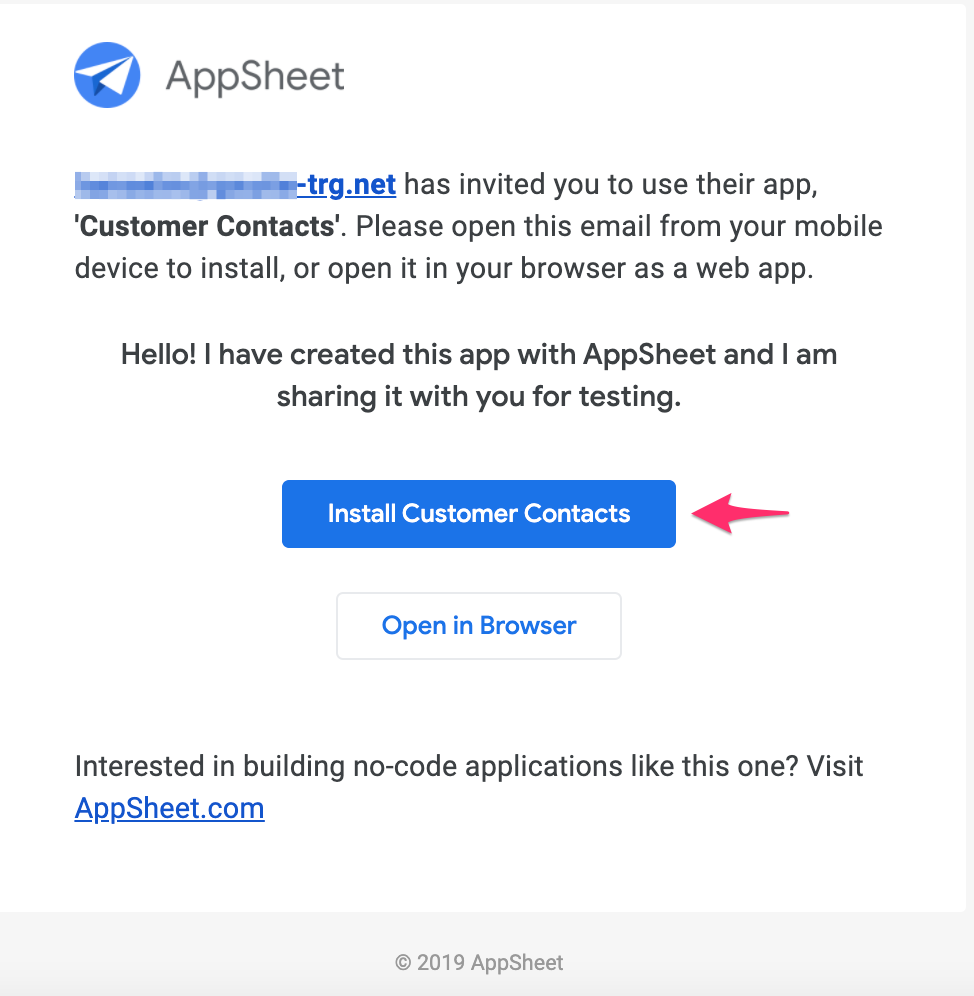
* Login to the email account you used in the previous task and verify that the email was received.

## Task 7. Install and run the app on your device

Now that the app has been published and shared, it is a good practice to run and test the app on your own device as an end user.

### **Install the app**

1. Open the email you received in the previous task on your device.



1. Click **Install Customer Contacts** in the email.
2. Click **Install**.
3. If you do not already have the AppSheet app installed on your device, you are prompted to install it first.

The AppSheet app is a hosting app that hosts the apps that your build.

When you install and run your app on your device, it will appear to run on its own, but it is actually "hosted" by the AppSheet Hosting app.

Click **Install** to install the AppSheet app from your device's app store.

**Note**

Once the *AppSheet* app is installed on your device do **not** open the app.

1. Return to your email and click **Install Customer Contacts** in the email again.
2. Click **Install** to install the **Customer Contacts** app.
3. If prompted, in the **Add to home screen**, click **Add automatically**.
4. Click **Accept** to accept the user agreement.

The app is now installed on your device.

1. Test the app on the device as an end user to verify its functionality.