V.2 Native Share Menus -Share to New Expensify

AUTHORS
Georgia, Lizzi @ Infinite Red

SLACK ROOM
#expensify-infinite-red

DEADLINE
N/A

TRACKING ISSUE
[Github link]

Strategic Context

The only way to get a billion users on our platform is to leverage the natural viral dynamics of collaboration. Currently, they do this via WhatsApp, SMS, and other chat tools. We need NewDot to be an effective replacement for those tools in all of their core flows.

High-level overview of the problem

There are plenty of times when you might want to share something from outside sources into New Expensify. But there's no way to quickly share links/photos/information. For example, if you are in the Photos App and decide you'd like to share a photo, you'll have to leave the app → open New Dot → go to the correct chat → click add attachments → find the same image again → and so on. That adds significant friction and makes it less likely that users will default to Expensify as their go-to chat app.

Timeline and urgency

This project is a requirement for Reunification as it's a core feature used by many of our Mobile App users. In Old Expensify, you can share photos to initiate a smart scan. To replicate this functionality of our previous platform, we need to add support for sharing and initiating SmartScans into NewDot.

Terminology

Direct Message (DM) - a chat between two individual people. Specifically not a group chat.
Left Hand Nav (LHN) - the component containing the list of chats the user is in, shown on the left-hand side of the screen on the desktop-width UI.
Native Share Menu - the native UI that pops up when a share is triggered. This includes the list of apps that are available to share into.
Participant - The person, people, or workspace that money is requested from.
Request - A request for payment from a person or workspace. This can also be an expense split with other people.
Share-to-Expensify - The user flow while attempting to send something from outside the app to NewDot, such as sending someone a photo from the Photos App. This applies whether the user picks "Share" or "Scan" within the app.
Share-from-Expensify - The user flow when attempting to share something in-app to an external source, such as sending a QR code to iMessage.
Share attempt - A noun to describe the process of Share-to-Expensify (so we can say "The user cancels the share attempt" instead of "The user cancels the share.") This applies whether the user picks "Share" or "Scan" within the app.
Share to chat - Specifically describes the act of sharing a file or message to chat, via the "Share" tab, to distinguish from Share-to-Expensify generally.
SmartScan/Scan - The feature in Expensify that allows users to upload a photo of a receipt, which is used to automatically fill out fields in a money request.

High-level of proposed solution

[Pre-design] / [Design Pre-design] / [2nd iteration Pre-design]

Let's add New Expensify into the Native Share Menus for iOS and Android to allow sharing files, links, and text into New Expensify from outside of the App (slack). There will be no restrictions on file type. Users can also choose instead to use the file to create a request with SmartScan, if the file type is compatible.

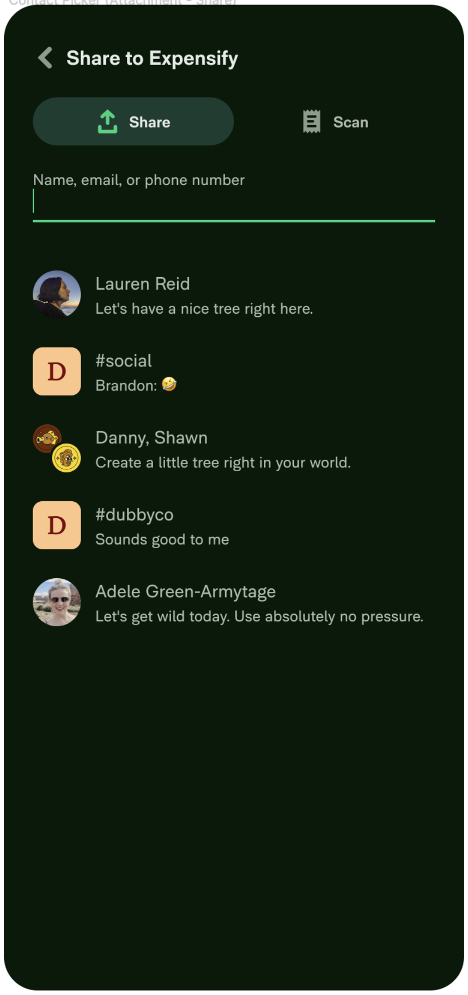
When a share is initiated and NewDot is selected in the system UI, a "Share to Expensify" screen will show. This screen will have a tab bar with two tabs to choose from:

1. "Share", where the user can send the shared file, message, or link to any existing chat or new DM, or
2. "Scan", where the user can create a new request, with the shared file attached as the receipt.

On first load, we will default to the "Share" flow, but after that we'll record the user's tab selection and show them their last-selected tab.

If the user has shared a file incompatible with SmartScan, the tab bar will not appear, as there is no file to scan for the request. (Pre-design thread)

Contact Picker (Attachment - Share)



Lauren Reid
Let's have a nice tree right here.

#social
Brandon: 🤪

Danny, Shawn
Create a little tree right in your world.

#dubbyco
Sounds good to me

Adele Green-Armytage
Let's get wild today. Use absolutely no pressure.

Share

“Share” is the tab view selected by default. When selected, the page will feature our search component and allow users to search and select the destination of the share, which can be any chat they are currently in, or a new DM.

The second page will display which chat the user selected to share to, a preview of any file being shared, and a text input where the user can write a message to share. This text input will be prefilled if the user chose to share text, including a link.

Once the user clicks “Share”, they will be navigated to the chat they shared to, where they will see the attachment and/or message they just sent, with the same end result as if they had sent the message from the chat screen itself.

If a user adds a message to an attachment, the resulting chat will appear in New Expensify as the message first and then the attachment on the next line (as a separate message). (slack)

Scan

When the user selects the “Scan” tab, the search component for requests will be shown, allowing the selection of workspaces or people as request participants.

On the second page, we will reuse the existing design of the Money Request creation screen, allowing the user to preview the file they’d like to scan, confirm where the request should be sent, and edit any details they’d like to before submitting the request.

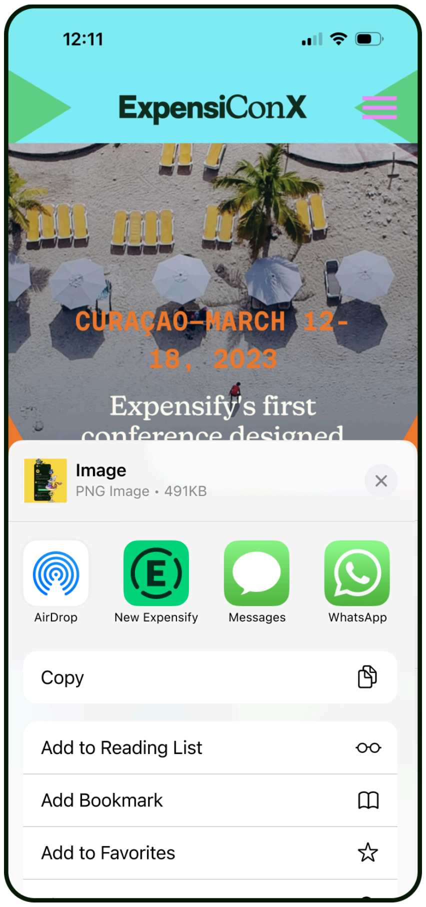
User is not logged in

If a user has not yet logged into New Expensify when they try to share, they will be shown the sign-in screen and the share attempt will be abandoned. The user can attempt again to share once they are logged in.

UI Additions & Changes

App Icon in Native Share Menu

The icon for New Expensify will appear in the Native Share Menu.



Share

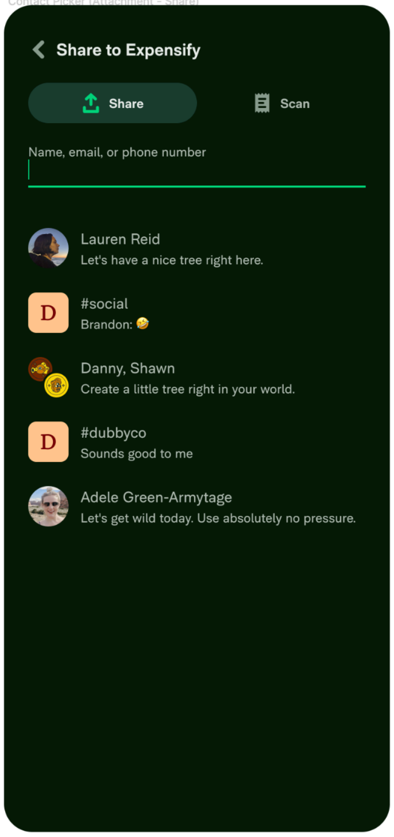
After the user selects New Expensify in the Native Share Menu, they will be directed to the **Share Root** page.

Choosing who to share to

When the user continues in the “Share” tab, we will display a chat search (based on the existing **Chat Search** page), which allows a user to:

- find any existing chat they have, or
- create a new DM by typing in contact information.

Contact Picker (Attachment - Share)



Share to Expensify

Name, email, or phone number

Lauren Reid
Let's have a nice tree right here.

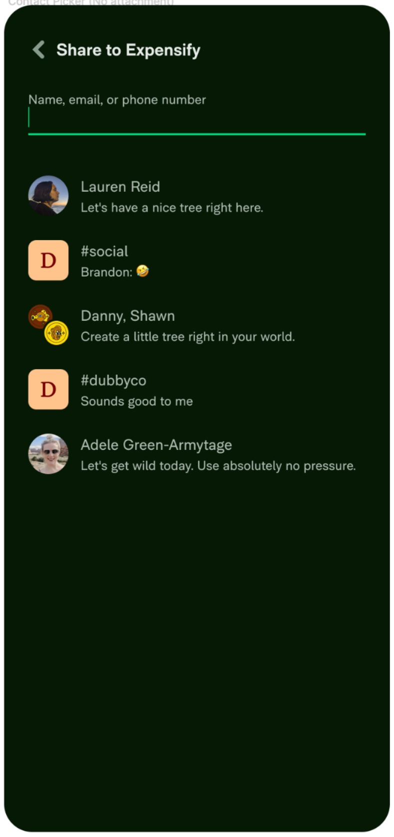
#social
Brandon: 😋

Danny, Shawn
Create a little tree right in your world.

#dubbyco
Sounds good to me

Adele Green-Armytage
Let's get wild today. Use absolutely no pressure.

Contact Picker (No attachment)



Share to Expensify

Name, email, or phone number

Lauren Reid
Let's have a nice tree right here.

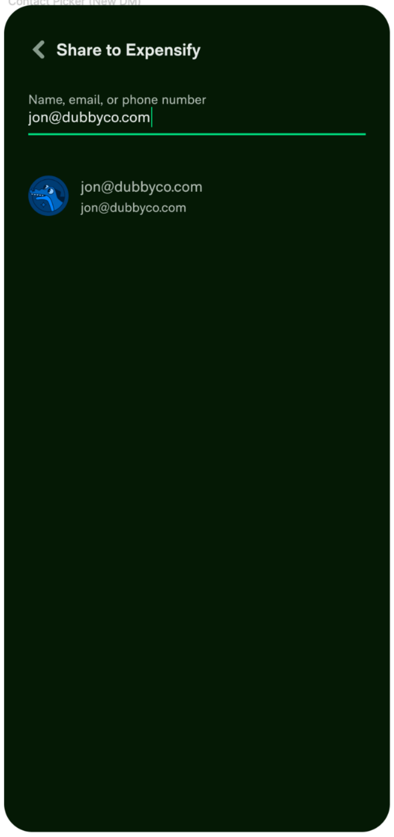
#social
Brandon: 😋

Danny, Shawn
Create a little tree right in your world.

#dubbyco
Sounds good to me

Adele Green-Armytage
Let's get wild today. Use absolutely no pressure.

Contact Picker (New DM)



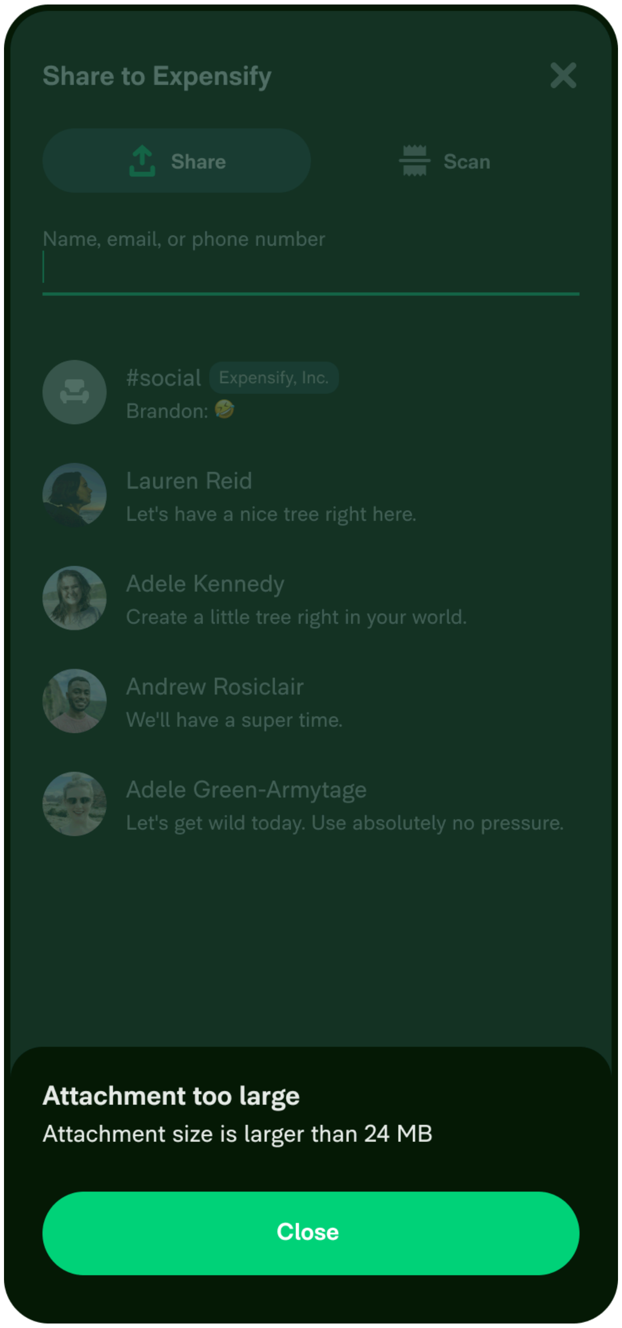
Share to Expensify

Name, email, or phone number
jon@dubbyco.com

jon@dubbyco.com
jon@dubbyco.com

Warning when files are too large

If the user tries to share a file that is above the attachment size limit (above 24MB), we will immediately show an error message, in the same style as the current attachment picker in a chat. (Pre-design thread)


Share to Expensify

Name, email, or phone number

#social Expensify, Inc.
Brandon:

Lauren Reid
Let's have a nice tree right here.

Adele Kennedy
Create a little tree right in your world.

Andrew Rosiclair
We'll have a super time.

Adele Green-Armytage
Let's get wild today. Use absolutely no pressure.

Attachment too large
Attachment size is larger than 24 MB

Pressing “Close” will take the user out of this modal flow, back into New Expensify. If the user wishes to go back to the app they shared from, they have other options provided by their operating system. See “Abandoning a share attempt” for details.

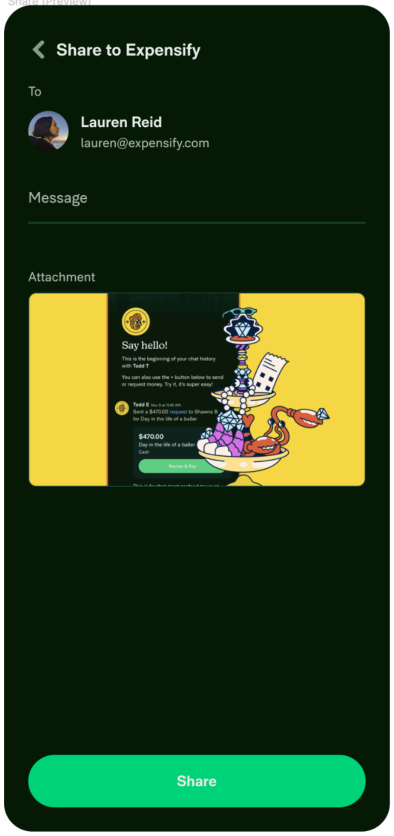
Previewing the share

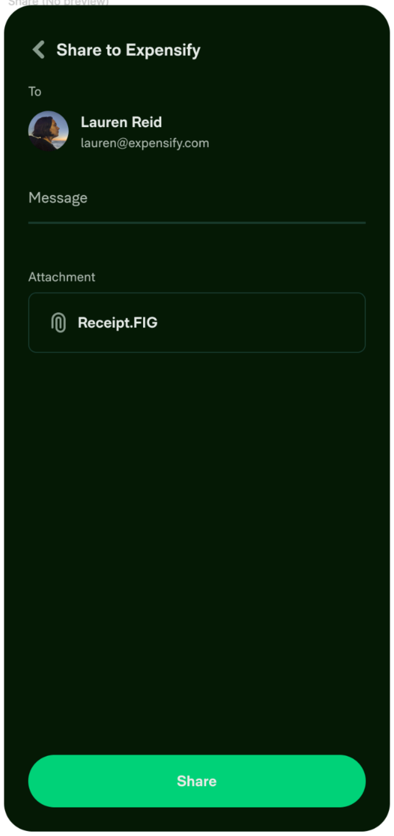
When the user selects a chat, we navigate to the **Compose Message** page.

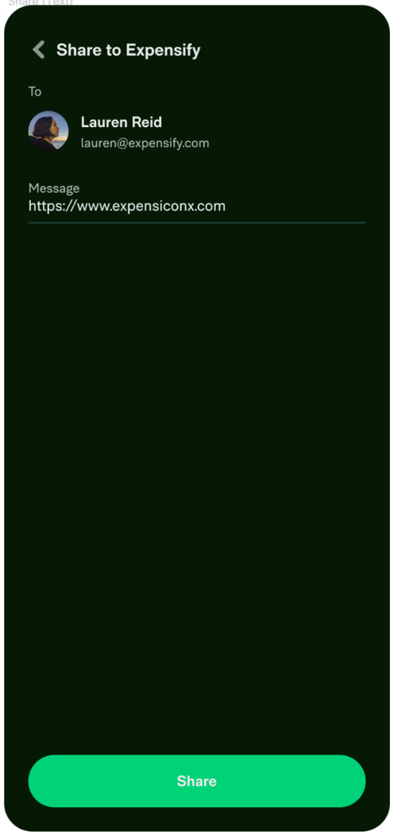
Here, the user will see:

- Who they are sharing to
- A preview of the file they are sharing (if applicable)
- A text input for an optional message

The content on this screen varies depending on what the user shared: a previewable file, a non-previewable file, or text.







Sharing files

When a file is shared:

- the `AttachmentView` component will appear, and
- The message field will be empty.

The `AttachmentView` will determine whether a file is previewable or not, and to render either a preview of the attachment, or an icon with the file name. We will not be making changes to what `AttachmentView` can preview.

Sharing text

When the user shares text, such as a link or a message, it will be added to the message field, which will auto-grow to show multiple lines.

Since there is no file to preview, the `AttachmentView` component is omitted.

The user can then edit the message as desired.

Sending the shared data

In either case, once the user is ready, they press "Share" to send the attachment and message.

Pressing "Share" navigates to the chat they shared to, where they can see the sent attachment and/or message.

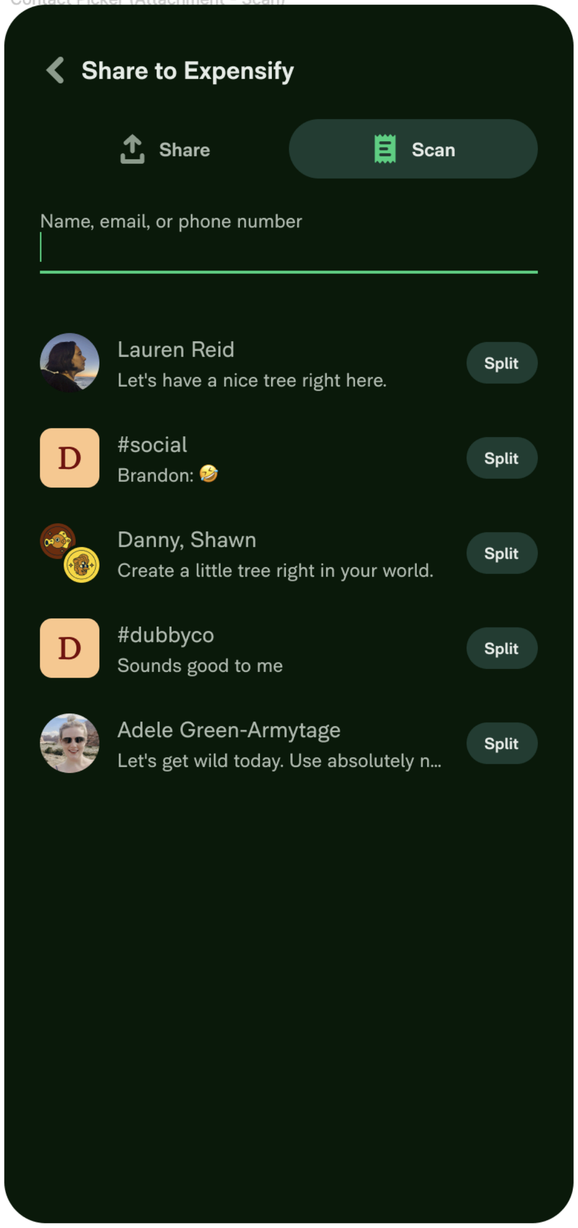
Error handling

If the user tries to share or enter a message longer than 10k characters, the text input will be highlighted red and an error displayed, and the error will have to be addressed before the user can continue. (Pre-design thread)

Scan

If the user selects the "Scan" tab instead of "Share", they will enter the request flow, identical to creating a money request in the app today, starting on the **Request Participants** page, embedded in the Root page tab navigator.

Contact Picker (Attachment - Scan)


Share to Expensify

Share | Scan

Name, email, or phone number

Lauren Reid
Let's have a nice tree right here.
Split

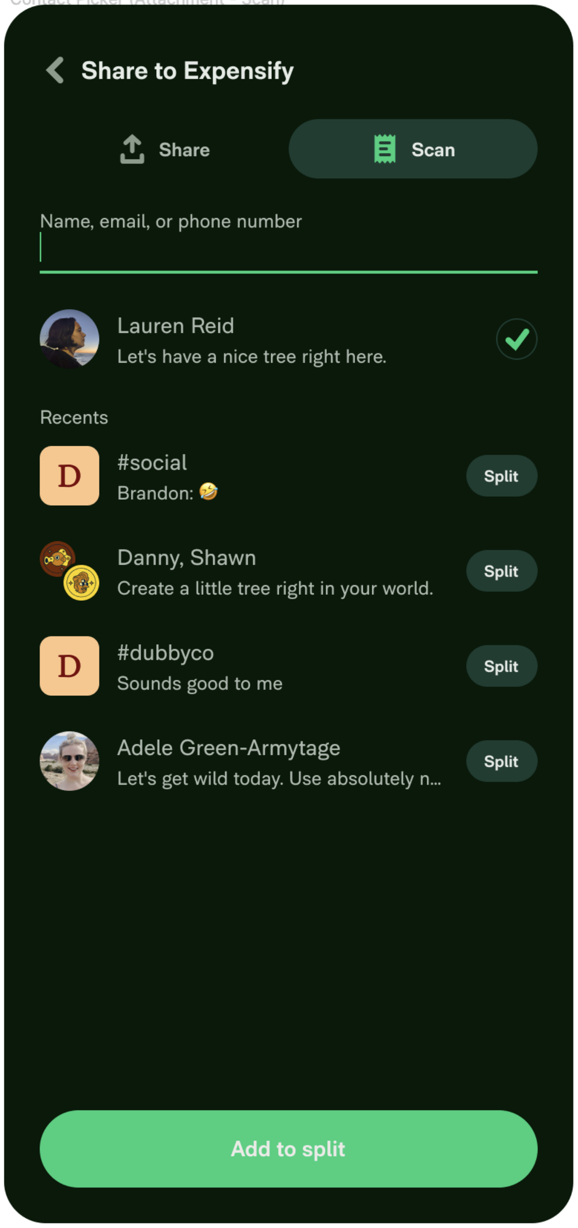
#social
Brandon: 🤣
Split

Danny, Shawn
Create a little tree right in your world.
Split

#dubbyco
Sounds good to me
Split

Adele Green-Armytage
Let's get wild today. Use absolutely n...
Split

Contact Picker (Attachment - Scan)


Share to Expensify

Share | Scan

Name, email, or phone number

Lauren Reid ✓
Let's have a nice tree right here.

Recents

#social
Brandon: 🤣
Split

Danny, Shawn
Create a little tree right in your world.
Split

#dubbyco
Sounds good to me
Split

Adele Green-Armytage
Let's get wild today. Use absolutely n...
Split

Add to split

The user can search for and select a workspace or people to request money from.

Selecting a single target

Tapping a list-item selects a single target and immediately navigates to the **Request Confirm** page.

Splitting an expense among multiple people

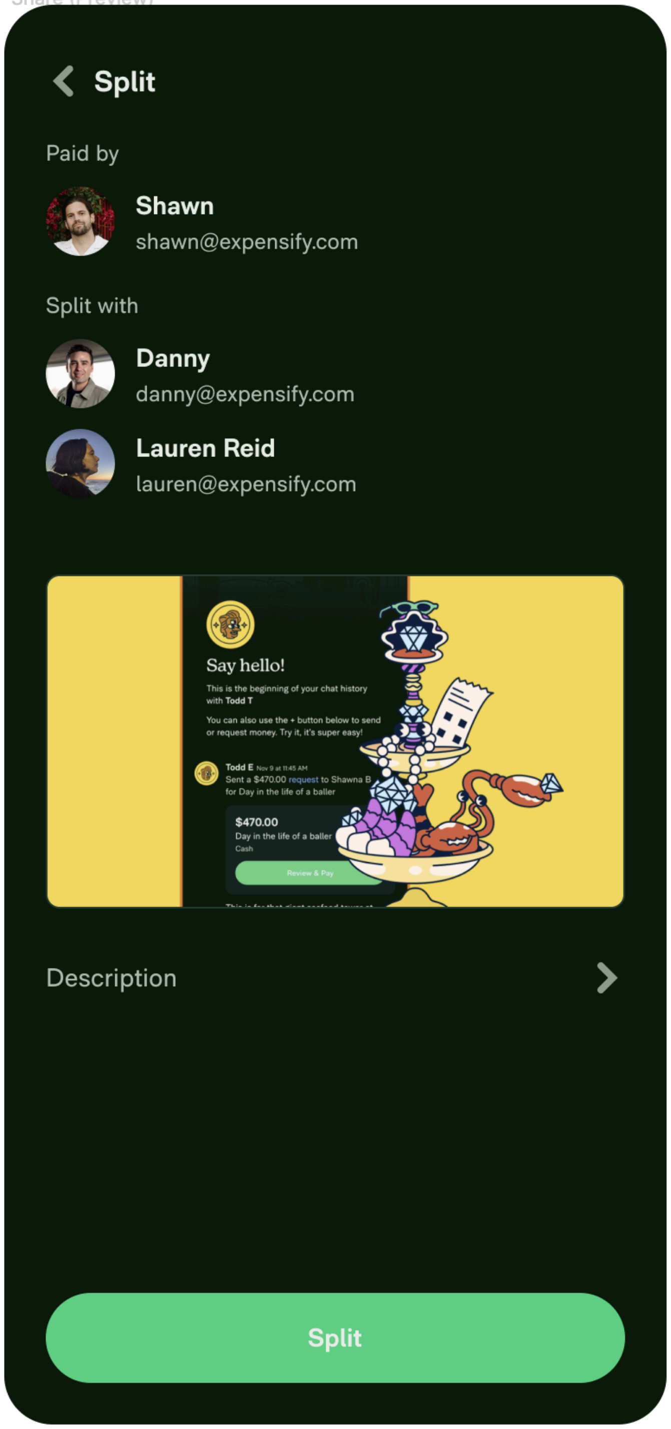
Tapping the “Split” button in a line item allows the user to select multiple people to split the expense with.

When someone is selected in this way, the “Add to Split” button appears. Touching this button navigates to the Split Confirm Page.

Confirming the details of the request

The **Request Confirm** page lets the user edit the details of the request before creating it.

Share (Preview)



Split

Paid by

Shawn
shawn@expensify.com

Split with

Danny
danny@expensify.com

Lauren Reid
lauren@expensify.com

Say hello!

This is the beginning of your chat history with **Todd T**

You can also use the + button below to send or request money. Try it, it's super easy!

Todd E Nov 9 at 11:45 AM
Sent a $470.00 request to Shawna B for Day in the life of a baller

$470.00
Day in the life of a baller
Cash

When the user creates the request by clicking the “Request”/“Split” button, they are navigated to the chat containing the request.

Abandoning a share attempt

If the user no longer wishes to share the item – for instance if they change their mind or need to back out during an error – they have several ways of going “back”, to cancel the share:

- Canceling and staying in New Expensify using the “back button” in the header
- Returning to the previous app using system gestures and/or controls

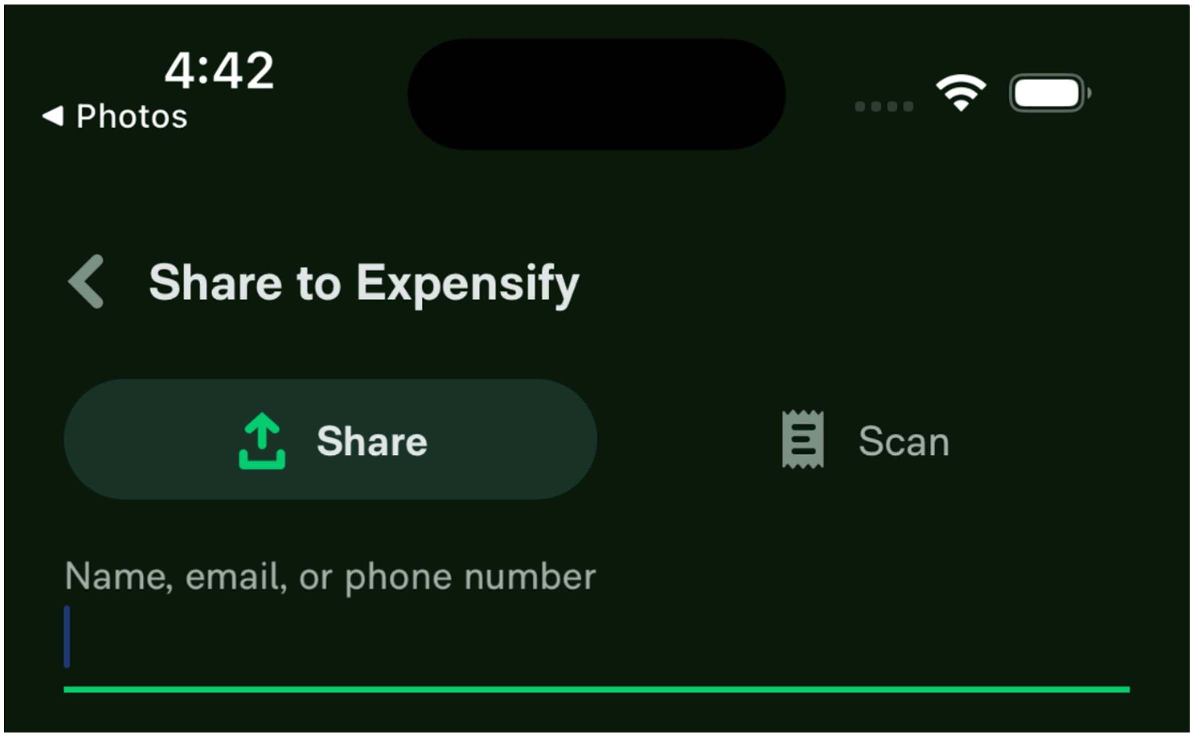
Return to New Expensify

To exit the share flow and continue into New Expensify, the user can press the “back” button in the header, which will close the share modal. If there is existing navigation history, they will be returned to the last non-modal screen they viewed; otherwise they will be returned to “Home” (i.e., displaying the last-accessed report, as when the app is first loaded).

Returning to the app they shared from

To return to the app they shared from, the user can use system-provided gestures or buttons.

On iOS the user can return to the app they shared from by touching the “breadcrumb” in the top left, in the status bar. This is operating system behavior, triggered by one app opening another, and can’t be changed.



On Android the user can use the back button or gesture to navigate back through the share screens, until they reach the **Share Root** page, at which point pressing back again will return them to the app they shared from.

Expensify.com / new.expensify.com

NewDot specific. Detailed section and implementation will be worked on by Infinite Red.

Data storage

No additional Data is stored.

Economic considerations

Are there any specific costs associated with your solution? If so, are these costs fixed (e.g. the cost of building a lounge), or are they variable based on usage (e.g. no. of API calls)?	N/A
Does Expensify have the ability to spread payments out over time? And is there a cost associated with doing so?	N/A

Are there aspects of the engagement that enable Expensify to only pay once a certain task is completed (e.g. onboarded a company and a billing card is on file)? N/A

How long will Expensify be committed to these costs? What risks might be associated with this length of commitment? N/A

On the flip side of all of this, what does Expensify stand to gain via your solution (e.g. interchange, active user revenue, etc.)? N/A

Accounting Implications

Does this require a new vendor? No
Vendor Name:
Vendor Contact Email:
Service Period:
Estimate spend:
Invoice Frequency: *Daily*
Expense Category and Department:
Require 1099: *No*
Is this related to a physical asset (lease, equipment, etc.)? *No*

Does this change relate to revenue, discounts, expenses, commissions, bonuses, active users or any other inward, outward or internal movement of funds? Do we need to start counting these revenue generating users as *Paid Members?* How is the accounting team addressing this change? Unknown

Does this change require moving money through Stripe, ECard, our ACH system or any bank account? Unknown

*Note to doc authors: Please make sure to **get at least one review from someone on the accounting team before moving to the detailed portion**. See related SO here. Thanks!*

Reviewed By

2023-04-24 CJ

If applicable, link the disclosure document here.

Legal and Compliance considerations

Does this require an update to ToS and/or privacy policy?	No
Does this justify any additional controls in our regular SOC audit?	No
Does this impact PCI?	No
Are there any possible trademark considerations?	No
Could anything about your design be patented?	No
Will you be working with a new vendor and signing a contract?	No

Risk Assessment

We need to evaluate every project and every proposed vendor for potential risk.

For every project that requires a code/hardware change:

- Create a GH with the title `projectName Risk Assessment`. Apply the labels `Task`, `Ring1`, `Compliance`, and an appropriate KSv2 option.
- Include a link to your design doc and any related research or notes you've already compiled.
- Infra will then follow this process to assess information change risk.

https://github.com/Expensify/Expensify/issues/277613

Note to doc authors: Please make sure to ***get at least one review for each section before moving to the detailed portion****.*

Out of scope considerations

- In the first version of this project, we will only support clicking on the App Icon to initiate a share (as opposed to being able to click on specific recent chats).

Alternate solutions

- We originally designed to follow the platform conventions for sharing (original design document), which would have the Android users directed to the main app to share, while iOS users would interact with a separate share extension that closes once the share is complete. After feedback on the detailed design, we decided to change to unify both platforms, which will increase consistency across platforms, as well as being simpler to implement. (Slack thread)

High-level overview reviewed by

Authors:

If you've made it this far in your design doc, now is the time to pause and ensure you've added your project to the CAP Sheet.

Please make sure to **get at least two reviews from each G&R tier before moving on** to the detailed portion. Please follow this SO to guarantee reviews by applying the DesignDocReview label to your tracking issue.

Reviewers:
After you have thoroughly reviewed this doc, add your name and date in the section that corresponds to your Growth and Recognition tier.

Expensifiers + Graduates
2023-11-21 - Danny M - This is going to be dope!
2023-11-22 - Chirag
2023-11-23 - JLi
2023-11-27 - Amy

Project Managers
2023-11-27 - Stites

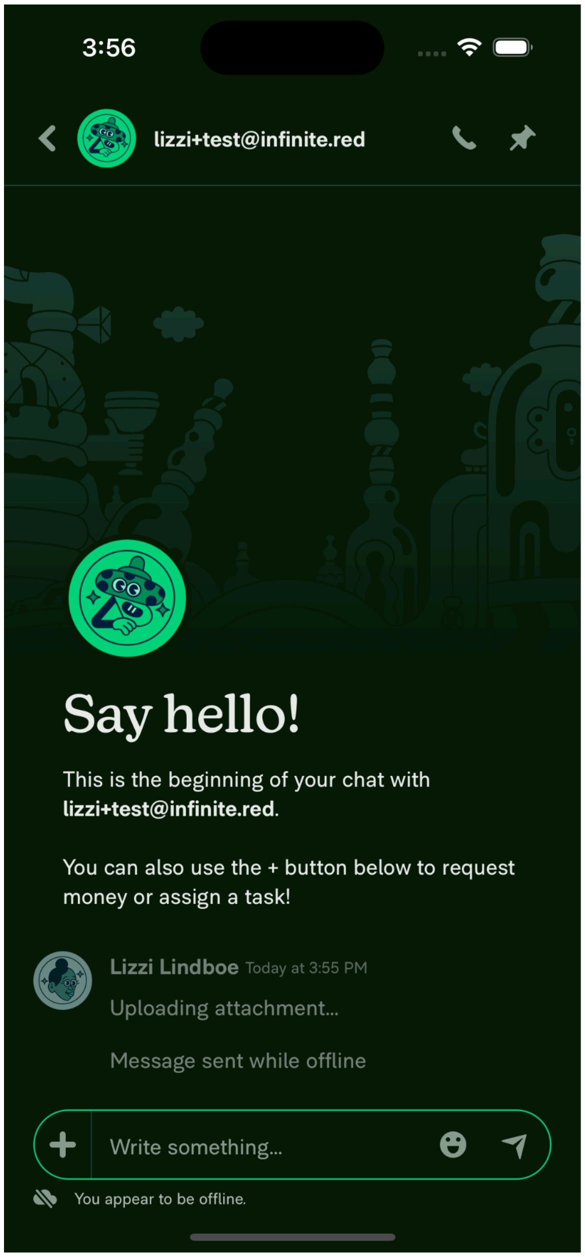
Product Managers
2023-11-20 - John L
2023-11-24 - Tom
2023-11-29 - Conor P, looking forward to getting this one in the wild
2023-12-04 - Robert C.

Generalists
2023-11-21 Tim Golen
2023-11-21 puneet - great!
2024-01-27 - Jason Mills - Looking good, a couple of minor questions related to hybrid app and newdot video player.
2024-03-02 - @dbarrett - Looks great!

Offline support

Users will still be able to share offline. Attachments and money requests will continue to follow the same offline patterns whether their creation is initiated from within the app or from a share attempt.

For example, when looking at the conversation that they shared to, users who have remained offline since sharing will see the same UI as when sending a message or attachment while offline otherwise:



Say hello!

This is the beginning of your chat with **lizzi+test@infinite.red**.

You can also use the + button below to request money or assign a task!

Lizzi Lindboe Today at 3:55 PM

Uploading attachment...

Message sent while offline

On new screens, we will use this same "You appear to be offline" indicator on the bottom of the page as well.

Detailed background

Terminology

Bundle ID - the unique identifier associated with an iOS app. Takes the form of a reverse URL. New Expensify's bundle ID is `com.chat.expensify.chat`. Each "flavor" of New

Expensify has a unique bundle ID so it can be identified as a unique app. For example, the development app's bundle ID is `com.expensify.chat.dev`.
Codegen - This is a tool created by Meta to generate native interfaces from JavaScript static types, using TypeScript or Flow.
New Architecture - A recent initiative from Meta to alter the way native code interfaces with JavaScript. The New Expensify app is close to fully migrating to the New Architecture mode.
Turbo Module - A Turbo Module is a native module that supports the New Architecture. It cannot include native components, only methods and events.

Native configuration

In order to make an app possible to share to, we have to do some configuration on the native side.

Content types

Both iOS and Android have systems for configuring what types of data can be shared to the app. This controls whether New Expensify will show in the system share menu.

iOS requires configuring an `NSExtensionActivationRule`. There are presets for certain types, and iOS also allows custom predicates to decide if a file or other data is supported.

Android is configured using a list of MIME types, where wildcards are also allowed. For example, `image/jpeg`, `image/*`, and `*/*` are all valid items in the configuration list.

Android Send Intents

Android uses "intents" to message requests between different apps and their components. To receive shared data from another app, we configure our app to respond to the `SEND` intent with an "intent filter".

iOS Share Extensions

In order to be able to share into an iOS app, that app has to implement a specific type of app extension, called a share extension, which is a completely separate executable from the main app. (This is in contrast to Android, where data can be shared directly to the app).

Share extensions were designed to be quick to spin up, send data off, i.e., to a web API, and then exit. They were not designed to send data directly to the local app, although this is possible by deep linking into the main app, and sharing data between the share extension and the main app via an app group, explained below.

Share extensions are a separate executable

Share extensions, like all app extensions, are built as a separate executable. This means that, by default, they don't share code or data with the main application. But there are mechanisms for sharing information between the two, such as app group directories (described below), and the NSUserDefaults API for smaller amounts of user data.

Share extensions should be quick to open

It's important that share extensions open quickly, since the user is performing a quick task. Since we won't be loading any UI in the share extension itself, this won't be an issue.

Detailed implementation of the solution

Libraries

React-native-share-menu (RNSM)

The react-native-share-menu library is the best fit for our needs of any existing tools; it has existing support and documentation for customizing iOS share extension behavior and passing data from the share extension to the main app, and tools for handling incoming share data to the Android app.

We were able to get it working with a few patches. However, it's slightly out of date and hasn't been maintained consistently. For example, there's this open issue due to an API change from react-native. Expensify has adopted the project, and Infinite Red has been contributing PRs to get the library up-to-date.

We would also like to update the library with more features to support the desired sharing workflow in New Expensify. See the react-native-share-menu updates section for details.

Javascript layer

Screens

All screens in this flow will:

- have a header with a back button,
- be titled "Share to Expensify", except the MoneyRequestConfirmPage, which will maintain its usual titles ("Manual" or "Split", depending on whether a single participant or multiple are selected, respectively).
- show the offline indicator at the bottom of the page when the user is offline.

ShareModalStackNavigator

We'll add a `ShareModalStackNavigator` to contain the screens for this flow. This will contain:

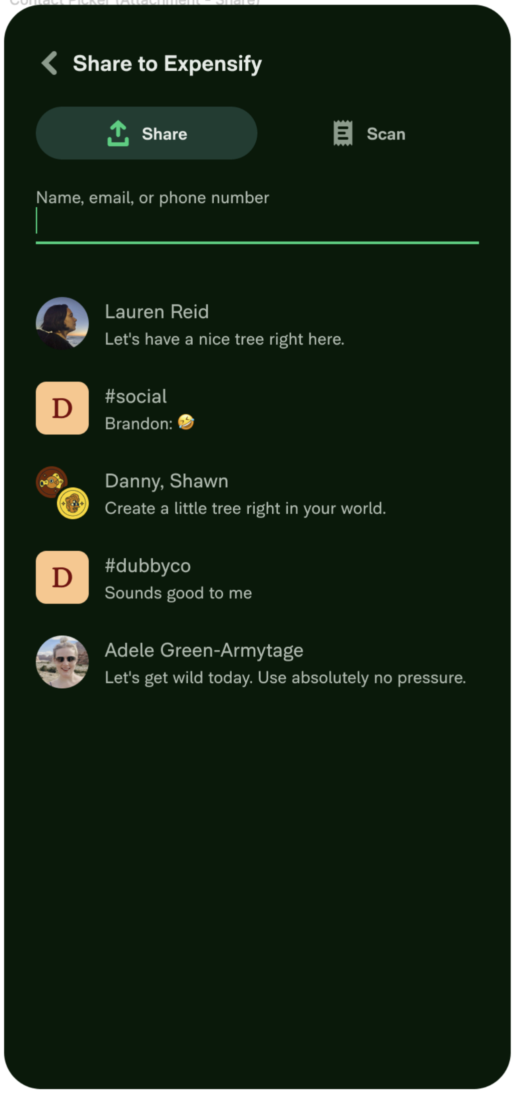
1. the Share entry point (`ShareRootPage`), where the user selects where to share to,
2. the new screen to compose Share messages on (`ShareComposeMessagePage`)

3. a duplicated `MoneyRequestConfirmPage` for use in this modal stack.

```javascript
const ShareModalStackNavigator = createModalStackNavigator({
  Share_Root: () => require('../../../pages/ShareRootPage').default,
  Share_Message: () => require('../../../pages/ShareComposeMessagePage').default,
  Share_Scan_Confirm: () => (`../../../MoneyRequestConfirmPage`).default,
});
```

ShareRootPage

Contact Picker (Attachment - Share)



Lauren Reid
Let's have a nice tree right here.

#social
Brandon: 🤣

Danny, Shawn
Create a little tree right in your world.

#dubbyco
Sounds good to me

Adele Green-Armytage
Let's get wild today. Use absolutely no pressure.

`ShareRootPage` will contain an `OnyxTabNavigator` to support switching between "Share" and "Scan" selection options:

- "Share" will use the new `ShareSelectChat` component,and

- “Scan” will use the existing **`MoneyRequestParticipantsSelector`** component.

```javascript
<OnyxTabNavigator
  id={CONST.TAB.SHARE_TAB_ID}
  selectedTab={fileIsScannable ? selectedTab : CONST.TAB.SHARE}
  hideTabBar={!fileIsScannable}
  tabBar={({state, navigation, position}) => (
    <TabSelector
      state={state}
      navigation={navigation}
      position={position}
    />
  )}
>
  <TopTab.Screen
    name={CONST.TAB.SHARE}
    component={ShareSelectChat}
  />
  <TopTab.Screen
    name={CONST.TAB.SCAN}
    component={() => (
      <MoneyRequestParticipantsSelector
        participants={iou.participants}
        onAddParticipants={IOU.setMoneyRequestParticipants}
        navigateToRequest={() =>
navigateToScanConfirmationStep(CONST.IOU.TYPE.REQUEST)}
        navigateToSplit={() =>
navigateToScanConfirmationStep(CONST.IOU.TYPE.SPLIT)}
        iouType={CONST.IOU.TYPE.REQUEST}
        isScanRequest
      />
    )}
  />
</OnyxTabNavigator>
```

Mirroring the implementation of the Request Money flow, we’ll store the selected tab under **`` `${ONYXKEYS.COLLECTION.SELECTED_TAB}${CONST.TAB.SHARE_ROOT_TAB_ID}` ``**, which will store and re-use the last-selected tab for the next time the user opens this screen.

If the file type of the shared data cannot be scanned, we do not want to render the tab bar. However, we still want to record that the user went with “Share” during their last use of this screen.

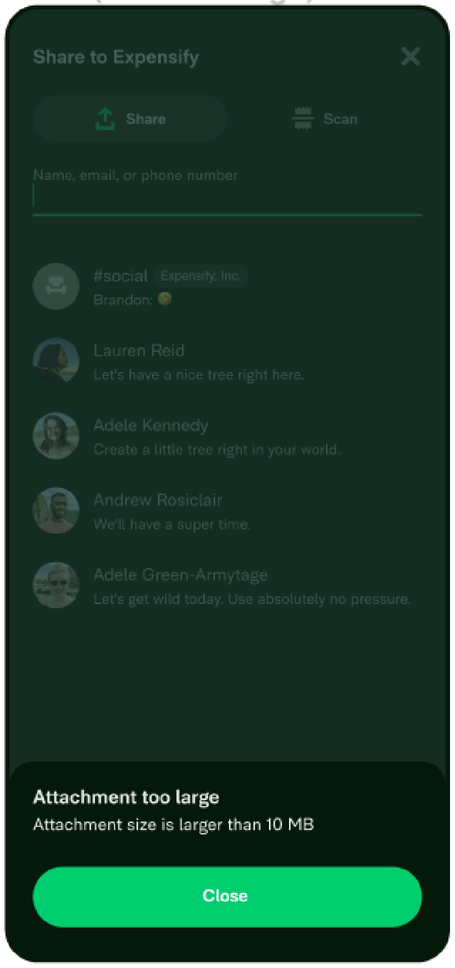
To accomplish this, we’ll update **`OnyxTabNavigator`** with a prop **`hideTabBar`**, to allow us to hide the tab bar, while also storing that the last-used tab was “Share”.

We'll decide when files are compatible with SmartScan by comparing with the existing list of allowed extensions in the CONST file.

Handling files that are too large

We will warn users early when they've tried to share a file above the attachment limit. We'll use the same component that the current attachment picker uses: `ConfirmModal`.

Error (Size too large)

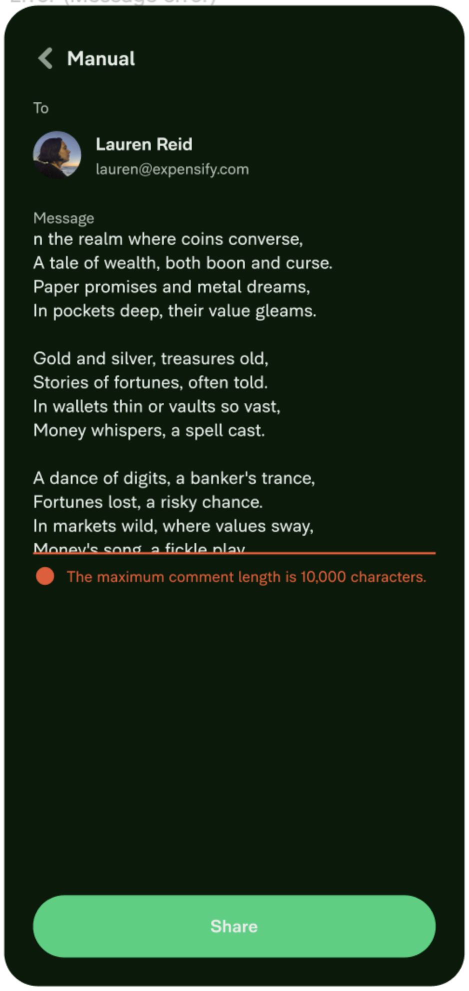

Share to Expensify
Share
Scan
Name, email, or phone number
#social
Expensify, Inc.
Brandon:
Lauren Reid
Let's have a nice tree right here.
Adele Kennedy
Create a little tree right in your world.
Andrew Rosiclair
We'll have a super time.
Adele Green-Armytage
Let's get wild today. Use absolutely no pressure.
Close

Attachment too large

Attachment size is larger than 10 MB

Handling messages that are too long

Error (Message error)



Manual

To

Lauren Reid
lauren@expensify.com

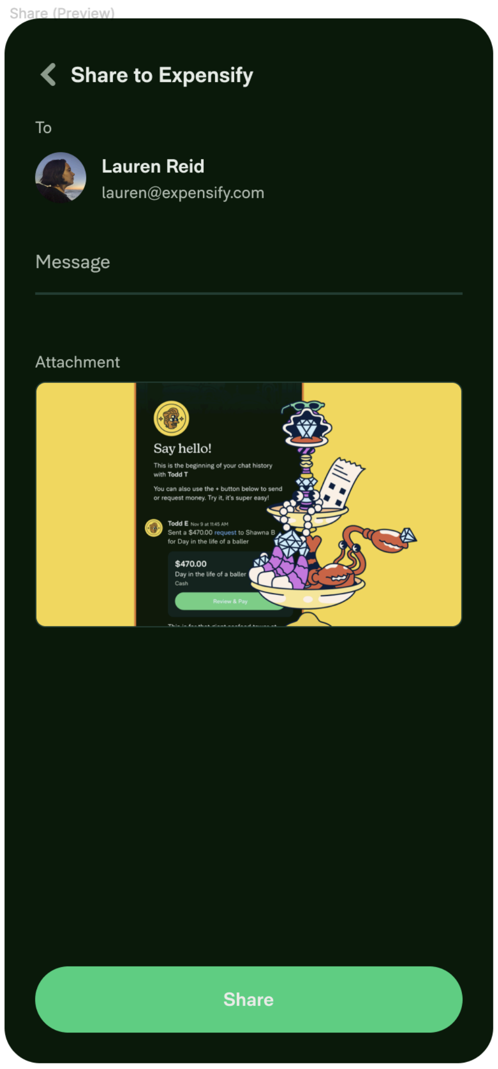
Message
n the realm where coins converse,
A tale of wealth, both boon and curse.
Paper promises and metal dreams,
In pockets deep, their value gleams.

Gold and silver, treasures old,
Stories of fortunes, often told.
In wallets thin or vaults so vast,
Money whispers, a spell cast.

A dance of digits, a banker's trance,
Fortunes lost, a risky chance.
In markets wild, where values sway,
Money's song, a fickle play

● The maximum comment length is 10,000 characters.

ShareComposeMessagePage



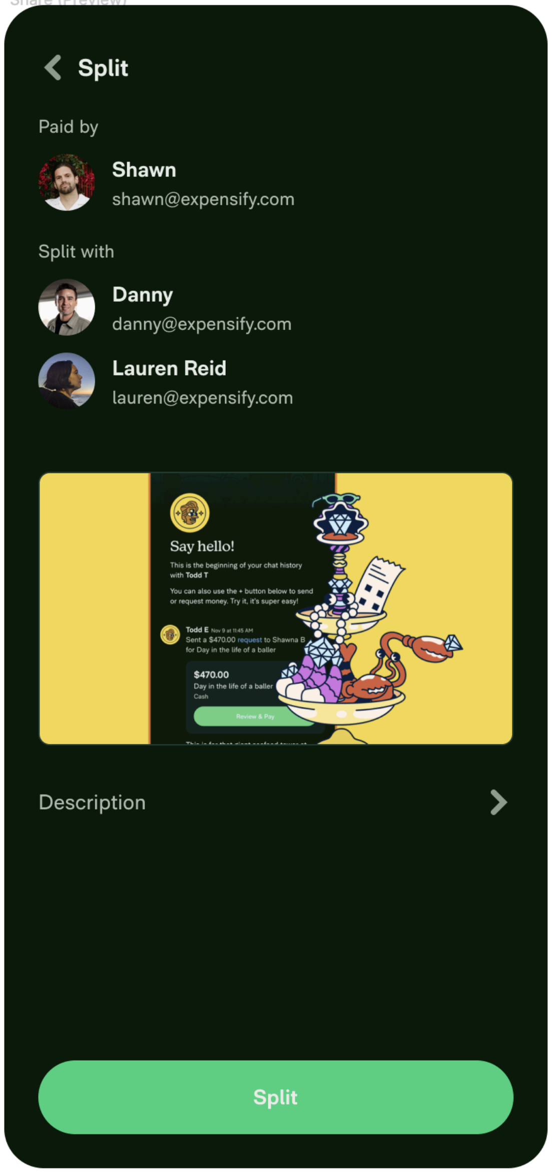
For this new screen, we'll use:

- `OptionRow` to display the conversation preview,
- `FormProvider` with the `TextInput` using `autoGrowHeight`, which we'll autofill if the shared data is text or a link. The TextInput will validate that the message is not longer than 10k characters before it can be submitted (source on limit).
- `AttachmentView`, displayed if the shared data is not text or a link

We'll determine the type of data shared based on the `mimeType` property provided by react-native-share-menu.

MoneyRequestConfirmPage

Share (Preview)



Paid by

Shawn
shawn@expensify.com

Split with

Danny
danny@expensify.com

Lauren Reid
lauren@expensify.com

Say hello!

This is the beginning of your chat history with **Todd T**

You can also use the + button below to send or request money. Try it, it's super easy!

Todd E Nov 9 at 11:45 AM
Sent a $470.00 request to Shawna B for Day in the life of a baller

$470.00
Day in the life of a baller
Cash

We don't anticipate any changes to this existing page, where the user can see the receipt and edit request details before submitting the request. We just need to add it under another route in this modal stack so we can navigate to it properly.

Components

New: ChatSearch / ShareSelectChat

We'll extract the `OptionsSelector` and shared business logic used in the existing `SearchPage` into a generic `ChatSearch` component that can be re-used to search through existing chats. The prop `onSelectChat` will control what the component does when a chat is selected.

To wrap the `ChatSearch` component with specific behavior related to finding a chat to search to, we'll create the `ShareSelectChat` component.

Navigating to the share flow

React-native-share-menu currently guides app developers to set up listeners for share data and then react within the listener when share data is present: in our case, navigate to the Share Root page.

Instead, we'd like to configure react-native-share-menu to deep link directly to the share modal root screen (e.g., on iOS, `new-expensify://share`). This should make behavior more predictable and make troubleshooting any bugs easier (any underlying errors with share data could cause a failure to navigate otherwise, and those errors should be more immediately visible if you're able to see the share screens).

We'll compare this with the previous approach of using a listener to make sure it doesn't introduce new problems. We're hoping it will be more performant, since we'll navigate as soon as possible.

Managing share data

Since share data is ephemeral, we will pass all data we can as route params to screens, which reduces the amount of state cleanup we have to do. We may need to implement a `ShareContextProvider` to manage the initial state passed by react-native-share-menu, depending on the approach we're able to take with navigation.

Native layer

Configuring content types

iOS

Using the `NSExtensionActivationRule`, we will be able to support files, text, and links:

```
Unset
<key>NSExtensionAttributes</key>
<dict>
	<key>NSExtensionActivationRule</key>
	<dict>
		<key>NSExtensionActivationSupportsImageWithMaxCount</key>
		<integer>1</integer>
		<key>NSExtensionActivationSupportsText</key>
		<true/>
		<key>NSExtensionActivationSupportsWebURLWithMaxCount</key>
		<integer>1</integer>
		<key>NSExtensionActivationSupportsFileWithMaxCount</key>
		<integer>1</integer>
```

```
        </dict>
</dict>
```

Android

We set Android to accept all file types for the `SEND` intent, which covers share requests of a single file or text string:

```
Unset
<activity
  ...
  android:documentLaunchMode="never">
  ...
  <intent-filter>
    <action android:name="android.intent.action.SEND" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="*/*" />
  </intent-filter>

</activity>
```

iOS implementation details

iOS needs a good deal more implementation: configuring the share extension to behave how we want, and then enabling it to open the main app and share data with it.

Setting up the Share Extension

Largely following the guide from the basic iOS setup, and then for the native share extension view.

In brief, we:

1. Create a new share app extension, using XCode, and configure, similar to the main app:
   a. Code signing
   b. The Podfile needs the react-native-share-menu pod added to the share extension started.
2. Add react-native-share-menu's `ShareViewController` to the share extension.
3. Create an App Group that contains the main app and the share extension so that they can share data.

Skipping the share extension UI

The default behavior of react-native-share-menu when **not** using a custom React Native UI inside the share extension is to show an operating system modal with a file preview, a text input and a "Post" button, based on `SLComposeServiceViewController` (docs link).

To avoid showing any UI, we modify its `ShareViewController` in the following ways:

1. Inherit from `UIViewController` instead of `SLComposeServiceViewController`, so we can render no UI instead of the iOS share modal
2. Immediately link to the main app during `viewDidLoad`

We'll add this as a documented option in react-native-share-menu as well.

Managing temporary files on iOS

Normally, when sharing on iOS, the share extension receives a temporary copy of the file that only lasts as long as the share extension stays open. The main app is not granted access to this file, and as we're sharing from the main app, we also need the file to last longer than the share extension UI does. To get around this, within the share extension, react-native-share-menu makes a copy of the file that the main app can access. But it never cleans up those files.

In New Expensify, we need to ensure the copied file exists long enough to support a pending API request, but is cleaned up after that request succeeds or fails.

To accomplish this, we'll add a new Onyx key, `TEMP_FILES_TO_DELETE`, which will track files that need cleaning up. We'll populate this key with files to delete in `successData` and `failureData` of `Report.addActions` (which contains the logic of `Report.addAttachments`). We can encapsulate this logic in a utility function, `cleanUpActions`:

```JavaScript
// src/libs/actions/Share.ios.js

// constructs share-specific data for use in `addAttachment`
const cleanUpActions = (file) => {
    if (!file || !file.source.includes(appGroupPath)) return [];
    return [
        {
            onyxMethod: Onyx.METHOD.MERGE,
            key: ONYXKEYS.TEMP_FILES_TO_DELETE,
            value: [file],
        },
    ];
};

// src/libs/actions/Report.js

function addActions(reportID, text = '', file) {
  ...
  const successData = [{
```

```
            onyxMethod: Onyx.METHOD.MERGE,
            key: `${ONYXKEYS.COLLECTION.REPORT_ACTIONS}${reportID}`,
            value: _.mapObject(optimisticReportActions, () => ({pendingAction:
null})),
        },
        ...cleanUpActions(file),
    ];
    ...
}
```

And then in an Onyx `connect` call, we can watch for updates to this key and call a method in react-native-share-menu, `deleteTempCopy`, to clean up the file:

```JavaScript
let isCleaningUpTempFiles = false;
Onyx.connect({
  key: ONYXKEYS.TEMP_FILES_TO_DELETE,
  callback: (val) => {
    if (!val || isCleaningUpTempFiles) return;
    isCleaningUpTempFiles = true;
    val.forEach((file) => {
      ShareMenu.deleteTempCopy(file);
    });
   Onyx.set(ONYXKEYS.TEMP_FILES_TO_DELETE, []);
   isCleaningUpTempFiles = false;
   },
});
```

React-native-share-menu updates

React-native-share-menu is the best library available for sharing to a react native app. That said, it's in need of maintenance: both to catch up to current React Native standards, as well as support features New Expensify needs. As a result, Expensify has now adopted the library (GitHub issue).

Below, we'll outline the changes we need to make to support share in New Expensify. All of these changes will come with testing and changelog updates so it's easy to cut a release once we're done.

Get the library up-to-date

We've already merged some simple fixes to the library to make it compatible with current React Native:

1. `jsBundleUrl` arguments changed in recent React Native versions (issue link).
2. Update Android `compileSdkVersion` (issue link).

3. `ShareMenuModule.java` needs to implement NativeEventEmitter methods to avoid warnings (issue link).

There's a few more items we'll have to add to get everything up-to-date:

Update the example project

To ensure that all updates to react-native-share-menu are simple to test, we'll update the repository's existing example project to the latest react-native version.

Add a privacy manifest for iOS

Starting with Xcode 15, Apple is requiring that libraries and apps that use specific features register their "reason" for doing so in a new privacy manifest. This includes UserDefaults, which react-native-share-menu uses to store shared data to pass to the main app. Its usage of UserDefaults is fully compliant with the allowed "reason" ("Declare this reason to access user defaults to read and write information that is only accessible to the app itself"), so we only need to register the reason and we're done.

Support the New Architecture

We were originally considering punting on this, but it sounds like New Architecture support is close, and we shouldn't introduce new libraries that don't support it (Slack conversation).

This will fortunately be straightforward. The app has no native components, so to support the New Architecture, we just have to re-write it as a Turbo Module. The steps are:

1. Define a Codegen spec in Typescript that the native modules will conform to, and add Codegen configuration to the project. This may require changing the data structures used by native methods to accommodate Codegen.
2. Update the library's podspec to New Architecture dependencies
3. Update the native module implementation to use the new specs and conform to the New Architecture APIs (currently best explained by the New Turbo Module guide: https://reactnative.dev/docs/the-new-architecture/pillars-turbomodules).

Fixes

MIME Type detection (iOS)

We've found that on iOS, when sharing from the Files app, react-native-share-menu was always reporting MIME Types of `text/plain` to JavaScript, regardless of the actual file type.

iOS uses Uniform Type Identifiers instead of MIME Types, and we have to match against those identifiers and translate them into MIME Types. We found the root of the bug and have a fix ready to go (GitHub link).

App group configuration (iOS)

RNSM incorrectly assumes that all app groups must match the pattern `group.<main.app.bundle.id>`. This isn't necessarily the case, and has caused pain points while prototyping since different build schemes of Expensify have different bundle ID structures. We'll update the library to instead have the app group be directly configured in Info.plist, instead of configuring it with the host app's bundle ID.

viewDelegate crash (iOS)

There have been intermittent crashes on iOS due to issues with the view delegate state handling. We have a basic fix for this, but there are related issues we should look into:

1. Sharing multiple times with `continueInApp` crashes (issue)
2. `viewDidDisappear` is not consistently called, so extension cleanup does not happen consistently and can lead to memory issues (issue)

Features

Manage copied files (iOS)

As previously mentioned, when navigating to the main app to share on iOS, RNSM currently copies files but never deletes them. We'll add a `deleteTempCopy` method that accepts a file name to allow apps to decide when files should be cleaned up.

"Skip share extension" support (iOS)

As previously mentioned, to support skipping the share extension UI entirely, we need to modify the existing RNSM view controller. We'll add this use case as a third documented option to the project, which we'll use in the app.

Publish it

Once we've got all our changes in, we'll need to publish a new major version of the library. The NPM publishing step will need to be handled by the Expensify team.

Manual tests

All of these tests should be performed on both Android and iOS.

SmartScan-compatible files

Note: these file extensions are compatible with SmartScan. Refer to this list when test instructions say "find a file compatible with SmartScan": 'jpg', 'jpeg', 'gif', 'png', 'pdf', 'htm', 'html', 'text', 'rtf', 'doc', 'tif', 'tiff', 'msword', 'zip', 'xml', 'message'.

User is online and already signed into the Expensify app

Share a single image to New Expensify

1. In another app, find an image compatible with SmartScan (with extension ‘jpg’, ‘jpeg’, ‘gif’, or ‘png’). Choose the share option, then choose New Expensify from the list of apps.
2. The “Share to Expensify” UI will be shown, with a tab bar displaying “Share” and “Scan”. Make sure “Share” is selected.
3. You should see a text input and a list of chats your user account is in. Choose a chat to share with by searching for it in the text box, and selecting it in the list shown beneath.
4. The Compose Message page will be shown. You should see:
    a. The chat the share will be sent to
    b. A text field where you can add an optional message. In this case, it should be empty.
    c. A preview of the image.
    d. A green “Share” button at the bottom of the page.
5. Add a message to the message field.
6. Click “Share”. You should be redirected to the conversation you shared to, and should see the shared image and message in that conversation.

Share a scannable, non-image file to New Expensify

What’s different from the image file test case? There will not be an image preview of the file; instead, there will be an attachment component that displays the file name.

1. In another app, find a file that isn’t an image, but is compatible with SmartScan (e.g., a zip file) to share. Choose the share option, then choose New Expensify from the list of apps.
2. The “Share to Expensify” UI will be shown, with a tab bar displaying “Share” and “Scan”. Make sure “Share” is selected.
3. You should see a text input and a list of chats your user account is in. Choose a chat to share with by searching for it in the text box, and selecting it in the list shown beneath.
4. The Compose Message page will be shown. You should see:
    a. The chat the share will be sent to
    b. A text field where you can add an optional message. In this case, it should be empty.
    c. A file attachment component showing a paper clip icon and the file’s name.
    d. A green “Share” button at the bottom of the page.
5. Add a message to the message field.
6. Click “Share”. You should be redirected to the conversation you shared to, and should see the shared image and message in that conversation.

Share a single, non-scannable file to New Expensify

What's different from the image file test case? There will not be an image preview of the file; instead, there will be an attachment component that displays the file name.

1. In another app, find a file that isn't an image and isn't compatible with SmartScan (e.g., an mp3) to share. Choose the share option, then choose New Expensify from the list of apps.
2. The "Share to Expensify" UI will be shown. The tab bar with "Scan" and "Share" should **not** be shown.
3. You should see a text input and a list of chats your user account is in. Choose a chat to share with by searching for it in the text box, and selecting it in the list shown beneath.
4. The Compose Message page will be shown. You should see:
    a. The chat the share will be sent to
    b. A text field where you can add an optional message. In this case, it should be empty.
    c. A file attachment component showing a paper clip icon and the file's name.
    d. A green "Share" button at the bottom of the page.
5. Add a message to the message field.
6. Click "Share". You should be redirected to the conversation you shared to, and should see the shared image and message in that conversation.

Share text to New Expensify

What's different from the image or file test cases? There will be no preview or attachment component, and the text will be automatically inserted in the message field.

1. In another app, highlight and share some text, then choose New Expensify from the list of apps to share to.
2. The "Share to Expensify" UI will be shown.There should be no tab bar with "Share" and "Scan" as options. You can only proceed by sharing to a chat.
3. You should see a text input and a list of chats your user account is in. Choose a chat to share with by searching for it in the text box, and selecting it in the list shown beneath.
4. The Compose Message page will be shown. You should see:
    a. The chat the share will be sent to
    b. A text field, pre-filled with the text that was shared. You should be able to edit this text.
    c. A green "Share" button at the bottom of the page.
5. Edit the message in the message field.
6. Click "Share". You should be redirected to the conversation you shared to, and should see the message you wrote in that conversation.

Share a link to New Expensify

This test case should have the same behavior as sharing text. In the message field, the link will not be clickable, highlighted, or previewed, just treated as text.

Input text into message field that is too long

1. Share text over 10k characters to New Expensify.
2. The “Share to Expensify” UI will be shown.There should be no tab bar with “Share” and “Scan” as options. You can only proceed by sharing to a chat.
3. You should see a text input and a list of chats your user account is in. Choose a chat to share with by searching for it in the text box, and selecting it in the list shown beneath.
4. The Compose Message page will be shown. You should see:
    a. The chat the share will be sent to
    b. A text field, pre-filled with the text that was shared.
    c. A green “Share” button at the bottom of the page.
5. Click “Share”. The message field should be highlighted red and inform you that the message is too long to send.
6. Edit down the message to under 10k characters, and press “Share” again. The message should be sent to the chat you selected.

Try to share multiple files to New Expensify

1. Go to an app that allows selecting and sharing of multiple files (e.g., a photo gallery app).
2. Select and share multiple files.
3. You should not see New Expensify listed as an option among the applications in the system share dialog.

Scan an image of a receipt

1. In another app, find an image of a receipt to share. Choose the share option, then choose New Expensify from the list of apps.
2. The “Share to Expensify” UI will be shown, with a tab bar displaying “Share” and “Scan”. Make sure “Scan” is selected.
3. You should see:
    a. a text input
    b. a list containing people and workspaces, each with a “Split” button
4. Choose a person to send the request to by searching for it in the text box, and selecting it in the list shown beneath.
5. You should now see the Request Confirm page, with a preview of the receipt image, and the option to edit request fields. Update fields as desired, and create the request by pressing the “Request” button.
6. You should be redirected to the report for the request you just created, with a preview of the request shown in the chat.

Scan a PDF of a receipt

Repeat the image test, but share a PDF instead of an image of a receipt.

Split a scanned request

1. In another app, find an image of a receipt to share. Choose the share option, then choose New Expensify from the list of apps.
2. The “Share to Expensify” UI will be shown, with a tab bar displaying “Share” and “Scan”. Make sure “Scan” is selected.
3. You should see:
    a. a text input
    b. a list containing people and workspaces, each with a “Split” button
4. Choose people to split between by pressing the “Split” button for multiple people, and then press the large green “Split” button at the bottom of the screen.
    a. Do not choose to Split with a workspace. This appears to be an option but does not work. This is an existing bug (GitHub issue).
5. You should now see the Request Confirm page, with a preview of the receipt image, and the option to edit request fields. Update fields as desired, and create the request by pressing the “Request” button.
6. You should be redirected to the report for the request you just created, with a preview of the request shown in the chat.

Backgrounding share and resuming

1. Share a file or text to New Expensify. Continue with the “Share” tab and choose a chat to share to.
2. On the screen with the message input, add some text to the message input.
3. Open another app besides New Expensify (i.e., background New Expensify. Don’t kill it.)
4. Re-open New Expensify. You should see the message input with the same text you had before the app was backgrounded.
5. Send the message. You should see the message that you wrote sent to the chat.

Cancel flow before sharing

Using back button in header

1. Share a file to New Expensify.
2. On the chat selection screen, press the back button in the header.
3. The result depends on whether the app was previously opened:
    a. If the New Expensify app was already open and running in the background, the back button will navigate to the screen that was visible prior to the share attempt
    b. If the New Expensify app was not in the background prior to the share attempt, the app will show the home screen (i.e., the most recently viewed chat in the app).

Abandon share attempt and try again with a new file

1. Share a file to New Expensify.
2. On the chat selection screen, choose a chat to share to.
3. Navigate fully back out of the app. How to do this depends on the platform:
    a. On iOS, press the small “breadcrumb” button in the status bar with the name of the app you shared from
    b. On Android, use the back button/gesture until you exit New Expensify and are returned to the app you shared from
4. Choose a **different** file to share to New Expensify. Follow the usual share steps, and verify that the new file is what is previewed and sent.

Share a file over 24MB

1. In another app, select a file over 24MB. Choose the share option, then choose New Expensify from the list of apps.
2. The “Share to Expensify” UI will be shown. In addition, a bottom sheet modal will appear, saying that the file is too large to share.
3. Press the button in the modal to return to the New Expensify app. You should be navigated to a report.

Test sticky tabs

1. Share anything into a chat in New Expensify using the “Share” tab.
2. Next, share an image file to New Expensify. When the app loads, it should focus the “Share” tab.
3. Instead, click the “Scan” tab and add the image to a request. Finish creating the request.
4. Next, share another image file into New Expensify. When the app loads, it should now focus the “Scan” tab, since the last action you completed was scanning.

User is offline and already signed into the Expensify app

User experience should be the same for offline users, except they will see indicators that they are offline at the bottom of each screen.

When looking at the conversation that they shared to, users who have remained offline will see the same UI as when sending a message or attachment while offline otherwise.

Test offline share

1. Turn on airplane mode on phone
2. Choose a file to share, and share to New Expensify. On the screen where you select a chat to share to, you should see an Offline indicator at the bottom of the page.
3. Choose a chat to share the file to

4. You should see the Compose Message page with an Offline indicator at the bottom of the page. Add a message and press send.
5. While offline, navigate to the chat you shared to
6. The offline message pending state should be shown
7. Turn off airplane mode on phone
8. The message should be sent, and attachment and message should become visible in the conversation.
9. Verify the message was sent correctly by checking the chat on another device.

Test offline share after killing app, iOS

Note: this test is useful on iOS to ensure the temp file cleanup doesn't occur prematurely.

1. On iOS, turn on airplane mode on phone
2. Choose a file to share, and share to New Expensify. On the screen where you select a chat to share to, you should see an Offline indicator at the bottom of the page.
3. Choose a chat to share the file to
4. You should see the Compose Message page with an Offline indicator at the bottom of the page. Add a message and press send.
5. While still offline, swipe up to open the app switcher, and swipe New Expensify up to kill the app
6. Re-open New Expensify and navigate to the chat you shared to
7. The offline message pending state should be shown
8. Turn off airplane mode on phone
9. The message should be sent, and attachment and message should become visible in the conversation
10. Verify the message was sent correctly by checking the chat on another device.

Test offline scanning

1. Turn on airplane mode on phone
2. Choose a receipt image to share, and share to New Expensify
3. The "Share to Expensify" UI will be shown, with a tab bar displaying "Share" and "Scan". "Share" should be selected by default. Choose "Scan" instead.
4. You should see:
    a. a text input
    b. a list containing people and workspaces, each with a "Split" button
5. Choose a person to send the request to by searching for it in the text box, and selecting it in the list shown beneath.
6. You should now see the Request Confirm page, with a preview of the receipt image, and the option to edit request fields. Update fields as desired, and create the request by pressing the "Request" button.
7. You should be redirected to the report for the request you just created, with a pending preview of the request shown in the chat.
8. Turn off airplane mode. The request should now be fully created.
9. Verify the request was sent correctly by checking the chat on another device.

User is not signed into the Expensify app

1. Log out in the New Expensify app if already signed in.
2. Go to another app and try to share a file to New Expensify
3. The login form will be displayed. Sign in.
4. The sign in process continues as normal. The user is not redirected to the share flow.
5. Go back to the other app and try one of the share flows listed above. They should work as described.

Automated tests

We do not plan on adding automated tests. The share operating system modal on iOS cannot be operated by automated testing tools on a simulator, and when running on a real device, the only testing option is to use XCUITest, which currently has issues with deeply nested Views, as in a React Native app.

Alternate solutions (detailed)

Alternative libraries

There is another library react-native-share-extension to use RN in a share extension but it's even less maintained than react-native-share-menu.

We could also write a new library, but despite the maintenance status of react-native-share-menu it works well with the latest RN with a few small patches. It makes the most sense to build on it as a foundation.

Supporting group creation for share to chat

Originally, we were supporting sharing to chats with people and groups, including creating new groups, by reusing the "New Group" page. This did not support sharing to rooms or other chats, however. From pre-design, the preferred solution was to start with the "Chat Search" functionality, which allows sharing to any existing chat, as well as creating new DMs. (Slack thread)

Custom React Native UI in the iOS share extension

React-native-share-menu supports embedding a custom React Native UI inside of the share extension. We originally wrote a design document to support this flow. This approach added implementation complexity, since the share extension is a separate process, and the app's storage layer was not implemented to support multiple threads writing to it.

Given that this approach introduced platform differences and implementation complexity, the feedback was to instead use an approach that skips the share extension and goes directly to the main app. (Slack thread)

System requirements

We'll need to work with the internal Expensify team on how we want to handle react-native-share-menu updates and the new release of the library, as the Expensify team will have control of the repo and the NPM account.

Plan of action / Rollout plan

These are the options we considered for breaking up the work:

1. **Beta flags.** We can't use beta flags to hide the ability to share in the app, as that's controlled by native configuration.
2. **Smaller PRs without the native configuration added.** This will likely be more trouble than it's worth, because development and reviewing PRs would require re-adding complex native configuration each time, which is easy to mess up (and hard to automate, because most of the relevant files are managed by Xcode).
3. **Ship for one platform at a time.** Since Android is very simple compared to iOS, we could ship Android alone first, but it's not clear that it would be an improvement. The problem with shipping Android first is that iOS requires us to make enough changes to their shared API that we'd end up redoing a significant chunk of work, and would need to test Android again at the end of the process. It's fairly trivial to add Android on at the end of the process.
4. **Split up "Share" and "Scan".** This would not impact development time much, but could reduce complexity in PR review and testing. Overall it's not clear that it would be worth it, as most of the work will be in foundational code, not the features themselves.

As a result, we decided that a feature branch for Native Share would be the best approach. These are the steps:

1. We will implement this feature on a feature branch on the Infinite Red fork of the project.
2. Simultaneously, we will plan and implement the changes necessary for react-native-share-menu.
3. We'll do a major version release of react-native-share-menu once all the changes are in. We'll need Expensify's help for this.
4. Once the new version react-native-share-menu is released, we'll incorporate it into the feature branch and make the PR against New Expensify.

Communication Plan

A thorough Communication Plan ensures our internal (help docs, Zingtree) and external resources (Community posts and marketing items) are completed on time when the product changes. The leader of this design doc is responsible for the below tasks.

1. **Create a Launch/Guides task GH** using this template, a #launch team member will be automatically assigned after you create the GH. The Launch team member will be responsible for organizing any sales campaign related to the project.
2. **Create an External Resource GH** using this template, a #Resource-Management team member will be automatically assigned after you create the GH and responsible for organizing the updates.
3. **Create an Internal Resource GH** using this template, a #Resource-Management team member will be automatically assigned after you create the GH and responsible for organizing the updates.
    a. Reach out in #resource-management if you have any questions about making these GHs.
4. **Marketing Comms Chore GH** using this template a #marketing team member will be automatically assigned after you create the GH and responsible for organizing the updates
5. **Wrap up CAP Sheet** entries. As soon as your project(s) go live, please note your Launch Date(s) on the CAP Sheet.

Detailed implementation reviewed by

Authors:
Please make sure to **get the correct number of reviews from each G&R tier** before beginning to implement the detailed portion. Please follow this SO to guarantee reviews by applying the DesignDocReview label to your tracking issue. Thanks!

Reviewers:
After you have thoroughly reviewed this doc, add your name and date in the section that corresponds to your Growth and Recognition tier.

Expensifiers + Graduates (Need at least 2)
2023-11-22 - Chirag
2023-11-27 - Amy
2023-12-01 - Ariel

Project Managers (Need at least 2)
2023-11-27 - Stites

2023-12-06 - Greeny

Product Managers or Generalists (Need at least 4 engineers and 4 non-engineers)
2023-11-20 - John L
2023-11-21 Puneet - seems like a solid and thorough plan to me. Let's do it!
2023-11-24 - Tom
2023-11-28 Steph E
2023-11-29 - Conor P
2023-12-04 - Robert C.

Project wrap up

Once the project is finished, update the CAP Sheet project status and launch date. Then, complete this section and email its contents to strategy@. What went well? What could we have done better? What did we learn?