

unREST among the docs

What are we going to talk about

- Designing the docs
- For a better developer experience

Who is this discussion for

- Technical writers - because they are faced with such problems and dilemmas almost daily
- Designers - who are working on technical information systems
- Developers - who want to be better at writing systems

whoami

- Akshay Bhalotia
- Professional interests
 - Payment systems
 - Developer SaaS
- Personal interests
 - Cats - proud dad to 2 cats
 - Boardgames
 - Mechanical keyboards

Find me here    

But why would you listen to me

I've seen some 💩 in my life

- An iOS Developer at an outsourcing boutique ([Optimus Information, Inc.](#)) and a FinTech API startup ([Razorpay](#))
- An account manager at tech events and media company ([HasGeek](#))
- Support and solutions engineer at another FinTech API startup ([Setu](#))
- Developer relations manager at a real-time video API startup ([Dyte](#))
- And now, a product manager at a data gateway API company ([Phyllo](#))

Problem

Docs are hard

- Primary function: **give info**
- But
 - in an easily readable manner
 - in an easily findable manner
 - help with the next steps
 - give enough but not too much
 - expect returning users

What is harder

- Documenting frontend SDKs
- Documenting UI features
- With version control
- Feature-based releases, without a release cycle
- Breaking changes

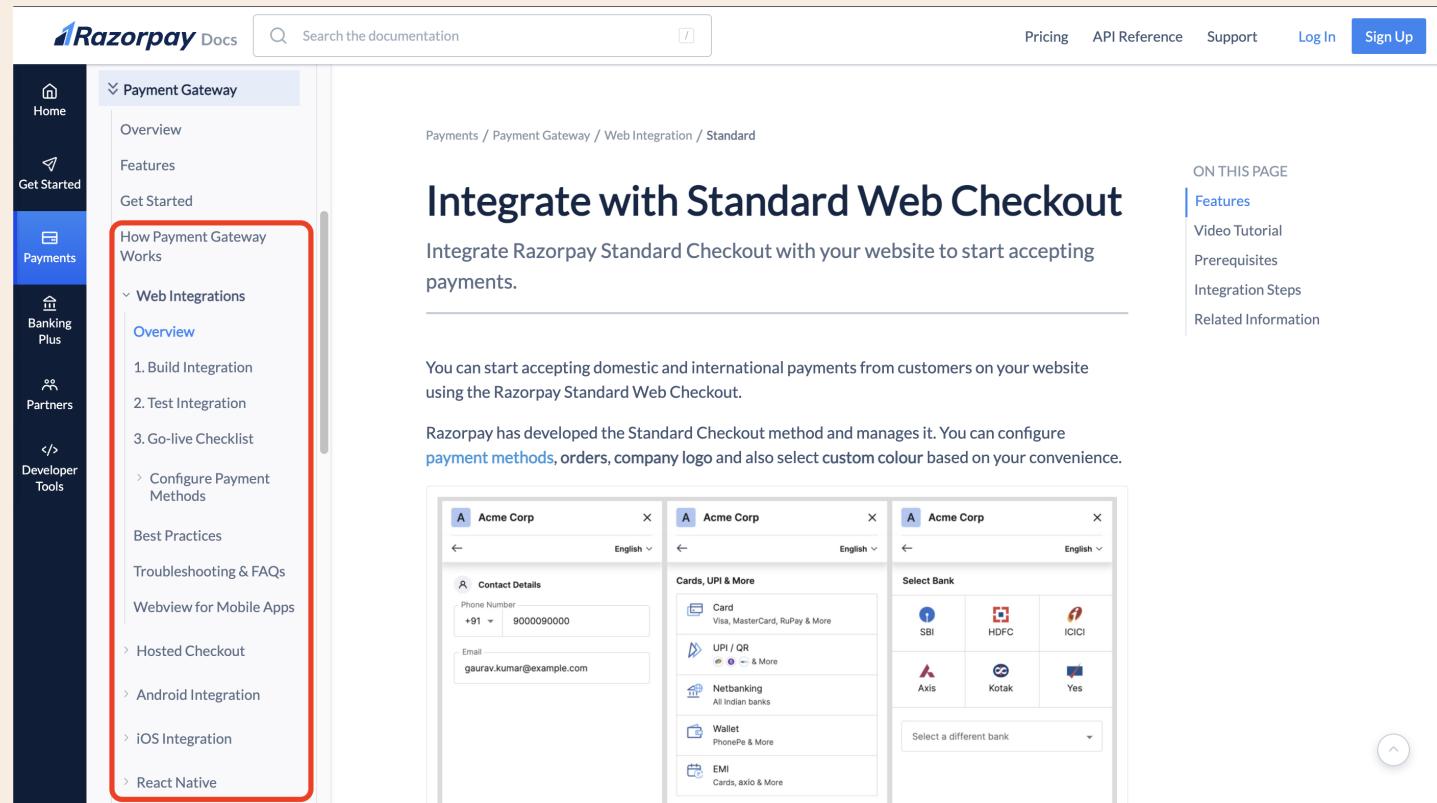
**Let's see a few examples of how I've tried tackling
some of these challenges**

Razorpay - Payment APIs for India

The screenshot shows the Razorpay Documentation website. The sidebar on the left is dark blue with white icons and text for Home, Get Started, Payments (which is selected and highlighted in blue), Banking Plus, Partners, and Developer Tools. The main content area has a light gray background. At the top, there's a navigation bar with the Razorpay logo, a search bar, and links for Pricing, API Reference, Support, Log In, and Sign Up. Below the navigation is a horizontal menu with tabs: Payments (selected), Banking Plus, Partners, and Developer Tools. The Payments tab's content includes a 'Payments' section with a description and a 'Explore Payments' button, followed by an 'INTEGRATE WITH PAYMENT GATEWAY' section with links for various integration options. A red box highlights the first four integration links: Web Integration, React Native, iOS Integration, and Ecommerce Plugins. To the right of this section is a screenshot of a payment interface showing an order from 'Acme Corp' for ₹ 6,000. The interface includes fields for phone number (+919999999999) and email (gaurav@example.com). It also lists 'PREFERRED PAYMENT METHODS' (Netbanking - ICICI Bank, Netbanking - Yes Bank) and 'CARDS, UPI & MORE' (Card: Visa, MasterCard, RuPay & More). The overall design is clean and modern, using a color palette of blues, whites, and grays.

An intro page helps the reader navigate to their appropriate section.

Razorpay - Payment APIs for India



The screenshot shows the Razorpay Documentation website. The left sidebar has a dark background with white icons and text. It includes links for Home, Get Started, Payments (which is highlighted in blue), Banking Plus, Partners, and Developer Tools. Under the Payments section, there's a dropdown menu for 'Payment Gateway' which is also highlighted in blue. This dropdown contains links for Overview, Features, Get Started, How Payment Gateway Works, Web Integrations (which is expanded and highlighted with a red border), and Best Practices. The 'Web Integrations' section has three sub-links: 1. Build Integration, 2. Test Integration, and 3. Go-live Checklist. Below these are three collapsed sections: 'Configure Payment Methods', 'Best Practices', and 'Troubleshooting & FAQs'. The main content area has a light gray background. At the top, there's a navigation bar with the Razorpay logo, a search bar, and links for Pricing, API Reference, Support, Log In, and Sign Up. The main title 'Integrate with Standard Web Checkout' is displayed in large bold text. Below it, a sub-section title 'Integrate Razorpay Standard Checkout with your website to start accepting payments.' is shown. There are three screenshots of the Razorpay checkout interface: one showing contact details, one showing payment methods like Cards, UPI & More, and another showing a bank selection screen with SBI, HDFC, ICICI, Axis, Kotak, and Yes. To the right of the main content, there's a sidebar titled 'ON THIS PAGE' with links for Features, Video Tutorial, Prerequisites, Integration Steps, and Related Information.

When you click on any of the options, you land on the related overview section, but all the other options are available in the sidebar for easy navigation.

Razorpay - Payment APIs for India

The screenshot shows the Razorpay Docs website. The left sidebar has a dark theme with categories like Home, Get Started, Payments (selected), Banking Plus, Partners, and Developer Tools. Under Payments, 'Payment Gateway' is expanded, showing sections like Overview, Features, Get Started, How Payment Gateway Works, Web Integrations (selected), and sub-sections such as 1. Build Integration, 2. Test Integration, 3. Go-live Checklist, and Best Practices. A red box highlights the '1. Build Integration' section. The main content area shows the title '1. Build Integration' and a sub-section '1.1 Create an Order in Server'. Below it is a table with columns: Payment Stages, Order State, Payment State, and Description. The 'Description' column contains the text 'The customer submits the payment'. To the right of the main content, there's a sidebar titled 'ON THIS PAGE' with links to other sections like '1.1 Create an Order in Server', '1.2 Integrate with Checkout on Client-Side', and '1.3 Handle Payment Success and Failure'.

All the steps are listed within the context, even if they are not related to the front end. Context matters. Complete information helps developers make better decisions.

Razorpay - Payment APIs for India

The screenshot shows the Razorpay Documentation website. The left sidebar has a dark theme with categories like Home, Get Started, Payments (which is selected), Banking Plus, Partners, and Developer Tools. The main content area has a light background. A red box highlights the section titled "1.2 Integrate with Checkout on Client-Side". This section contains instructions for adding a Pay button and tables comparing Handler Function and Callback URL. Below this is a "Code to Add Pay Button" section with code samples for JavaScript.

1.2 Integrate with Checkout on Client-Side

Add the Pay button on your web page using the checkout code, Handler Function or Callback URL.

Handler Function or Callback URL

Handler Function	Callback URL
When you use this:	When you use this: <ul style="list-style-type: none">On successful payment, the customer is shown your web page.On failure, the customer is notified of the failure and asked to retry the payment.
	<ul style="list-style-type: none">On successful payment, the customer is redirected to the specified URL, for example, a payment success page.On failure, the customer is asked to retry the payment.

Code to Add Pay Button

Copy-paste the parameters as `options` in your code:

```
Manual Checkout with Callback URL (JavaScript)  
Manual Checkout with Handler Functions (JavaScript)
```

Clear differentiation and code samples for available options.

Razorpay - Payment APIs for India

The screenshot shows the Razorpay API documentation page for 'Checkout Options'. A red box highlights the parameter definitions. The left sidebar shows navigation categories like Home, Get Started, Payments, Banking Plus, Partners, and Developer Tools. The main content area has a header 'Checkout Options' and a table of parameters:

Parameter	Type	Description
key	string	API Key ID generated from the Razorpay Dashboard. mandatory
amount	integer	The amount to be paid by the customer in currency subunits. For example, if the amount is ₹100, enter 10000 . mandatory
currency	string	The currency in which the payment should be made by the customer. See the list of supported currencies . mandatory
name	string	The business name shown on the Checkout form. mandatory
description	string	Description of the purchase item shown on the Checkout form. Must start with an alphanumeric character. optional
image	string	Link to an image (usually your business logo) shown on the Checkout form. Can also be a base64 string, if loading the image from a network is not desirable. optional

On the right, there's a sidebar titled 'ON THIS PAGE' with sections for 1.1 Create an Order in Server, 1.2 Integrate with Checkout on Client-Side, and 1.3 Handle Payment Success and Failure.

Proper reference for all parameters.

Razorpay - Payment APIs for India

The screenshot shows the Razorpay Docs website with the 'Payments' section selected in the sidebar. The main content area is titled '1. Build Integration' and contains steps for integrating the Razorpay iOS Standard SDK. A red box highlights the first step: '1.1 Import Razorpay iOS Standard SDK Library.' This step has three sub-options: 'Cocoapod', 'Manual', and 'Objective-C'. Below this, there are seven more numbered steps: '1.2 Initialize Razorpay iOS Standard SDK.', '1.3 Create an Order in Server.', '1.4 Pass Payment Options and Display Checkout Form.', '1.5 Handle Success and Errors Events.', '1.6 Store the Fields in Database.', '1.7 Verify Payment Signature.', and '1.8 Verify Payment Status.'. To the right of the main content, a sidebar titled 'ON THIS PAGE' lists additional sub-sections: '1.1 Import Razorpay iOS Standard SDK Library' (with sub-options for Cocoapod, Manual, Objective-C), '1.2 Initialize Razorpay iOS Standard SDK', '1.3 Create an Order in Server' (with sub-options for API Sample Code, Error Response Parameters), '1.4 Pass Payment Options and Display Checkout Form' (with sub-options for SDK Integration Check, Initialize Payments, Enable UPI Intent on iOS), and '1.5 Handle Success and Errors Events'.

Switching to some other platform, say iOS, follows a very similar structure so the reader can expect what to find here.

Razorpay - Payment APIs for India

Razorpay Docs

Pricing API Reference Support Log In Sign Up

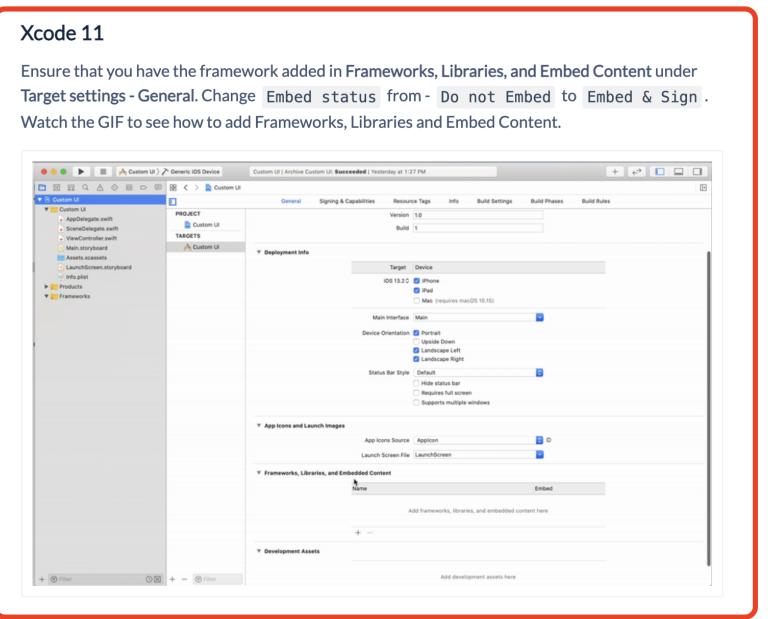
Home Get Started Payments Banking Plus Partners Developer Tools

Payment Gateway

- Overview
- Features
- Get Started
- How Payment Gateway Works
 - Web Integrations
- Hosted Checkout
 - Integrate with Hosted Checkout
 - Build Integration
 - Test Integration
 - Go-live Checklist
 - Best Practices for Hosted Checkout Integration
- Android Integration
- iOS Integration
 - Integrate with iOS Standard SDK
 - Build Integration
 - Test Integration

Xcode 11

Ensure that you have the framework added in Frameworks, Libraries, and Embedded Content under Target settings - General. Change Embed status from - Do not Embed to Embed & Sign . Watch the GIF to see how to add Frameworks, Libraries and Embedded Content.



ON THIS PAGE

- 1.1 Import Razorpay iOS Standard SDK Library
 - Cocoapod
 - Manual
 - Objective-C**
- 1.2 Initialize Razorpay iOS Standard SDK
- 1.3 Create an Order in Server
 - API Sample Code
 - Error Response
 - Parameters
- 1.4 Pass Payment Options and Display Checkout Form
 - SDK Integration Check
 - Initialize Payments
 - Enable UPI Intent on iOS
- 1.5 Handle Success and Errors Events

1.2 Initialize Razorpay iOS Standard SDK

Platform-specific instructions and videos/gifs for steps to be performed in the IDE.

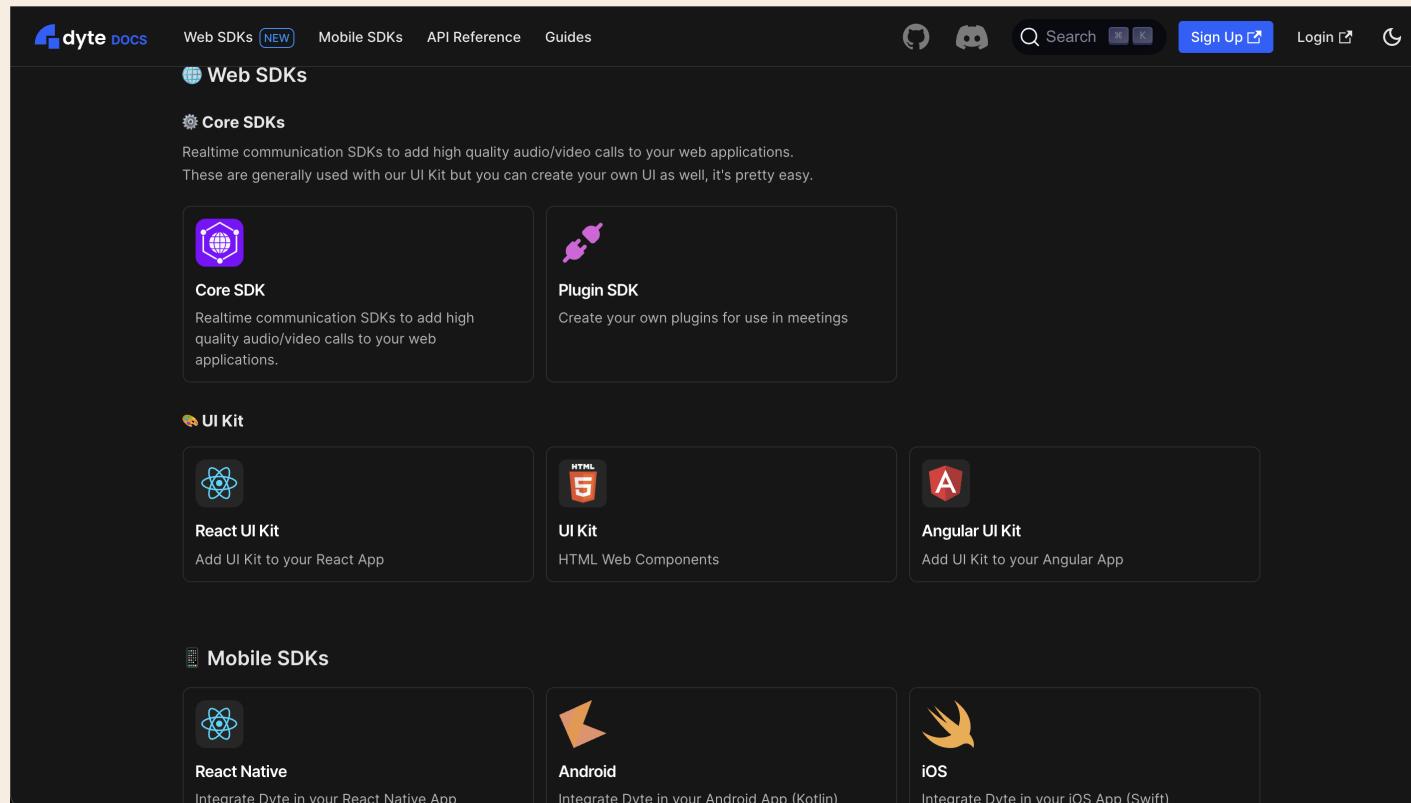
Razorpay - Payment APIs for India

The screenshot shows the Razorpay API documentation page for initializing the iOS Standard SDK. A red box highlights the first section, "1.2 Initialize Razorpay iOS Standard SDK". This section contains two "Watch Out!" callouts. The first callout discusses API keys and their storage, stating: "API keys should not be hardcoded in the app. Must be sent from your backend as app-related metadata fetch." The second callout discusses Swift version requirements and alias usage, stating: "For Swift version 5.1+, ensure that you declare `var razorpay: RazorpayCheckout!`. For versions lower than 5.1, use `var razorpay: Razorpay!`. Alternatively, you can use the following alias and retain the variable as `Razorpay`. `typealias Razorpay = RazorpayCheckout`". Below this, there is a code editor showing Swift code for initializing the SDK.

```
ViewController.swift  ViewController.m  Copy
import Razorpay
class ViewController: UIViewController, RazorpayPaymentCompletionProtocol
```

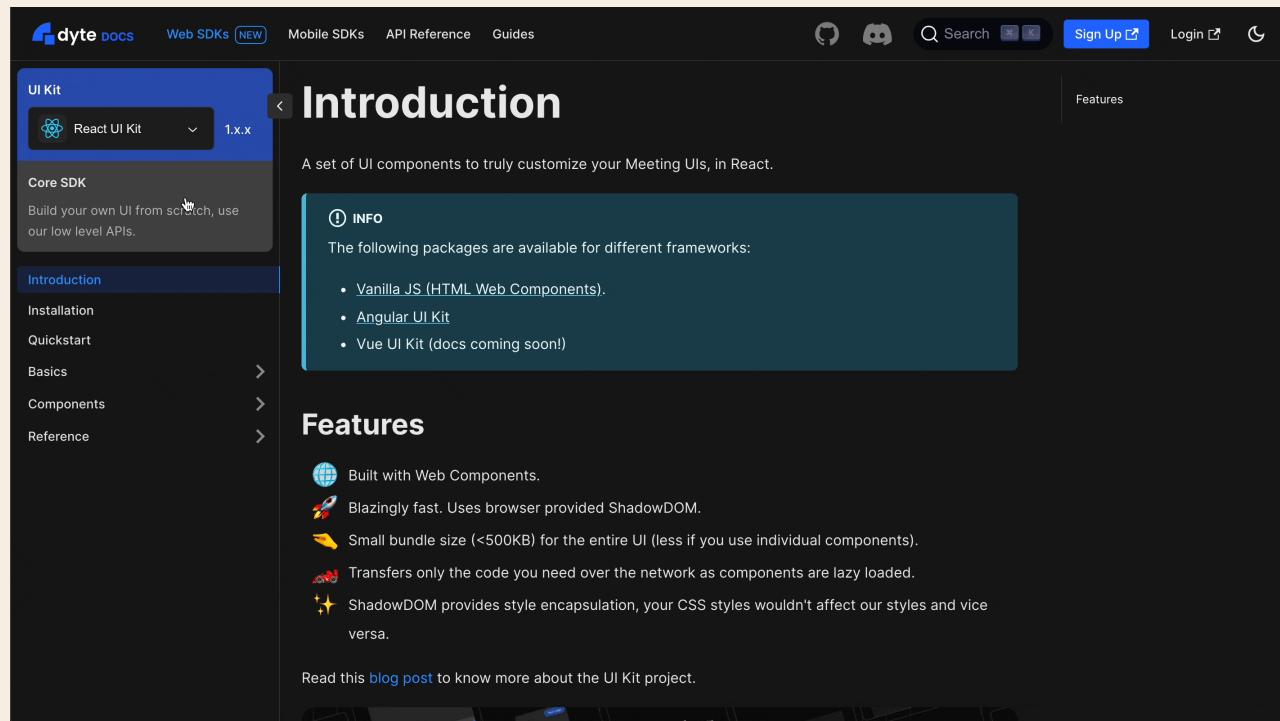
Using callouts to draw attention to specific points.

Dyte - real-time audio and video call APIs



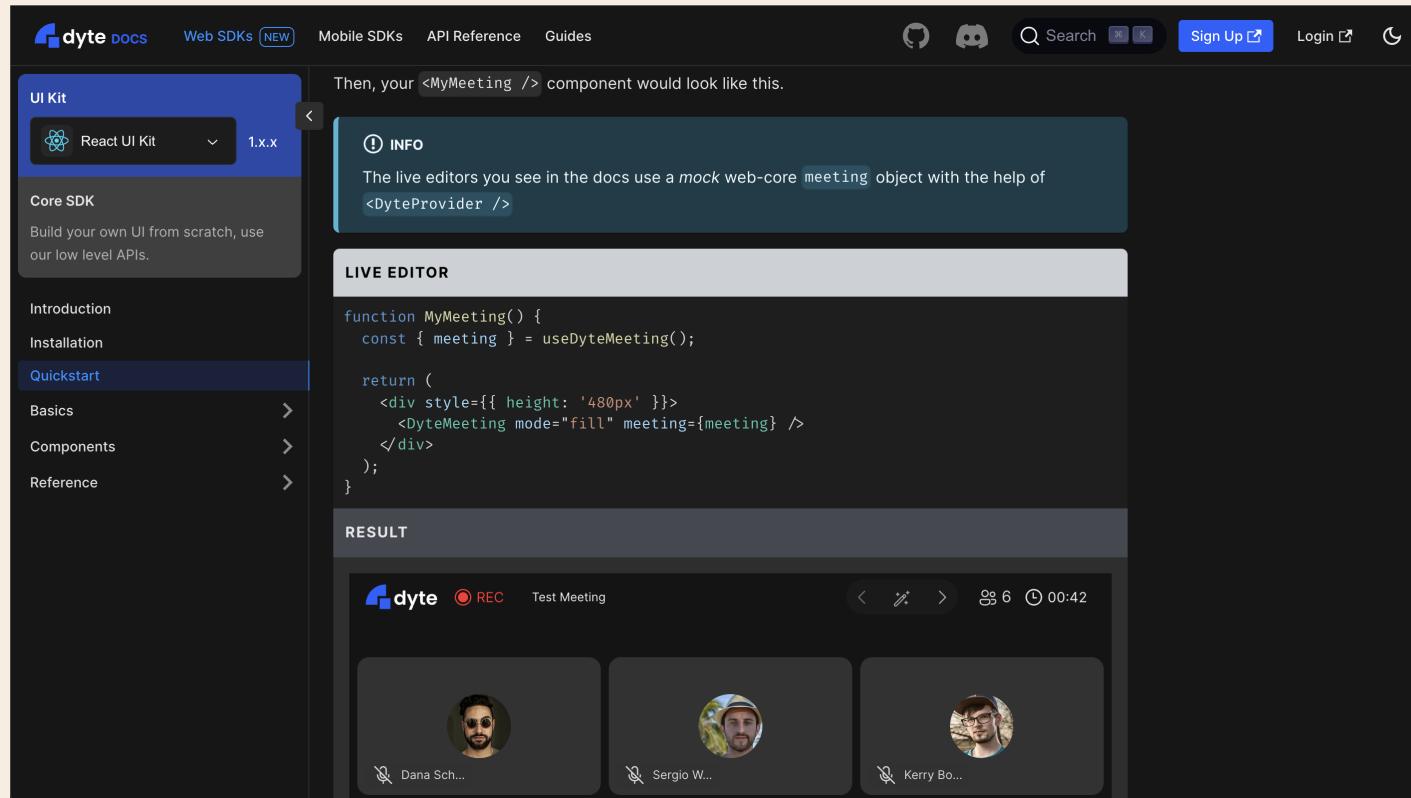
The landing page shows all the available options.

Dyte - real-time audio and video call APIs



One of the best things we think we designed is the navigation. See how both the UI SDK and Core SDK are housed under the Web SDK section but at the top level. And within each of them, you can select a different platform or framework. The best part? When you select a component or page and switch the framework, the context is preserved!

Dyte - real-time audio and video call APIs



Strong belief in "show not tell" - live editor!

Dyte - real-time audio and video call APIs

The screenshot shows the Dyte Docs website with the 'Colors' section selected. The left sidebar has a 'Design System' category expanded, showing 'UI Kit' and 'Core SDK'. The main content area displays a code snippet for CSS variables, followed by sections for 'Brand', 'Background', 'Text', and 'Text On Brand' with corresponding color swatches and hex/color values.

UI Kit
React UI Kit 1.x.x

Core SDK
Build your own UI from scratch, use our low level APIs.

Basics

- Atomic Design
- Components Basics
- Design System**
- Using Web Core Hooks
- Before You Start

Components

Reference

Colors

```
--dyte-colors-brand-500: 33 96 253;
--dyte-colors-background-1000: 8 8 8;
/* ... rest of the shades */
```

CSS Variables are set in the format: R G B.

Here are all the color tokens, along with their default values.

Brand

Value	Color
300	#497CFD
400	#356EFD
500	#2160FD
600	#0D51FD
700	#0246FD

Background

Value	Color
1000	#080808
900	#1A1A1A
800	#1E1E1E
700	#2C2C2C
600	#3C3C3C

Text

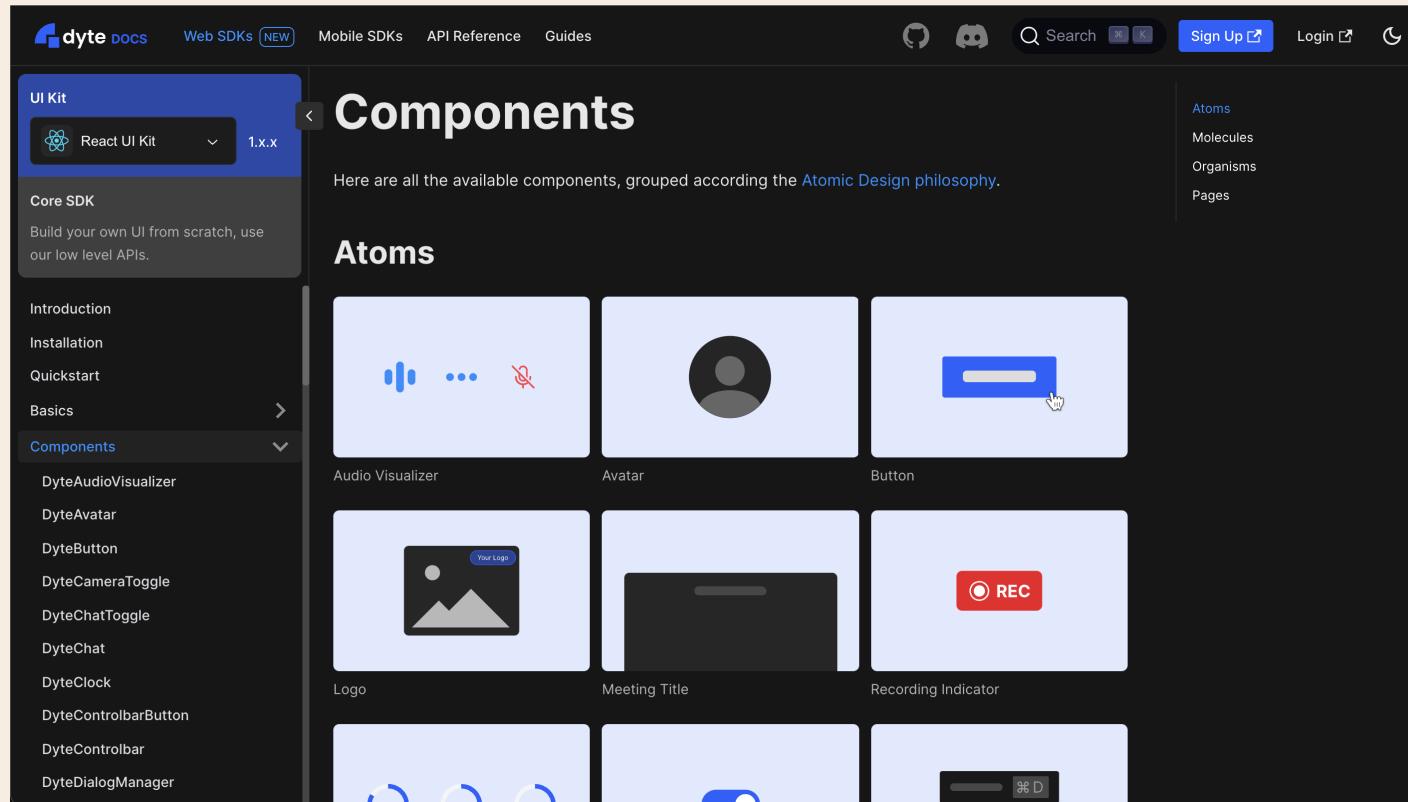
Value	Color
1000	rgb(255 255 255)
900	rgb(255 255 255 / 0.88)
800	rgb(255 255 255 / 0.76)
700	rgb(255 255 255 / 0.64)
600	rgb(255 255 255 / 0.52)

Text On Brand

Value	Color
1000	rgb(255 255 255)
900	rgb(255 255 255 / 0.88)
800	rgb(255 255 255 / 0.76)
700	rgb(255 255 255 / 0.64)
600	rgb(255 255 255 / 0.52)

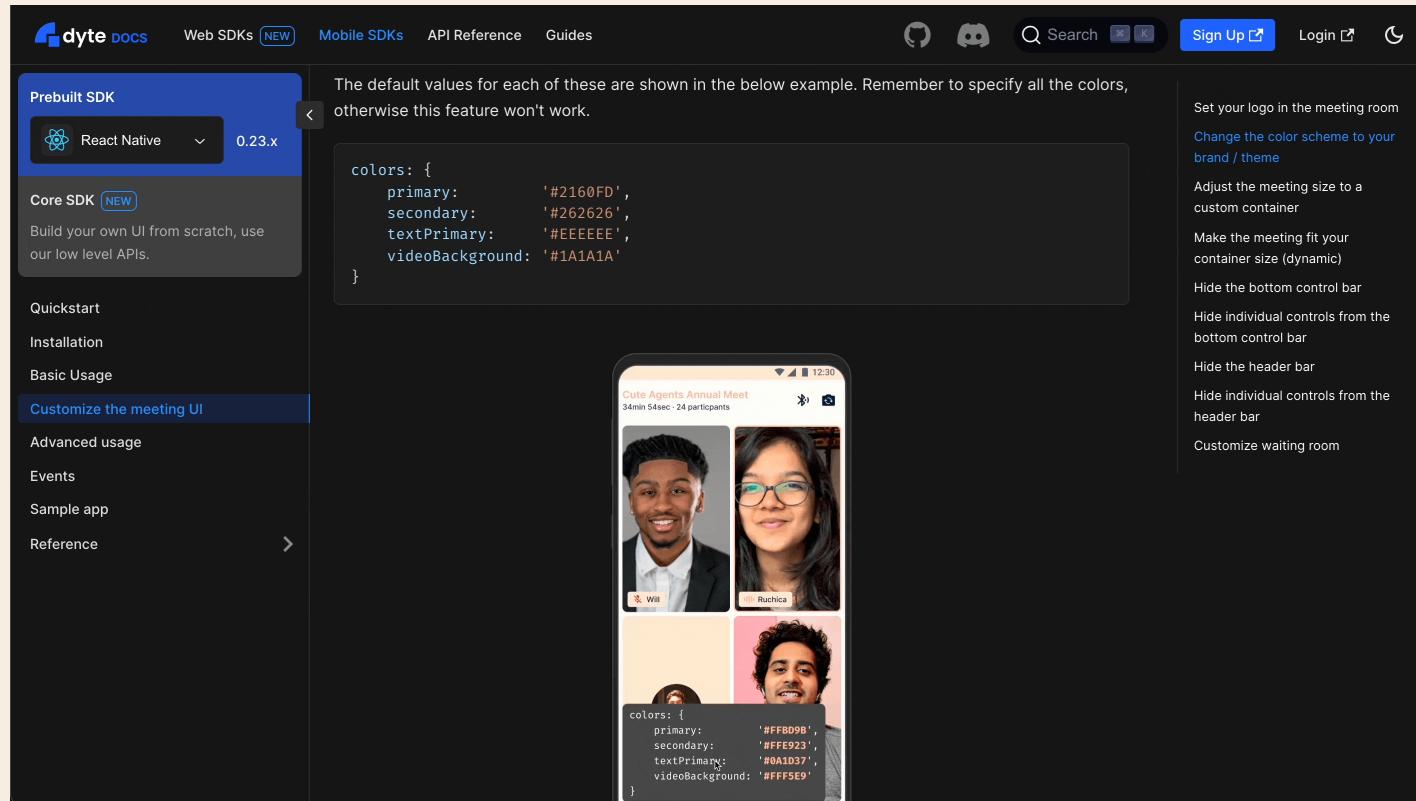
Strong belief in "show not tell" - a visual representation

Dyte - real-time audio and video call APIs



Strong belief in "show not tell" - actual renders

Dyte - real-time audio and video call APIs



Strong belief in "show not tell" - gifs which show the effect of code change

Dyte - real-time audio and video call APIs

The screenshot shows the Dyte Docs API Reference page. On the left, there's a sidebar with sections like UI Kit, Core SDK, Introduction, Installation, Quickstart, Basics, Components (with DYTEAUDIOVISUALIZER selected), DYTEAvatar, DYTEButton, DYTECameraToggle, DYTEChatToggle, DYTEChat, DYTEClock, DYTEControlBarButton, DYTEControlbar, and DYTEDialogManager. The main content area has tabs for LIVE EDITOR and RESULT. The LIVE EDITOR tab shows a code snippet: <DyteAudioVisualizer meeting={meeting} />. The RESULT tab shows a small video thumbnail. Below this, the Props section is expanded, showing properties for DYTEAUDIOVISUALIZER. The 'iconPack' prop is set to 'defaultIconPack' (Type: IconPack). The 'isScreenShare' prop is set to 'false' (Type: boolean). Other props shown are 'participant' and 'size'.

All relevant info at a single place.

Dyte - real-time audio and video call APIs

The screenshot shows the DYTE Docs website with a dark theme. The navigation bar includes links for 'dyte docs' (highlighted), 'Web SDKs (NEW)', 'Mobile SDKs', 'API Reference', and 'Guides'. The main content area is titled 'Participants' under the 'Core SDK' section. It lists several participant maps:

- `waitlisted`: A map that contains all the participants waiting to join the meeting.
- `active`: A map that contains all the participants except the local user who are supposed to be on the screen at the moment
- `pinned`: A map that contains all the pinned participants of the meeting.
- `count`: The number of participants who are joined in the meeting.
- `pageCount`: Number of pages available in paginated mode.
- `maxActiveParticipantsCount`: The maximum number of participants that can be present in the active state.

Below this, there is a snippet of JavaScript code:

```
meeting.participants.pinned.on('participantJoined', (participant) => {
  console.log(`Participant ${participant.name} got pinned`);
});
```

Further down, a section titled 'Set participant view mode' explains the difference between 'ACTIVE_GRID' and 'PAGINATED' modes. The footer notes that the view mode indicates whether participants are populated in 'ACTIVE_GRID' mode or 'PAGINATED'.

All relevant info at a single place.

Dyte - real-time audio and video call APIs

The screenshot shows the Dyte API documentation interface. The top navigation bar includes links for 'dyte docs' (highlighted), 'Web SDKs (NEW)', 'Mobile SDKs', 'API Reference', and 'Guides'. The right side features icons for GitHub, Discord, a search bar, and 'Sign Up' and 'Login' buttons.

The left sidebar contains a 'Core SDK' section with a 'JavaScript' tab (highlighted) and version '0.38.x'. Other sections include 'UI Kit', 'Installation', 'Quickstart', 'Local User', 'Room Metadata', 'Participants' (with 'The participant object' and 'Participant Events' sub-sections), 'Chat', 'Polls', 'Plugins', 'Reference' (with 'DyteClient' and 'DyteSelf' sub-sections), and 'DYTEParticipants' (highlighted).

The main content area is titled 'DYTEParticipants' and describes it as a module representing meeting participants. It lists four maps: 'joined', 'waitlisted', 'active', and 'pinned'. Below this, under 'DYTEParticipants', is a detailed list of methods:

- .waitlisted
- .joined
- .active
- .pinned
- .viewMode
- .currentPage
- .lastActiveSpeaker
- .count
- .maxActiveParticipantsCount
- .pageCount
- .acceptWaitingRoomRequest(id)
- .rejectWaitingRoomRequest(id)
- .setViewMode(viewMode)
- . setPage(page)
- .disableAllAudio(allowUnMute)
- .disableAllVideo()
- . disableParticipant(id)

On the right side of the main content area, there is a vertical list of additional methods:

- meeting.participants.waitlisted
- meeting.participants.joined
- meeting.participants.active
- meeting.participants.pinned
- meeting.participants.viewMode
- meeting.participants.currentPage
- meeting.participants.lastActiveSpeaker
- meeting.participants.count
- meeting.participants.maxActiveParticipantsCount
- meeting.participants.pageCount
- meeting.participants.acceptWaitingRoomRequest(id)
- meeting.participants.rejectWaitingRoomRequest(id)
- meeting.participants.setViewMode(page)
- meeting.participants.setPage(page)
- meeting.participants.disableAllAudio(allowUnMute)
- meeting.participants.disableAllVideo()
- meeting.participants.disableParticipant(id)

But also separately as a reference.

Dyte - real-time audio and video call APIs

The screenshot shows the Dyte UI Kit documentation page for 'Atomic Design'. The page has a dark theme with a blue header bar. The header includes the Dyte logo, 'dyte docs', 'Web SDKs (NEW)', 'Mobile SDKs', 'API Reference', 'Guides', and navigation icons for GitHub, Slack, and a search bar. On the right side of the header are 'Sign Up' and 'Login' buttons.

The main content area features a large title 'Atomic Design' with a subtitle 'Inspired by Atomic Design principles, our UI Kit is built in layers.' Below this, there's a paragraph explaining the layers: 'What that means is you can quickly get started by using just one component and keep combining even more components that will give you an entire prebuilt UI.' Another paragraph states, 'But you can also go down layers as your need for customization evolves.'

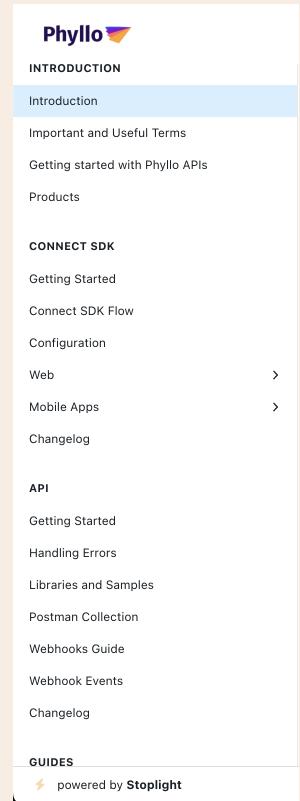
On the left, a sidebar menu is open under the 'Basics' section, showing 'Atomic Design' as the selected item. Other items in the sidebar include 'Components Basics', 'Design System', 'Using Web Core Hooks', 'Before You Start', 'Components', and 'Reference'.

In the center, there's a diagram illustrating the atomic design layers: ATOMS (a single circle), MOLECULES (three circles), ORGANISMS (a grid of nine circles), TEMPLATES (a document icon), and PAGES (a document icon with a fold). Below the diagram, a note says 'Check out the [Components](#) page to see the full list of grouped components.'

At the bottom of the page, there's a section titled 'Pages' with the subtext 'Our topmost, easiest to use layer.' and a note about the 'DyteMeeting' component.

Bonus - Design for developers!

Phyllo - Data gateway for the digital economy



Clear demarcation for SDK and API sections

Phyllo - Data gateway for the digital economy

The screenshot shows the Phyllo API documentation website. The left sidebar contains a navigation menu with sections like INTRODUCTION, CONNECT SDK, API, and GUIDES. The 'Important and Useful Terms' section is currently selected and highlighted in blue. The main content area has a search bar at the top. Below it, the title 'Important and Useful Terms' is displayed in bold. Underneath, there are two sections: 'Glossary' and 'General'. The 'General' section contains a bulleted list of definitions for terms such as Work platform/source platform, Creator, Developer, App, Data, Consent, and Access.

INTRODUCTION

- Introduction
- Important and Useful Terms
- Getting started with Phyllo APIs
- Products

CONNECT SDK

- Getting Started
- Connect SDK Flow
- Configuration
- Web >
- Mobile Apps >
- Changelog

API

- Getting Started
- Handling Errors
- Libraries and Samples
- Postman Collection
- Webhooks Guide
- Webhook Events
- Changelog

GUIDES

powered by **Stoplight**

Search Phyllo API...

Important and Useful Terms

Glossary

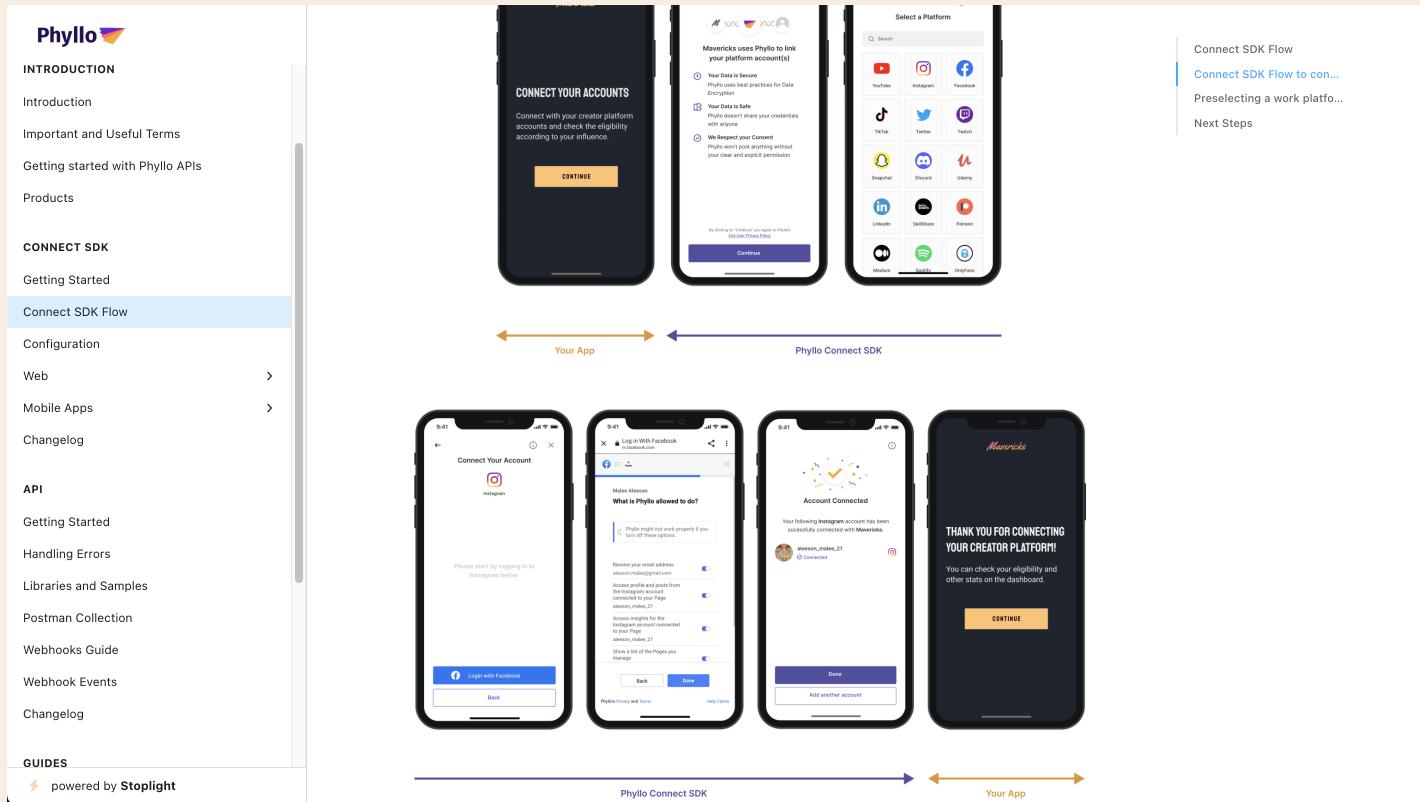
While using Phyllo, we would like to introduce you to a few terms that we frequently use. We want you to be familiar with these to work smoothly and avoid confusion.

General

- **Work platform/source platform** - A "work platform" is a platform where the users perform any activity that generates value. Examples could be Twitter, YouTube, Medium, Upwork, TikTok, Substack, etc.
- **Creator** - A "creator" is an individual who creates value in and from the digital world using one of the work platforms.
- **Developer** - A "developer" is an entity (one person, a group of persons, or an organization) that wishes to build an app or utility for creators by using their data from the relevant work platforms. We consider you a developer :)
- **App** - An "app" is a web application or a mobile application built by you.
- **Data** - "Data" is the information that Phyllo obtains from the work platform using the creator's consent and access to share with you.
- **Consent** - "Consent" refers to the explicit permission of the creator to allow Phyllo and you to collect, view, and use their data within a specified purpose.
- **Access** - "Access" refers to the scope of data permissions that the creator grants to Phyllo to

Glossary towards the beginning

Phyllo - Data gateway for the digital economy



Visual indicators for the flow

Phyllo - Data gateway for the digital economy

Phyllo

INTRODUCTION

- Introduction
- Important and Useful Terms
- Getting started with Phyllo APIs

Products

CONNECT SDK

- Getting Started
- Connect SDK Flow
- Configuration**

Web

Mobile Apps

Changelog

API

- Getting Started
- Handling Errors
- Libraries and Samples
- Postman Collection
- Webhooks Guide
- Webhook Events
- Changelog

GUIDES

⚡ powered by **Stoplight**

Connect SDK Customization

We offer specific UI customizations on our Connect SDK to make it look more like a part of your app or website rather than a separate component. Here is a list of what you can do.

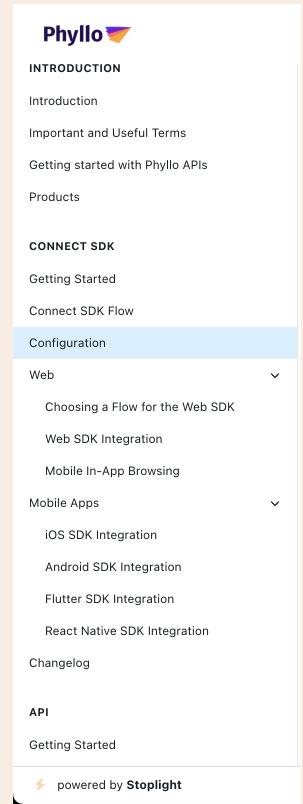
Accent color

Configuration

- Connect SDK Customizati...
- Accent color
- Logo
- Name
- Restricting to a specific W...

Visual reference for the components

Phyllo - Data gateway for the digital economy



The screenshot shows the left sidebar of the Phyllo API documentation. At the top is the Phyllo logo. Below it is a navigation menu with several sections:

- INTRODUCTION**
 - Introduction
 - Important and Useful Terms
 - Getting started with Phyllo APIs
 - Products
- CONNECT SDK**
 - Getting Started
 - Connect SDK Flow
 - Configuration** (this section is currently selected, indicated by a blue background)
- Web** (with a dropdown arrow)
 - Choosing a Flow for the Web SDK
 - Web SDK Integration
 - Mobile In-App Browsing
- Mobile Apps** (with a dropdown arrow)
 - iOS SDK Integration
 - Android SDK Integration
 - Flutter SDK Integration
 - React Native SDK Integration
- Changelog
- API**
 - Getting Started

At the bottom of the sidebar, there is a footer note: "⚡ powered by **Stoplight**".

A clear distinction between web (which is browser-based) and mobile native SDKs

Phyllo - Data gateway for the digital economy

The screenshot shows the Phyllo documentation website. The left sidebar contains navigation links for 'INTRODUCTION', 'CONNECT SDK', 'Web' (selected), 'API', and 'powered by Stoplight'. The main content area has a title 'Choose a flow' and a paragraph explaining the two available flows: 'popup' based and 'redirect' based. It highlights that the popup-based flow is a significant improvement over the redirect-flow. Below this, there are two sections: 'POPUP' and 'REDIRECT', each with a detailed description. A note in a callout box states: 'We are unable to support the popup flow of Phyllo web SDK via in-app browsing experience. If this forms a large part of your use case or customer experience, you may want to switch to redirect flow or use a smart combination of popup and redirect (as fallback).'. The right sidebar includes links to 'Understanding the Popup ...', 'Choose a flow' (which is active), 'Advantages of popup flow', and 'Which flow should you ch...'. The overall design is clean with a white background and blue accents for navigation.

Choose a flow

We currently offer two flows on the web – **popup** based and **redirect** based. The popup-based flow is a significant improvement in the user experience over the redirect-flow, which is the existing market norm.

POPUP

When you use this, the user stays on the same page (which is the developer's page – your page!) where the Phyllo SDK opens on the foreground and the platform page opens in a popup window. We will post the connection results via JavaScript event handlers.

REDIRECT

When you use this, the user gets redirected to Phyllo SDK and then to the platforms, and finally the URL specified by you. We will post the connection results on the redirect URL as query params.

(i) We are unable to support the popup flow of Phyllo web SDK via [in-app browsing experience](#). If this forms a large part of your use case or customer experience, you may want to switch to redirect flow or use a smart combination of popup and redirect (as fallback).

Advantages of popup flow

- The Phyllo Connect SDK opens in a fullscreen modal on your web page, which means the URL that the creator sees is still your URL. This helps build trust.
- The SDK opens in an HTML iframe which restricts cross-domain communication. This means only a Phyllo-controlled JS can interact with the iframe, thereby reducing security risks.
- The platform authentication and permission pages open in a secure browser popup (or a new tab) in the foreground, which means your web page which contains the SDK is still open and

There is a choice between two options

Phyllo - Data gateway for the digital economy

The screenshot shows the Phyllo documentation website. The left sidebar contains navigation links for 'INTRODUCTION', 'CONNECT SDK', 'Web' (with 'Choosing a Flow for the Web SDK' highlighted), 'Mobile Apps', 'API', and 'powered by Stoplight'. The main content area has a title 'Which flow should you choose' and two sections: 'POPUP' and 'REDIRECT'. The 'POPUP' section contains text about browser compatibility and security concerns. The 'REDIRECT' section contains text about mobile integration and security. A sidebar on the right lists related topics: 'Understanding the Popup ...', 'Choose a flow', 'Advantages of popup flow', and 'Which flow should you ch...'. At the bottom of the page, there is a note: 'An annotation has been added to the original document on the right side of the page. It highlights the "POPUP" section and includes the following text: "A comparison between the two flows is available in the original document."'

Which is clearly explained and differences highlighted

Phyllo - Data gateway for the digital economy

The screenshot shows a documentation page for the Phyllo API. The left sidebar has a navigation menu with sections like Home, Phyllo API, INTRODUCTION, CONNECT SDK, and API. Under API, 'Getting Started' is selected. The main content area is titled 'Choosing a Flow for the Web SDK'. It compares Popup flow and Redirect flow, noting that Popup flow is more reliable and efficient. A callout box highlights that Popup flow is recommended over Redirect flow due to its advantages. At the bottom, there's a note about the future of in-app browsing.

Since the web app can maintain state and process events, a creator can choose to retry a connection in case of failure and / or connect more accounts (if applicable) within a single journey. You get a native event trigger for each successful connection, disconnection, and even intermediate states such as connection failures with a reason, as long as the SDK is open.

For the creator, this leads to a higher number of page loads, clicks, etc and you need to wait for the creator to end their Connect SDK journey before you can get information about the creator's journey and what has happened in the SDK.

Moreover, since the redirect is a terminal step, events like connection failure cannot be processed. Even if they were to be processed somehow, the context and value for such intermediate events would be lost at the terminal step.

Popup flow works without any dependency on cookies or browser storage. Keep in mind though, that the platform login might not work if cookies are disabled.

Redirect flow currently cannot work without using cookies on the browser. However, in both the flows, platform login might not work if cookies are disabled.

💡 By now it should be clear that we are unable to support the popup flow of Phyllo web SDK via in-app browsing. We recommend using popup flow (or our dedicated mobile SDKs in case of native apps) as much as possible given its clear advantages over redirect. Detailed comparison and advantages can be found above. To know more about what is in-app browsing and why these problems occur, you may head to our dedicated page on [Mobile In-App Browsing](#).

We hope with this information, you will be able to make a better choice for your creators (and we all know that moving to popups is the better choice!) and make the transition to our updated SDK. If you want to discuss your specific use case further, drop a line to us.

Understanding the Popup ...
Choose a flow
Advantages of popup flow
Which flow should you ch...

powered by **Stoplight**

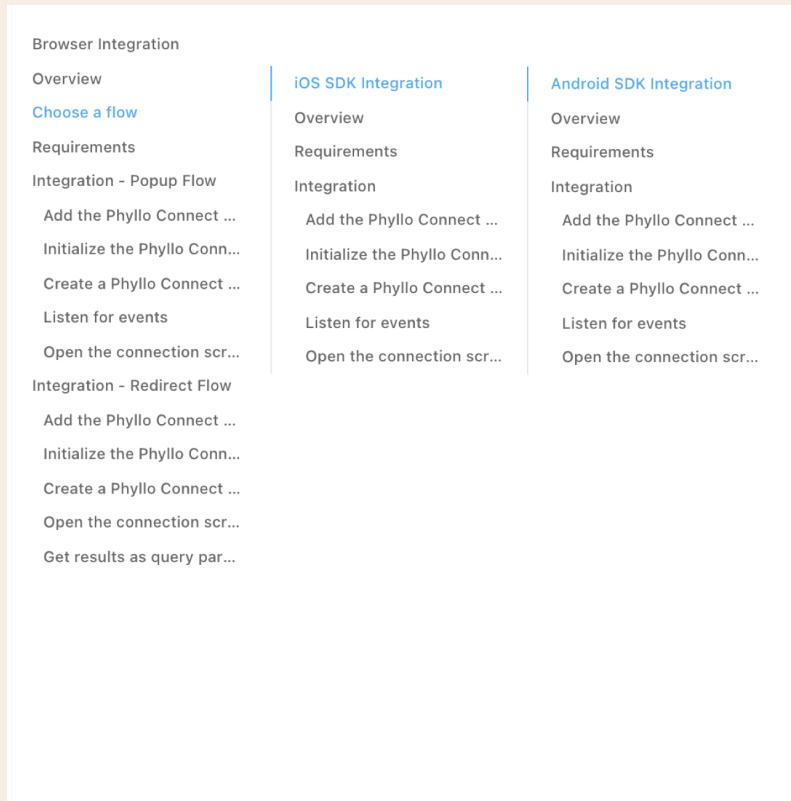
Along with a callout for the most common mistake/misunderstood path

Phyllo - Data gateway for the digital economy

The screenshot shows the Phyllo API documentation website. The left sidebar contains a navigation menu with sections like INTRODUCTION, CONNECT SDK, and API. Under the API section, 'Mobile In-App Browsing' is highlighted. The main content area has a search bar at the top. Below it, a large heading says 'Mobile In-App Browsing'. A paragraph explains what mobile in-app browsing is and why it's challenging. A 'tl;dr' summary follows, stating that Phyllo can't support the popup flow via in-app browsing and recommends using a smart combination of popup and redirect. A 'What is in-app browsing' section is also present. The URL in the browser address bar is <https://docs.getphyllo.com/docs/api-reference/connect-SDK/web/mobile-in-app...>.

And that misunderstood path - mobile in-app browsing - also gets a page of its own to explain things better

Phyllo - Data gateway for the digital economy



Try to keep the flow and explanation of steps same, or as much similar as possible

Phyllo - Data gateway for the digital economy

The screenshot shows the Phyllo API documentation interface. On the left is a navigation sidebar with sections like INTRODUCTION, CONNECT SDK, and API. The Changelog section is highlighted with a blue background. The main content area is titled "Changelog" and contains a "Web" section. The "Web" section lists three releases: 2.0.6 (09 Jul 2022), 2.0.5 (04 Jun 2022), and 2.0.4 (24 Jun 2022). Each release has a list of changes. To the right of the main content is a vertical sidebar titled "Changelog" which lists releases for various platforms: Web, iOS, Android, Flutter, and React Native.

Changelog

Web

2.0.6
Release date: 09 Jul 2022

- Internal fixes and optimizations

2.0.5
Release date: 04 Jun 2022

- Enhanced error messages in the Connect SDK for better connection experience.
- New connection help guide for Instagram at <https://help.getphyllo.com/>. We'll work on adding more such guides for other platforms too in the future, based on the connection requirements and common pitfalls. Together with the enhanced error messages, these help the creator to recover from connection errors easily.
- New native event for connection failure, with appropriate reasons, to help developers track user journeys or take actions.

Release date: 24 Jun 2022

- Remove cookie dependency from the SDK in popup flow.

Release date: 06 Jul 2022

- Fixed an issue with specific browser+plugin combination.

Changelog

Platform	Version
Web	2.0.6
Web	2.0.5
Web	2.0.4
Web	2.0.0
Web	0.1.20
iOS	0.2.2
iOS	0.1.25
iOS	0.1.24
iOS	0.1.23
iOS	0.1.21
iOS	0.1.20
iOS	0.1.16
Android	0.2.0
Android	0.1.17
Android	0.1.14
Android	0.1.13
Flutter	0.2.0
Flutter	0.1.27
Flutter	0.1.25
Flutter	0.1.24
Flutter	0.1.20
React Native	0.2.0
React Native	0.1.18
React Native	0.1.17
React Native	0.1.15

⚡ powered by **Stoplight**

Maintain a changelog (this is probably not the best way but work with whatever you have)

Let's see a few examples of how I've seen others give it a go

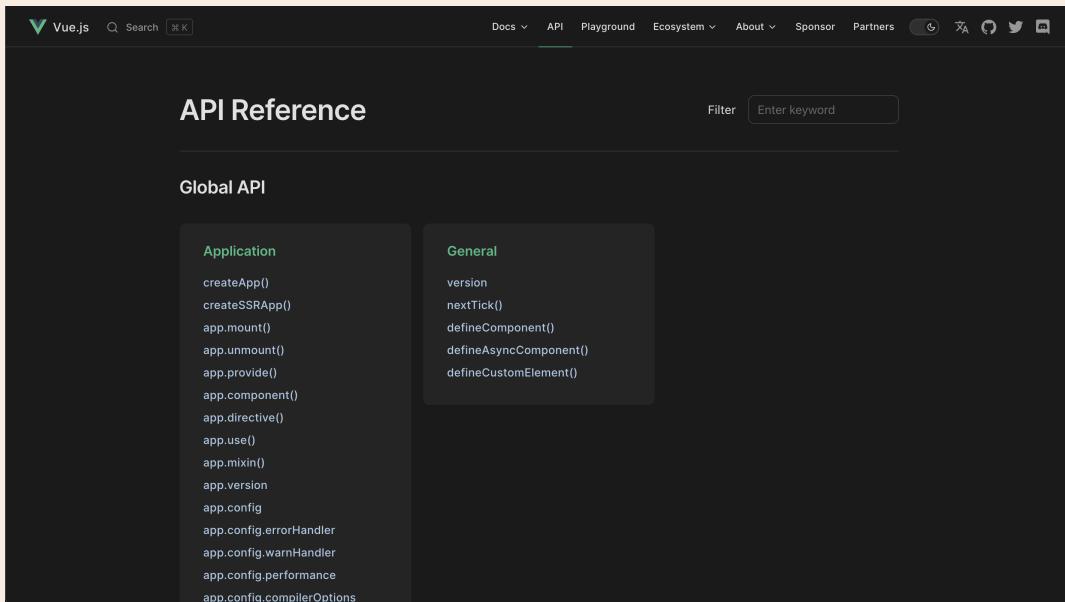
I tried turning towards two of the most popular frontend frameworks that are used to build modern apps, and here is what I found.

React JS

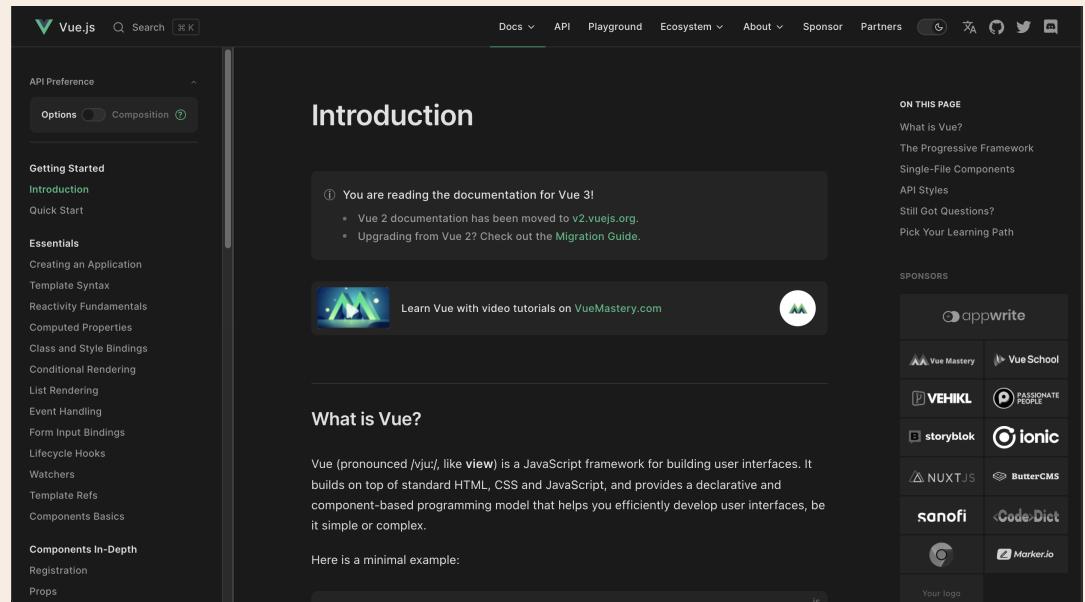
The image shows two screenshots of the React.js documentation website. The left screenshot displays the 'API' section of the documentation, featuring a sidebar with navigation links like 'Home', 'Learn', 'API', and a search bar. The right screenshot shows the 'Installation' page under the 'Learn React' section. This page includes a heading 'Installation', a paragraph about React's design for gradual adoption, and a 'Try React' section with a code editor containing 'App.js' and a preview window showing the output 'Hello, world'.

You would imagine something that is so popularly used to build interfaces and visual components would have more visual diversity in the docs.

Vue JS



The screenshot shows the Vue.js API Reference page. At the top, there's a navigation bar with links for Docs, API, Playground, Ecosystem, About, Sponsor, and Partners. Below the navigation is a search bar with a placeholder 'Enter keyword'. The main content area is titled 'API Reference' and features a 'Global API' section. Under 'Global API', there's a sidebar for 'Application' methods like createApp(), createSSRApp(), etc., and a 'General' sidebar for methods like version, nextTick(), defineComponent(), etc. A 'Filter' button is located at the top of the sidebar.



The screenshot shows the Vue.js Introduction page. At the top, there's a navigation bar with links for Docs, API, Playground, Ecosystem, About, Sponsor, and Partners. Below the navigation is a search bar with a placeholder 'Enter keyword'. The main content area is titled 'Introduction'. It includes a note about the documentation moving to v2.vuejs.org and upgrading from Vue 2. To the right, there's a sidebar titled 'ON THIS PAGE' with links to 'What is Vue?', 'The Progressive Framework', 'Single-File Components', 'API Styles', 'Still Got Questions?', and 'Pick Your Learning Path'. Below this is a 'SPONSORS' section featuring logos for appwrite, Vue Mastery, VueSchool, VEHIKL, PASSIONATE PEOPLE, storyblok, ionic, NUXTJS, ButterCMS, sanofi, Code-Dict, and Marker.io. There's also a placeholder 'Your logo'.

But they are rather boring walls of method references and guides/tutorials.

Lottie

The screenshot shows the Lottie documentation website. At the top left is a search bar with placeholder text "search...". To its right is a logo consisting of a teal stylized letter "L" inside a white rounded square. Below the search bar is a sidebar with the following navigation links:

- Home
- Sponsorship
- Supported After Effects Features
- Community Showcase
- After Effects
- Android
- Android - Jetpack Compose
- iOS/MacOS
- React Native
- Web
- Windows
- Contribute to Docs
- Other Platforms

A small teal icon with three horizontal lines is located at the bottom left of the sidebar. On the right side of the page, there is a large list of topics:

- [Loading Animation](#)
- [Animation View](#)
 - [Supplying Images](#)
 - [Playing Animations](#)
 - [Animation Settings](#)
 - [Using Markers](#)
 - [Dynamic Animation Properties](#)
 - [Adding Views to Animations](#)
 - [Enabling and Disabling Animation Nodes](#)
- [Image Provider](#)
 - [BundleImageProvider](#)
 - [FilepathImageProvider](#)
- [Animation Cache](#)
 - [LRUAnimationCache](#)
- [Value Providers](#)
 - [Primitives](#)
 - [Prebuilt Providers](#)
- [Animated Control](#)
- [Animated Switch](#)
- [Animated Button](#)
- [Examples](#)
 - [Changing Animations at Runtime](#)
- [Supported After Effects Features](#)
- [Alternatives](#)
- [Why is it Called Lottie?](#)
- [Contributing](#)
- [Issues or Feature Requests?](#)

Installing Lottie

Lottie supports [CocoaPods](#) and [Carthage](#) (Both dynamic and static). Lottie is written in *Swift 4.2*.

Something that is used to build animations has absolutely no animations on the docs and just a lot of text.

Auth0

The screenshot shows two main sections of the Auth0 documentation:

- Regular Web App**: Described as "Traditional web app that runs on the server". It lists the following frameworks:
 - Apache
 - ASP.NET (OWIN)
 - ASP.NET Core MVC
 - ASP.NET Core v2.1
 - Django
 - Express
 - Go
 - Java
 - Java EE
 - Java Spring Boot
 - Laravel
 - Next.js
 - NGINX Plus
 - PHP
 - Play On Java
 - Python
- Native/Mobile App**: Described as "Mobile or desktop app that runs natively on a device". It lists the following platforms:
 - Android
 - Android - Facebook Login
 - Cordova
 - Device Authorization Flow
 - Expo
 - Flutter
 - Ionic & Capacitor (Angular)
 - Ionic & Capacitor (React)
 - iOS / macOS
 - React Native
 - UWP
 - WPF / Winforms
 - Xamarin

Provide a lot of starter kits and guides for different frameworks.

Agora

Developers Docs API Reference SDKs Help Console Sign up

Video Calling ▾ Android iOS Web macOS Windows Electron Unity Flutter React Native

Search

Overview >

Get started >

SDK quickstart

UI Kit quickstart

Develop >

Reference >

SDK quickstart

Real-time video immerses people in the sights and sounds of human connections, keeping them engaged in your app longer.

Video Calling enables one-to-one or small-group video chat connections with smooth, jitter-free streaming video. Agora's Video SDK makes it easy to embed real-time video chat into web, mobile and native apps.

Thanks to Agora's intelligent and global Software Defined Real-time Network (SD-RTN™), you can rely on the highest available video and audio quality.

This page shows the minimum code you need to integrate high-quality, low-latency Video Calling features into your app using Video SDK.

Understand the tech

This section explains how you can integrate Video Calling features into your app. The following figure shows the workflow you need to integrate this feature into your app.

```
graph LR; App[AgoraVideo SDK] --> User1((User)); App --> User2((User)); User1 <--> Cloud[agora]; User2 <--> Cloud; subgraph Workflow [Workflow]; 1[1. Retrieve a token] --> 2[2. Join a channel]; 2 --> 3[3. Send and receive video and audio in the channel]; end
```

1. Retrieve a token
2. Join a channel
3. Send and receive video and audio in the channel

Video Calling v4.x ▾

Make it a breeze to interoperate.

And lastly, we have the love of all documentation geeks, that probably everyone looks up to.

Stripe

Stripe provides two frontend integrations - Stripe Checkout (pre-built complete checkout UI) and Stripe Elements (components to build the payments part of your checkout experience). Here is how they document both.

Stripe

The screenshot shows the Stripe Documentation homepage with the 'Payments' tab selected. Under the 'Web Elements' heading, it describes Stripe Elements as a set of prebuilt UI components for building your web checkout flow. It highlights that it's available as a feature of Stripe.js, our foundational JavaScript library for building payment flows. Stripe.js tokenizes sensitive payment details within an Element without ever having them touch your server. The page lists several features of Elements, including automatic input formatting, complete UI translations, responsive design, custom styling rules, and one-click checkout with Link. Below this, there's a 'Get started with Elements' button and three examples of payment elements: Payment Element (RECOMMENDED), Wallet Button Element, and Card Element.

The screenshot shows the Stripe Documentation homepage with the 'Payments' tab selected. Under the 'Mobile Native Elements' heading, it describes Stripe Elements as a set of prebuilt UI components for building your mobile native checkout flow. It's available as a feature of our Mobile SDKs (iOS, Android, and React Native). The page lists elements features, including automatic input formatting, complete UI translations, mobile specific behaviors, and device hardware integrations like card scanning with the camera. Below this, there's a 'Get started with Elements' button and two examples of mobile payment elements: Mobile Payment Element and Mobile Address Element (BETA).

Introduce the feature on a landing page and give the most common options to read further about.

Stripe

The screenshot shows the Stripe Documentation website. The top navigation bar includes links for Home, Payments (which is underlined), Business operations, Financial services, Developer tools, No-code, All products, APIs & SDKs, and Support. A search bar and user account options (Create account, Sign in) are also present. Below the navigation, there are filters for Platform (Web, iOS, Android), Frontend (HTML, React, Next.js), and Backend (Ruby, Node, PHP, Python, Go, .NET, Java). The main content area is titled "Custom payment flow". It features a "Prebuilt Checkout page" and a "Custom payment flow" button. A preview window displays a code editor with a Ruby script named "server.rb". The code sets up a Sinatra application to handle payment intents, requiring the Stripe gem and specifying a test API key. It defines a method to calculate the order amount and creates a PaymentIntent with a fixed amount of 1400 cents.

```
require 'sinatra'
require 'stripe'
# This is a public sample test API key.
# Don't submit any personally identifiable information in requests made with this key.
# Sign in to see your own test API key embedded in code samples.
Stripe.api_key = 'sk_test_tR3PYbcVNZZ796tH88S4VQ2u'

set :static, true
set :port, 4242

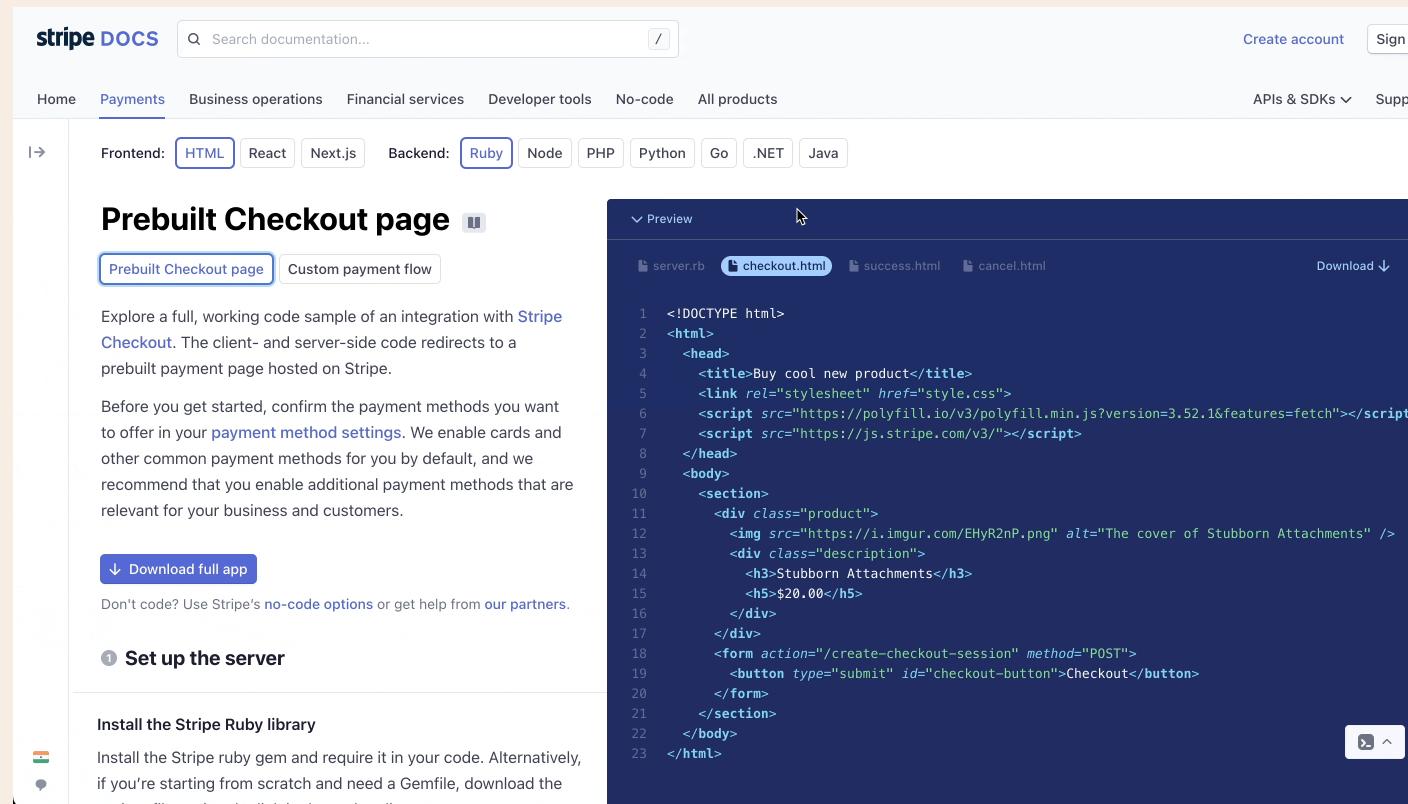
# Securely calculate the order amount
def calculate_order_amount(_items)
  # Replace this constant with a calculation of the order's amount
  # Calculate the order total on the server to prevent
  # people from directly manipulating the amount on the client
  1400
end

# An endpoint to start the payment process
post '/create-payment-intent' do
  content_type 'application/json'
  data = JSON.parse(request.body.read)

  # Create a PaymentIntent with amount and currency
  payment_intent = Stripe::PaymentIntent.create(
```

But why it works so well is that they also provide sort of a live code integration walkthrough. This is where you land when you click "get started with Elements".

Stripe

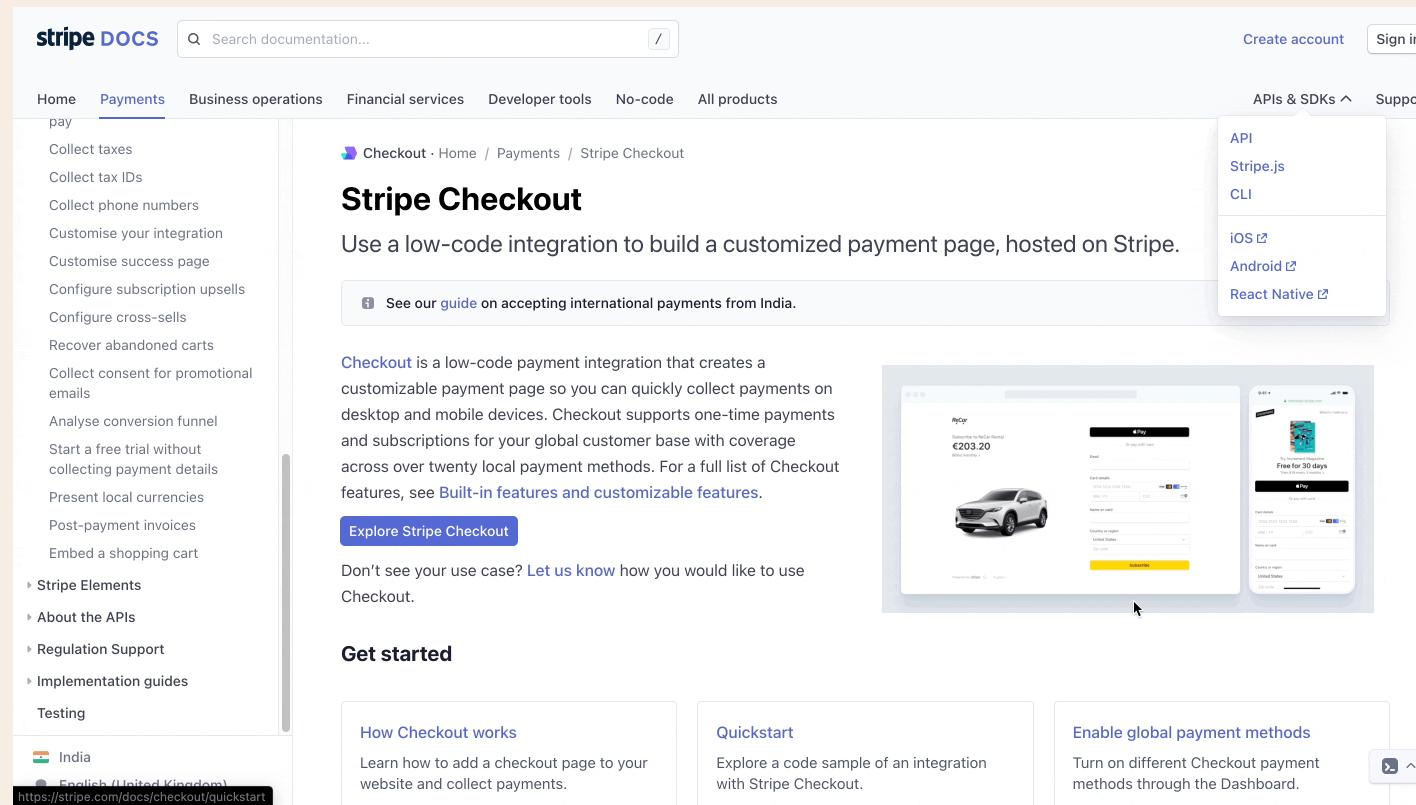


The screenshot shows the Stripe Documentation website for the Payments section. The top navigation bar includes links for Home, Payments, Business operations, Financial services, Developer tools, No-code, All products, APIs & SDKs, and Support. A search bar and sign-in options are also present. Below the navigation, there are filters for Frontend (HTML, React, Next.js) and Backend (Ruby, Node, PHP, Python, Go, .NET, Java). The main content area is titled "Prebuilt Checkout page" and features a preview of the code. The preview window shows a file structure with files like server.rb, checkout.html, success.html, and cancel.html. The checkout.html file contains the following code:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Buy cool new product</title>
5          <link rel="stylesheet" href="style.css">
6          <script src="https://polyfill.io/v3/polyfill.min.js?version=3.52.1&features=fetch"></script>
7          <script src="https://js.stripe.com/v3/"></script>
8      </head>
9      <body>
10         <section>
11             <div class="product">
12                 
13                 <div class="description">
14                     <h3>Stubborn Attachments</h3>
15                     <h5>$20.00</h5>
16                 </div>
17             </div>
18             <form action="/create-checkout-session" method="POST">
19                 <button type="submit" id="checkout-button">Checkout</button>
20             </form>
21         </section>
22     </body>
23 </html>
```

Very similar experience for the pre-built checkout section too, since most options on that screen (such as payment methods, collect address, etc.) are controlled by the API request you make before starting the payment.

Stripe



The screenshot shows the Stripe Documentation website for the 'Payments' section, specifically the 'Stripe Checkout' page. The top navigation bar includes links for Home, Payments, Business operations, Financial services, Developer tools, No-code, All products, Create account, and Sign in. A search bar is also present. On the left, a sidebar lists various payment-related features like Collect taxes, Configure subscription upsells, and Stripe Elements. The main content area features a heading 'Stripe Checkout' with a sub-copy about using low-code integration to build customized payment pages. It includes a callout for accepting international payments from India and a 'Explore Stripe Checkout' button. Below this, there's a note about use cases and a 'Get started' section with three cards: 'How Checkout works', 'Quickstart', and 'Enable global payment methods'. A sidebar on the right provides links to APIs & SDKs (API, Stripe.js, CLI, iOS, Android, React Native) and Support.

Despite this great step-by-step hand holding, they have complete references for their SDKs in case someone would like to make use of a specific component for a specific use case, or a tinkerer would like to tinker.

Conclusion

I don't know if there are rules or guidelines that I can give you. It's all very contextual and that's not great.

What we can certainly do

- Think of the user
- Think about what matters to them
- Try not to overwhelm
- But keep information accessible
- Interoperability and drawing parallels between frameworks help

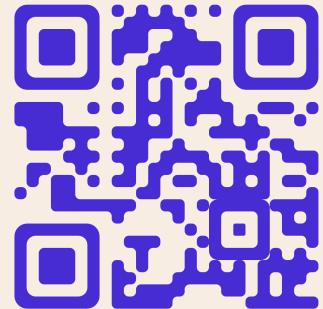
My sincere wish

Standards for method definitions, across languages and frameworks, including UI components

Thank you for listening to my rant



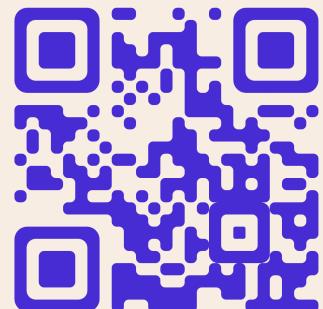
Twitter



GitHub



LinkedIn



Email



I can try answering your questions now, no guarantees though.