Hypothesis Client Documentation

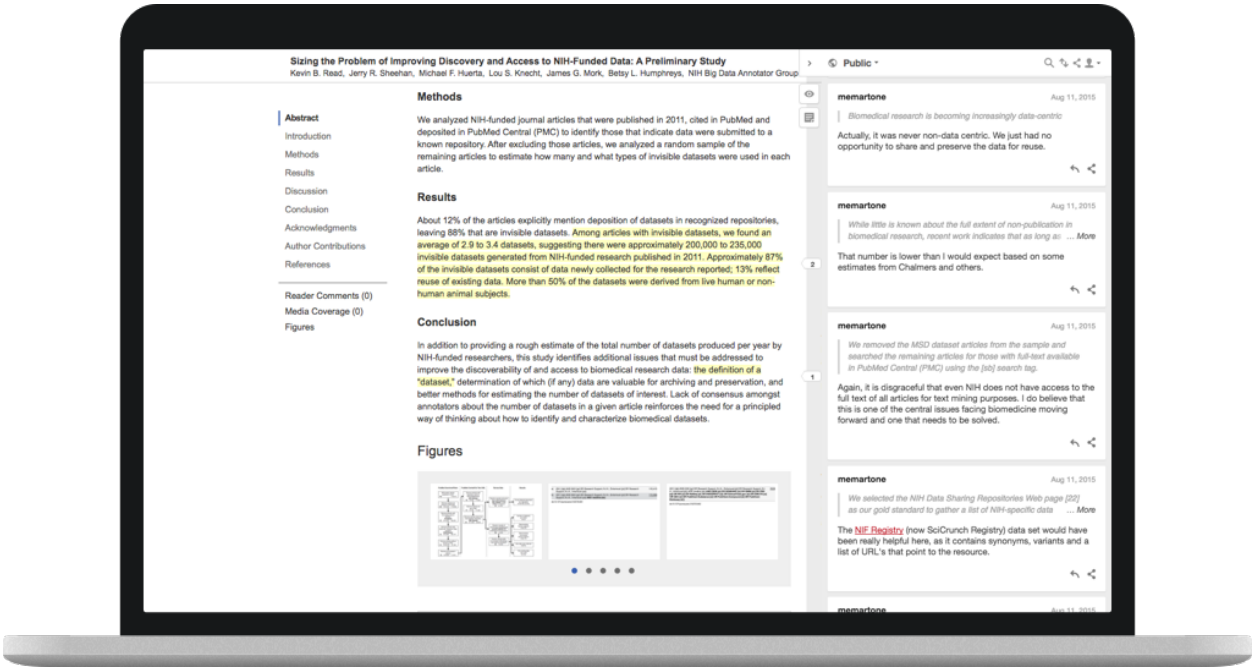
Release 1.0.0

Hypothesis Project Contributors

Mar 19, 2019

Contents

The Hypothesis client is a browser-based tool for making annotations on web pages. It's a client for h. It's used by the Hypothesis browser extension, and can also be *embedded directly into web pages*.



CHAPTER 1

For Content Publishers

This section is for **content publishers** who publish content to the web and want to integrate the Hypothesis client into their web pages. If you want to add the Hypothesis client to your web pages so that visitors can annotate the pages without having to install their own Hypothesis browser extension, these pages will help you get started.

1.1 How to Add Hypothesis to Your Website

To add Hypothesis to your website, just add this one line to the HTML source of each page that you want to have the Hypothesis client on:

```
<script src="https://hypothes.is/embed.js" async></script>
```

1.1.1 Enabling annotation of iframed content

The simplest way to support annotation of iframed content is to add the above script tag to the document displayed in the iframe. This will display the sidebar in the iframe itself.

Additionally Hypothesis has limited support for enabling annotation of iframed content while showing the sidebar in the top-level document. To use this:

1. Add the above script tag to the top-level document
2. Do not add the script tag to the iframed documents themselves, the client will do this itself.
3. Opt iframes into annotation by adding the "enable-annotation" attribute:

```
<iframe enable-annotation>
...
</iframe>
```

This method *only* works for iframes which are same-origin with the top-level document. The client will watch for new iframes being added to the document and will automatically enable annotation for them.

1.2 Configuring the Client

This page documents the configuration settings that you can use to configure the Hypothesis client once it's embedded in your website.

1.2.1 Configuring the Client Using JSON

The Hypothesis client can be configured by providing a JSON config object in the body of the hosting page:

```
<script type="application/json" class="js-hypothesis-config">
  {
    "openSidebar": true
  }
</script>
<script async src="https://hypothes.is/embed.js"></script>
```

Not all configuration settings can be set in this way, some must be *set using JavaScript* (see below).

Note: The body of the `.js-hypothesis-config` tag must be valid JSON, invalid JSON will cause the entire config object to be ignored.

1.2.2 Configuring the Client Using JavaScript

`window.`**`hypothesisConfig`**`()`

Alternatively, the Hypothesis client can be configured from the hosting page by providing a JavaScript function named *`window.hypothesisConfig()`* that returns a configuration object. Some configuration settings (for example settings that register callback or event handler functions) can *only* be set from JavaScript:

```
window.hypothesisConfig = function () {
  return {
    "openSidebar": true
  };
};
```

1.2.3 Config Settings

Client Behavior

These settings configure the behavior and initial state of the client when it loads.

`openSidebar`

`Boolean`. Controls whether the sidebar opens automatically on startup. (Default: `false`.)

`showHighlights`

`String|Boolean`. Controls whether the in-document highlights are shown by default. (Default: `"always"`).

`true` or `"always"` - Highlights are always shown by default.

`false` or `"never"` - Highlights are never shown by default, the user must explicitly enable them.

`"whenSidebarOpen"` - Highlights are only shown when the sidebar is open.

> **Warning:** The "always", "never" and "whenSidebarOpen" values are currently still experimental and may change in future. `true` and `false` values are the stable API.

theme

`String`. Controls the overall look of the sidebar.(Default: `classic`).

`"classic"` - Enables the card view for annotations, the bucket bar, the sidebar minimize button, the highlights button and the new note button in the toolbar. It also disables the close button in the toolbar. The classic theme is enabled by default.

`"clean"` - Enables the clean view for annotations in the sidebar, disables the bucket bar, the sidebar minimize button, the highlights button and the new note button in the toolbar and enables the close button in the toolbar. It will also show a cleaner and more minimal onboarding tutorial.

enableExperimentalNewNoteButton

`Boolean` - Controls whether the experimental New Note button should be shown in the notes tab in the sidebar. (Default: `false`).

`true` - The button is shown.

`false` - The button is not shown.

usernameUrl

`String`. This allows you to specify a URL to direct a user to, in a new tab when they click on the annotation author link in the header of an annotation. The username is appended to the end of *usernameUrl*.

For example:

```
window.hypothesisConfig = function () {
  return {
    usernameUrl: 'https://partner.org/user/',
  };
};
```

services

`Array`. A list of alternative annotation services which the client should connect to instead of connecting to the public Hypothesis service at hypothes.is. May optionally include information (in the form of grant tokens) about user accounts that the client is logged in to on those services.

For example:

```
window.hypothesisConfig = function () {
  return {
    services: [{
      apiUrl: 'https://hypothes.is/api/',
      authority: 'partner.org',
      grantToken: '***',
      icon: 'https://openclipart.org/download/272629/sihouette-animaux-10.
↪svg'
    }],
  };
};
```

By default, if no *services* setting is given, the client connects to the public Hypothesis service at hypothes.is.

Warning: The *services* setting is currently still experimental and may change in the future.

Note: Currently only one additional annotation service is supported - only the first item in this *services* array is used, and any further items in the array are ignored.

Each item in the *services* array should be an object describing an annotation service.

Required keys:

apiUrl
`String`. The base URL of the service API.

authority
`String`. The domain name which the annotation service is associated with.

grantToken
`String|null`. An OAuth 2 grant token which the client can send to the service in order to get an access token for making authenticated requests to the service. If `null`, the user will not be logged in and will only be able to read rather than create or modify annotations. (Default: `null`)

See also:

Generating authorization grant tokens for how to generate grant tokens for the hypothes.is service.

Optional keys:

enableShareLinks
`boolean`. A flag indicating whether annotation cards should show links that take the user to see an annotation in context. (Default: `true`).

groups
`String[]|null`. An array of group IDs. If provided, the list of groups fetched from the API will be filtered against this list so that the user can only select from these groups.

This can be useful in contexts where it is important that annotations are made in a particular group.

icon
`String|null`. The URL to an image for the annotation service. This image will appear to the left of the name of the currently selected group. The image should be suitable for display at 16x16px and the recommended format is SVG.

onLoginRequest
`function`. A JavaScript function that the Hypothesis client will call in order to log in (for example, when the user clicks a log in button in the Hypothesis client's sidebar).

This setting can only be set using *window.hypothesisConfig()*.

If the hosting page provides an *onLoginRequest* function then the Hypothesis client will call this function instead of doing its usual procedure for logging in to the public service at hypothes.is.

No arguments are passed to the *onLoginRequest* function.

The *onLoginRequest* function should cause a log in procedure for the hosting page to be performed - for example by redirecting to a log in page, or by opening a popup log in win-

dow. After a successful log in the hosting page should reload the original page with a non-null *grantToken* for the logged-in user in the *services* configuration setting.

onLogoutRequest
`function`. A JavaScript function that the Hypothesis client will call in order to log out (for example, when the user clicks a log out button in the Hypothesis client's sidebar).

This setting can only be set using *window.hypothesisConfig()*.

If the hosting page provides an *onLogoutRequest* function then the Hypothesis client will call this function instead of doing its usual procedure for logging out of the public service at hypothes.is.

No arguments are passed to the *onLogoutRequest* function.

The *onLogoutRequest* function should cause a log out procedure for the hosting page to be performed. After a successful log out the hosting page should reload the original page with no *grantToken* in the *services* configuration setting.

onSignupRequest
`function`. A JavaScript function that will be called when the user clicks the "Sign up" link in the sidebar. No arguments are passed and the return value is unused.

This setting can only be set using *window.hypothesisConfig()*.

onProfileRequest
`function`. A JavaScript function that will be called when the user clicks the user profile (user name) link in the sidebar. No arguments are passed and the return value is unused.

This setting can only be set using *window.hypothesisConfig()*.

onHelpRequest
`function`. A JavaScript function that will be called when the user clicks the "Help" link in the sidebar. No arguments are passed and the return value is unused.

This setting can only be set using *window.hypothesisConfig()*.

branding
Branding lets you adjust certain aspects of the sidebar's look and feel to better fit your site's own look.

`Object`. The key-value pairings used to identify how the brandable elements in the sidebar should be presented. The allowed keys will be described below. The values will be directly mapped to the css styles for the elements which it affects. That means any valid css property for the specified type will work. For example, if the value type is a Color, you can specify any browser supported color value (hex, rgb, rgba, etc.).

For example:

```
window.hypothesisConfig = function () {
  return {
    branding: {
      appBackgroundColor: 'white',
      ctaBackgroundColor: 'rgba(3, 11, 16, 1)',
      ctaTextColor: '#eee',
      selectionFontFamily: 'helvetica, arial, sans serif'
    }
  };
};
```

The following keys are supported in the *branding* object. You will also see what value type we are expecting.

> **Warning:** The *branding* setting is currently still experimental and may change in the future.

accentColor
: Color. We have several areas in our client that have pops of color that are secondary to the primary call to action elements. Things such as the "more" and "less" links to expand and collapse large annotation bodies.

appBackgroundColor
: Color. This will update the main background color of our app.

ctaBackgroundColor
: Color. This will update the main call-to-action button backgrounds. A call-to-action button example would be our "Post to {Group Name}" button when making an annotation.

ctaTextColor
: Color. This will update the text color inside of the call-to-action buttons.

selectionFontFamily
: Font Family. The selection text is the part of the annotation card that reflects what the user highlighted when they made the annotation. This value will update the font-family of that text.

annotationFontFamily
: Font Family. The annotation text is the actual annotation value that the user writes about the page or selection. This value will set the font-family of that text when it is being viewed as well as the font-family of the editor as the annotation is being written.

onLayoutChange
: function. This function will be a registered callback to be invoked when the sidebar layout changes. Changes to the layout occur on load, when the sidebar is toggled to show and hide, and when the user adjusts the sidebar manually.

  This setting can only be set using *window.hypothesisConfig()*.

  When a layout change happens the registered *onLayoutChange* function will receive a single Object as it's argument. This object details the layout parameters after the change.

  Layout object available fields:

  expanded
  : Boolean. If the sidebar is open, this value will be true.

  height
  : Number. The current visible height of the sidebar.

  width
  : Number. The current visible width of the sidebar.

externalContainerSelector

> **Warning:** This is an experimental API and may change in future.

string. A CSS selector specifying the containing element into which the sidebar iframe will be placed.

This option provides the publisher with more control over where the sidebar is displayed on the screen and how and when it appears and disappears.

When this option is not specified, Hypothesis chooses where to place the sidebar, typically on the right side of the page, and provides the user with controls to open and close it.

When this option is specified, the sidebar will be created and placed inside the specified element. Hypothesis will not display its own controls for opening and closing the sidebar and will not display the "bucket bar" showing where annotations are located on the page relative to the current scroll position.

Asset and Sidebar App Location

These settings configure where the client's assets are loaded from.

> **Warning:** These settings are currently still experimental and may change in the future.

assetRoot
`String`. The root URL from which assets are loaded. This should be set to the URL where the contents of the hypothesis package are hosted, including the trailing slash. (Default: for production builds: `"https://cdn.hypothes.is/hypothesis/X.Y.Z/"`, for development builds: `"http://localhost:3001/hypothesis/X.Y.Z/""`. ``X.Y.Z`` is the package version from `package.json`).

sidebarAppUrl
`String`. The URL for the sidebar application which displays annotations (Default: `"https://hypothes.is/app.html"`).

1.3 Interacting with the Client

This page documents the ways in which your website can interact with the Hypothesis client, once the client is embedded in your site.

data-hypothesis-trigger
You can add a button to your page that opens the Hypothesis sidebar.

If you need to have a custom trigger on your third party page to bring up the embedded Hypothesis sidebar, add the *data-hypothesis-trigger* attribute to the element that you want to enable. Clicking that element will cause the sidebar to open. Note, however, subsequent clicks do not hide the sidebar.

For example to add a `<button>` on a page to open the sidebar, simply add the *data-hypothesis-trigger* attribute:

```
<button data-hypothesis-trigger>
  Open sidebar
</button>
```

data-hypothesis-annotation-count
You can add a count of the number of annotations to your page.

If you need to show the total number of public annotations, page notes and orphaned annotations on your third party page where the Hypothesis client is embedded, add the *data-hypothesis-annotation-count* attribute to the element that you want to enable. The contents of the enabled element will be replaced with the count of public annotations and if there are no public annotations, with 0.

For example to display the annotation count in a `<div>` element, simply add the *data-hypothesis-annotation-count* attribute to the `<div>`:

```
<div data-hypothesis-annotation-count>
  Annotation count will appear here
</div>
```

CHAPTER 2

For Developers

This section contains documentation for **developers** contributing code to the Hypothesis client.

2.1 Developing the Client

This section documents how to setup a development environment for the client, how to run the client and its tests in a development environment, client coding standards and how to contribute code to the client.

2.1.1 Setting up a Client Development Environment

Prerequisites

You will need:

- git
- Node.js v6.3+
- Yarn

Building

To build the client for development:

```
git clone 'https://github.com/hypothesis/client.git'
cd client
npm install -g gulp-cli  # Tip: if you get a "permission denied" error try
                         # `sudo npm install -g gulp-cli` instead.
make
```

You now have a development client built. To run your development client in a browser you'll need a local copy of either the Hypothesis Chrome extension or h. Follow either *Running the Client from the Browser Extension* or *Running the Client From h* below. If you're only interested in making changes to the client (and not to h) then running the client from the browser extension is easiest.

2.1.2 Running the Client from the Browser Extension

This is the currently easiest way to get your development client running in a browser. It sets you up to make changes to the client and to the Chrome extension itself, but not to h.

1. Check out the browser extension and follow the steps in the browser extension's documentation to build the extension and configure it to use your local version of the client and the production Hypothesis service.
2. Start the client's development server to rebuild the client whenever it changes:

   ```
   make dev
   ```

3. After making changes to the client, you will need to run `make` in the browser extension repo and reload the extension in Chrome to see changes. You can use Extensions Reloader to make this easier.

2.1.3 Running the Client From h

This takes longer to setup than *Running the Client from the Browser Extension*. You should follow these steps if you want to make changes to h as well as to the client.

First follow the instructions for setting up a development install of h. Once you have a development install of h set up, you can configure it to use a local build of the client. **In the client repository**, run:

```
export SIDEBAR_APP_URL=http://localhost:5000/app.html
make dev
```

Next, you'll need to create an OAuth client which enables the Hypothesis client to request an access token from the service in order to make API calls.

1. Go to http://localhost:5000/admin/oauthclients (you'll need to be logged in to h as an admin user)
2. Select "Register a new OAuth client"
3. Choose a name (eg. "Client") and set the redirect URL to http://localhost:5000/app.html. Leave the other settings at their default values.
4. After creating the client, make a note of the randomly generated client ID.

In the 'hypothesis/h' repository, set the `CLIENT_URL` and `CLIENT_OAUTH_ID` env vars to tell h where to load the client from and what OAuth client to use, before running `make dev`:

```
export CLIENT_OAUTH_ID={ OAuth client ID from step above }
export CLIENT_URL=http://localhost:3001/hypothesis
make dev
```

Once the client and h are running, you can test it out by visiting: http://localhost:3000 or http://localhost:5000/docs/help in your browser.

You can also load the client into your own web pages by adding:

```
<script async src="http://localhost:5000/embed.js"></script>
```

to the page's HTML. Note that this will only work in pages served via plain HTTP. If you want to test out the client on pages served via HTTPS then building the client into a browser extension is the easiest option.

2.1.4 Running the Tests

Hypothesis uses Karma and mocha for testing. To run all the tests once, run:

```
make test
```

You can filter the tests which are run by running `make test FILTER=<pattern>`. See the documentation for Mocha's grep option.

To run tests and automatically re-run them whenever any source files change, run:

```
make servetests
```

This command will also serve the tests on localhost (typically *http://localhost:9876*) so that break points can be set and the browser's console can be used for interactive debugging.

2.1.5 Code Style

JavaScript

Hypothesis uses ESLint (a linter) and Prettier (an automated code formatter) to ensure style consistency and help prevent common mistakes. Plugins are available for most editors for these tools. We recommend that you set these up before making changes to the code.

To auto-format code and run lint checks locally using the CLI, run:

```
make format
make lint
```

CSS

Styling is authored in SASS. For guidance on writing CSS for Hypothesis projects, please see our CSS Guide.

2.1.6 Submitting Pull Requests

For general guidance on submitting pull requests to Hypothesis projects, please see the Contributor's Guide.

2.2 Environment Variables

This section documents all the environment variables supported by the client's build tasks.

SIDEBAR_APP_URL
: The default value for the *sidebarAppUrl* config setting (the URL of the sidebar app's iframe), used when the host page does not contain a *sidebarAppUrl* setting. `https://hypothes.is/app.html` by default.

PACKAGE_SERVER_HOSTNAME
: The hostname that is used to generate URLs to the package content server.

2.3 Mobile Development

2.3.1 Testing the Client on a Mobile Device

If you have made changes to the client that could affect the mobile experience, it is a good idea to test them on a real device. Such changes should ideally be tested with at least current versions of iOS Safari and Chrome for Android.

1. Make sure your development system and mobile device are on the same local network.
2. Configure h to allow incoming connections from other systems by editing `conf/development-app.ini` and changing the `host` setting from `localhost` to `0.0.0.0`.
3. Get the IP address or hostname of your development system (`<HOSTNAME>` in the steps below). You can do this using the `hostname` terminal command on Mac/Linux.

   Tip: If the output of `hostname` does not include a `.home` or `.local` suffix, you may need to append `.local` to get a hostname that is accessible from other devices on the network. If you have problems using the hostname, try using the IP address instead.

4. Set the `CLIENT_URL` environment variable to configure h to load the client from this host and start the dev server:

```
# In the h repository

# Configure the URL that the client is loaded from in pages
# that embed Hypothesis
export CLIENT_URL=http://<HOSTNAME>:3001/hypothesis

make dev
```

5. Set the *SIDEBAR_APP_URL* and *PACKAGE_SERVER_HOSTNAME* environment variables to load assets from this hostname and start the dev server:

```
# In the client repository

# Set URL which sidebar app ("app.html") is loaded from
export SIDEBAR_APP_URL=http://<HOSTNAME>:5000/app.html
# Set hostname used when generating client asset URLs
export PACKAGE_SERVER_HOSTNAME=<HOSTNAME>

make dev
```

   Tip: When `make dev` runs, it will print out the URLs used for h and client assets. These should include `<HOSTNAME>` instead of `localhost`.

6. On your mobile device, go to a page which has the client embedded such as `http://<HOSTNAME>:3000` or `http://<HOSTNAME>:5000/docs/help`.

2.4 Client Security

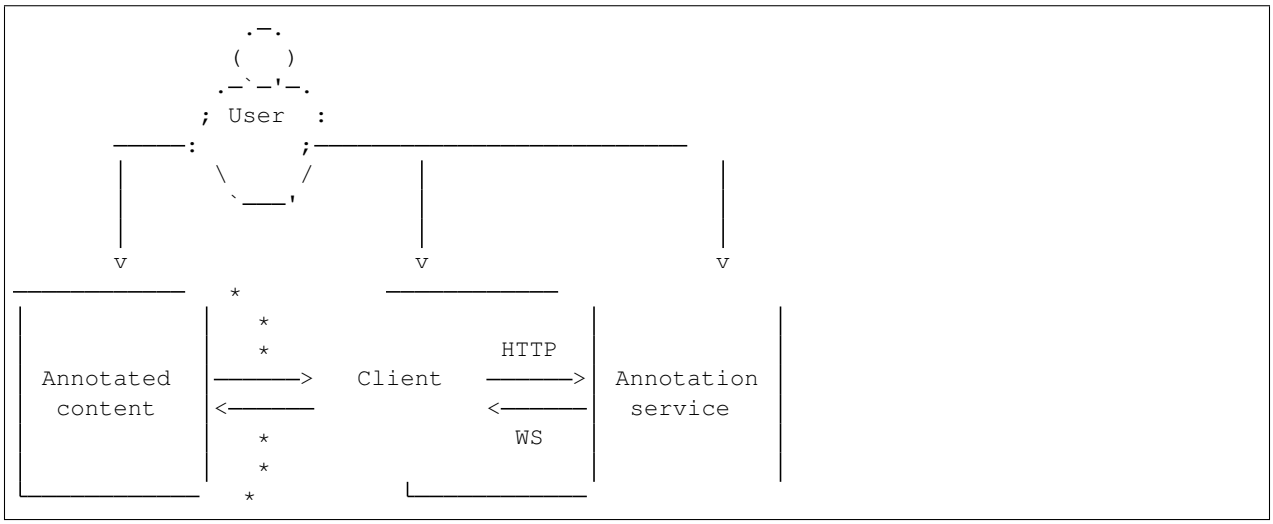
This document is intended to give an overview of the security considerations which must be kept in mind when working on the Hypothesis client. It outlines the overall security goals for the client, names some risks and attack vectors, and

identifies ways in which code in the client attempts to mitigate those risks.

2.4.1 Environment Overview

The Hypothesis client is a single-page web application which runs in a browser. Typically, it interacts with some annotated content (the page on which annotations are made) and an annotation service running on a remote server.

At different times, users interact directly with the client, with the annotated content, and with the annotation service. Data can flow in both directions: from the annotated content to the client and vice versa. Communication with the annotation service is also bidirectional, making use of an HTTP API and a WebSocket connection:



There are two important trust boundaries in this system:

1. Between the client code, executing in a browser, and the service, executing on a remote server.
2. Between the annotated content (which may be an HTML page or a PDF rendered as an HTML page) and the client application. This boundary is marked with asterisks (*) in the ASCII art above.

2.4.2 Threat Model

We are principally interested in ensuring that untrusted parties cannot gain access to data that is intended to be confidential, or tamper with such data when it is in transit. Protected data might include:

- user credentials
- annotation data or metadata which is displayed by the client
- user profile information
- group membership records
- user search history

We must assume that the user has a baseline level of trust in:

1. their browser software (and the platform it runs on)
2. our client software
3. the annotation service
4. any 3rd-party account provider mediating access to the annotation service (e.g. Google, Facebook, etc.)

Any other parties are considered untrusted. Untrusted actors thus include any and all of the following:

- the publishers of arbitrary web pages (including annotated content)
- advertisers or other 3rd-party contributors to arbitrary web pages (including annotated content)
- other users of the annotation service who have not been explicitly designated as trusted (through group membership, for example)
- members of the public who don't use the annotation service
- active attackers

We aim to defend confidential user data against any possibility of unauthorised access.

2.4.3 Potential Attack Vectors

The mechanisms of directed attack we are aiming to defend against are common to many web applications, namely:

- execution of untrusted code in a trusted context (principally by XSS)
- clickjacking
- phishing/imitation attacks
- eavesdropping of unencrypted network traffic by an untrusted party
- to a limited extent, cross-site request forgery, although this is mostly a concern for the annotation service

2.4.4 Design Considerations and Defenses

Same-Origin Policy Protections

The starting point for understanding many of the client-side security mechanisms is the web platform's same-origin policy (SOP), which ensures that any document on origin[1] "A" has very limited access to the execution context or DOM tree of any document on a different origin "B".

As shown in Fig. 2.1, the bulk of the Hypothesis client application executes within an `<iframe>` injected into the annotated content. This `<iframe>` has an origin distinct from that of the hosting page, which means that most of the protections of the SOP apply. Most importantly, code executing in the context of the annotated page cannot inspect the DOM of the client frame. The red border in the image is a visual representation of the trust boundary between the inherently untrusted execution context of the annotated page, and the trusted execution context of the client frame.

Instead, the components of the client which execute in the annotated page must communicate with the client frame using cross-document messaging. It is important that such **cross-document messaging should expose only the minimum information necessary about user data** to code executing in the annotated page. For example: in order to draw highlights, the annotated page needs to know the location of annotations, but it does not ever need to know the body text of an an annotation, and so it should not be possible to expose this over the messaging interface.

Todo: 2017-03-08

Currently the client shares an origin with the annotation service when delivered by any mechanism other than the Chrome extension. This makes any XSS vulnerability in the client a problem for the service and vice versa. We need to move the client to its own origin to better isolate the client from the service and minimise the risk posed by XSS.

---

[1] An origin is the tuple of (scheme, host, port) for a given web document.

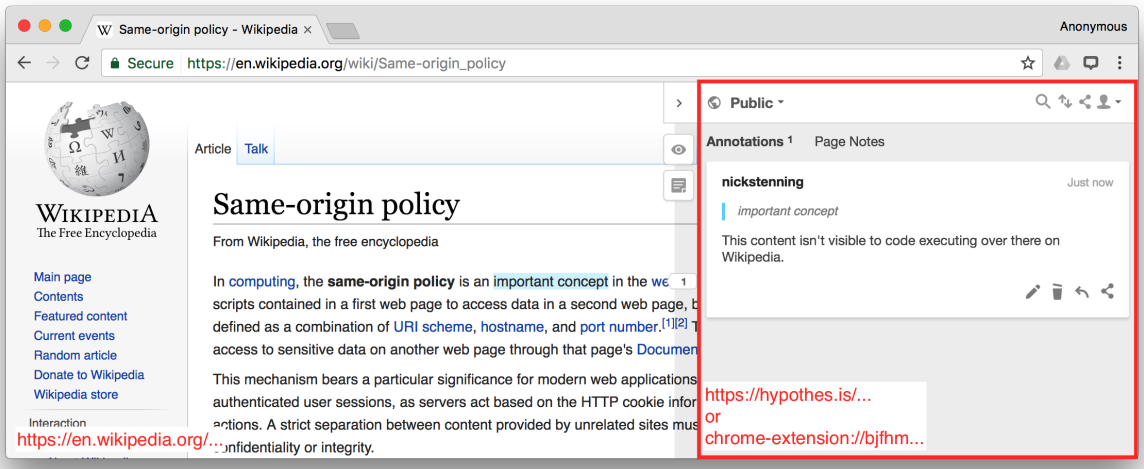


Fig. 2.1: Distinct origins for annotated content and client application

Input Sanitization

As alluded to above, the client frame is a trusted execution context. Any code running there has full access to everything the user has access to, which may constitute a major security flaw if that code was provided by another user (say, as a `<script>` tag in the body of an annotation).

This is an example of a cross-site scripting attack (XSS) and must be mediated by ensuring that **any and all user content displayed in the client frame is appropriately escaped and/or sanitised**.

Transport Layer Security

We ensure that it is hard to eavesdrop on traffic between the client and the annotation service by communicating with the annotation service over encrypted channels (`https://` and `wss://`).

Todo: 2017-03-08

This is not currently enforced by the client. Perhaps production builds of the client should refuse to communicate with annotation services over insecure channels?

Clickjacking Protections

The most straightforward way to protect an application from most kinds of clickjacking is the frame-ancestors Content-Security-Policy directive or the older X-Frame-Options HTTP Header.

Unfortunately, the client runs in a framed context (and on arbitrary origins) by default, so simply applying `X-Frame-Options: DENY` would break the client entirely.

Todo: 2017-03-08

The Hypothesis client would appear to have very little protection against clickjacking attacks that allow arbitrary websites to trick Hypothesis users into performing actions they did not intend to perform. It's not immediately clear what tools we have at our disposal to solve this problem.

Phishing/Imitation

At the moment there is little that would stop a website embedding a replica of the Hypothesis client in a frame and using it to harvest Hypothesis users' usernames and passwords.

Todo: 2017-03-08

Direct credential input must move to a first-party interaction (i.e. a popup window) where the user has the benefit of the browser toolbar to help them identify phishing attacks.

2.5 Architecture Decision Records

Here you will find documents which describe significant architectural decisions made or proposed when developing the Hypothesis client. We record these in order to provide a reference for the history, motivation, and rationale for past decisions.

See the introduction in the h repository for more details.

2.5.1 ADR 1: Component Architecture for Sidebar Application

Context

Background

Historically front-end web frameworks, including Angular 1.x, used a variety of MVC-based patterns for structuring the user interface part of an application , which of course is much of the code for a web app.

More recently (especially since React in 2013), web frameworks have generally moved to a simpler model where an application is structured as a tree of components with several key properties:

- Each component's internals are hidden from other components.
- Data is explicitly passed from one component to another and in one direction.
- Components have an explicitly declared API.
- Communication from child to parent happens via callback inputs.
- Components have a standard set of lifecycle hooks for creation, input changes and destruction.

A further pattern that emerged on top of this was to split components into those which purely reflect their inputs ("presentational" components) and those which are connected to services and state in the application ("container" or "connected" components).

This pattern made it easier to reason about, re-use and change pieces of the application in isolation, as well as enabling important optimizations and simpler framework implementations.

In Angular JS prior to v1.5, this pattern could be achieved by using a combination of features (element directives, isolate scope, bind-to-controller, controllerAs). Angular JS 1.5x introduced explicit support for this architecture via `.component()`.

Components in the Client

The client historically used traditional Angular 1.x methods of passing data between parts of the UI - properties on the scope inherited by child scopes and events. As the app grew larger it became harder to reason about where data in templates came from and who could change it ("$scope soup"). Newer parts of the UI have used element directives with isolate scopes to implement a component-based architecture, thus avoiding this problem. However:

- This requires a bunch of boilerplate for each directive.
- It isn't clear that we use this pattern from looking at the entry point to the sidebar.
- Important parts of the top level of the UI (the `*-controller.js` files and `viewer.html`) do not use this pattern and suffer from being hard to understand in isolation. Additionally they use a different mechanism (`ngRouter`) to control what is displayed than the rest of the UI, where we use `ng-if` guards.
- This lack of consistency makes it difficult to understand how the top level of the UI works.

Decision

- We will convert all element directives in `src/sidebar/directive` to components (ie. change them to use `angular.component()` and any refactoring this implies) and move them to `src/sidebar/components`. This change will be simple in most cases and will require some moderate refactoring for others.
- The top-level of the application will also be converted to a set of components using the same pattern and the router will be removed.

Status

In discussion

Consequences

In the short term this should make it easier to understand the sidebar app by improving consistency.

In the medium term, this brings the architecture of the client more into alignment with how it would be structured in other frameworks and gives us the opportunity to incrementally migrate to a more actively developed (and potentially smaller, faster, simpler) library for building our UI in future.

Presentational components can be potentially be extracted into their own packages for re-use in other parts of Hypothesis, though this is not an active priority.

2.5.2 ADR 2: ePub Support

Context

Background

Before now, the client had not had the necessary pieces to support the standard ePub book format. The primary obstacles that had to be traversed were 1) supporting iframe annotation, 2) document equivalency, and 3) scrolling to annotations.

Decision

- Standard ePubs viewers (including Readium and ePubJS) use iframes as a way to embed the different book pages and render them inside of their viewer. Hypothesis, before now, did not support going into iframes and allowing annotation. We modified our client to be able to watch for same origin iframes (cross origin not supported yet) on the page. When an iframe is encountered, we (if we have access to it) inject the Hypothesis embed code inside. In addition to the embed, we mirror the embed configuration values that were set. The last part of this injection is adding a "subFrameIdentifier" field to the configuration that is a random string that does two things 1) identifies the sub frame so the top frame has a point of unique reference and 2) the existence of this field tells the injected client that it should only load the Guest and not the full sidebar UI. So to recap, the client watches for iframes then injects the client and configuration into new frames. The Guest only clients inside of the frames will now be able to make and load annotations for their respective iframe locations. The client uses cross frame library to communicate what data to load - so subframes have those requests bubble up to the top frame. The sidebar stores an array of frames that it loads data for and does another cross frame call when data is returned. With all of that, the client is now able to support annotating content inside of an iframe.
- For document equivalency, we support for documents to set two new dc-* meta tags to indicate 1) what book we are in and 2) what chapter are we in. Together, this allows cross domain document equivalency down to the chapter level of an ePub. The tags are: "dc.identifier" for the chapter and "dc.relation.isPartOf" for the unique book identifier.
- Since ePubs use iframes for their presentation, those frames have content in various layouts/locations and move the visible area of the frame as the user is navigating to the next page (similar to how image sprites are used to show a single icon from a large image that has many icons). There is no standard event in web standards or ePub that we can use to navigate to the proper section in the frame and have it properly align the frame to include the page contents in the same manner that it does when you manually navigate to the page. That is, we could use "scrollTo" functions but those functions will just bring the section into view but makes no attempt to properly snap to the correct vertical and horizontal spacing that make up a whole page. This meant that unless we fixed it, users who select an annotation attempting to navigate to it could end up in scenarios where the highlighted section is visible but you cut the book's visible page in half. To fix this, we introduced an implementation agnostic "scrollToRange" event that we attempt to use before falling back to the traditional scrollTo event. That is, if the site registers a listener for "scrollToRange" and preventsDefault() we assume that they have taken the range and applied the proper scrolling behavior needed to get the range into view correctly.

Status

Deployed

2.5.3 Sidebar application architecture

This document describes a proposed plan to clarify the structure of the Hypothesis client's sidebar application.

Context

The Hypothesis client consists of three major pieces:

- The "boot" code which loads resources required by other parts of the client
- The "annotator" code which runs in the context of the host page
- The "sidebar" application which handles fetching, presenting and editing of annotations

The sidebar application is built on Angular JS [1]. Other than the components ADR the architecture has evolved in a slightly ad-hoc fashion.

For example, historically the app suffered from a number of problems related to performance [2] and testability [3] which motivated the introduction of a central state container. However it is not universally used for all the data needed to display the UI, creating confusion about what to use for new features.

In addition to inconsistencies, the structure that exists is difficult to discern without extensive reading of the code and new or occassional developers have found it difficult to grok. Poor naming of some modules also complicates understanding. For example the API client is called `store.js` (a reference to *Annotator Store*).

Proposal

Make explicit the implicit structure which the sidebar app's code has gradually been evolving toward over the last 12-18 months and to finish the transition. The sidebar app will consist of the following layers:

Store

- The *state container* [4] for the app, built on Redux.
- This maintains all the state needed to "draw" the user interface in a normalized (ie. avoiding duplication) representation.
- Organised into modules which maintain different aspects of the state.

Services

- Classes that handle interaction with external actors such as the host page, the annotation service (via the API, real-time API and OAuth endpoints) and local storage, scheduled/periodic tasks.
- These may maintain transient state (eg. the active `WebSocket` instance), but nothing required to "draw" the UI

Components

- Self-contained parts of the UI, each consisting of a template, styles and logic.
- These are connected to the services and store.

Utilities

- Helper functions and classes which do not depend on the other layers.

These layers will have the following dependency structure:

```
Utilities --> Store
   |           | | |
   |           V | |
   |-------> Services
   |           | | | |
   |           V V | |
   |-------> Components
   |           |   | |
   |           V   V V
   |-------> [ Entry point ]
```

The directory structure will be organized along functional lines in a similar manner to our Python apps:

```
src/sidebar/
  store/ # Redux store
    modules/ # Modules defining the logic related to a given part of the state
             # (eg. "annotations", "groups", "drafts").
  services/
  components/
  util/ # Utility classes & functions
  ...

  index.js # App entry point
```

Individual modules should be given clearer names to reflect their role in this new structure and avoid ambiguity.

Consequences

- The clearer directory layout should make it easier for new developers to understand the structure of the client and modify or add features, avoiding layering violations that create technical debt.
- If in future we decide to migrate away from AngularJS v1 we should be in an easier position to do so, since much of the code is independent of the UI framework and the part that does continue to use Angular (UI components) does so in a way which is conceptually compatible with most other frameworks.

Footnotes

[1] This was a popular choice at the time the code was originally written. As a framework it is now in maintenance mode and only receives updates for security or browser compatibility purposes. It is serving us adequately and there is no immediate pressure to migrate, but it is missing some of the performance and ecosystem benefits of newer frameworks, so I think it wise that we pave the way for this eventually.

[2] One of Angular's main tasks is to synchronize a DOM tree with a data source. The way it does this is essentially to *poll* the data source every time something *might* have changed and update every part of the DOM where the underlying data source has changed since the last update. In order to be efficient, the application's model needs to be set up in such a way that the total cost of executing all of these "watcher" functions is low. Redux's use of immutability is very helpful in this respect because it enables memoization.

[3] Various parts of the app related to updating the model were rewritten in a more functional style to reduce the reliance on mocking and thus improve the likelihood that a passing unit test translates into a working application.

[4] A *state container* is a centralized data store holding all the information needed to "draw" the UI, typically in a normalized form. Having this enables useful tooling (such as our debug logging) but more importantly it helps avoid bugs caused by not having a single source of truth.

Index