CPSC 490 Senior Project
Spring 2020
April 20th, 2020

Chaos

A novel system for organization of diverse digital data

Sarim Abbas

Supervised by Mark Santolucito and Professor Ruzica Piskac

Table of contents

Introduction

Much of the data we produce and collect now lives on the web, but it can be surprisingly fragile and ephemeral. Interesting content, including articles, images or streaming video may vanish due to link rot. Creative and productive tools, many now offered as web apps or services, may shutdown at any point, offering limited migration paths. It does not help that most of our data is stored in proprietary and inaccessible formats, instead of building on open standards or extending the capabilities of the operating system (OS). Moreover, the web apps we use daily are rarely interoperable, and so our data exists in silos, as evidenced by the dozens of app windows that we juggle in our daily workflows.

Chaos is a system for the organization, preview and archival of diverse digital information. It is driven by a few powerful assumptions about the inherent similarities of web apps and the data they produce. In doing, it aims to do for web app data what the OS has done for native app data. First, Chaos proposes an open, container file format that brings web apps into the realm of the file system. Second, it introduces a programmatic and graphical interface for the creation, editing and organization of these files. With its modular architecture, the use cases are theoretically infinite: users can archive links, write notes, hold conversations and pull data from their favorite apps, all in a single interface, with the proposed file format at its foundation. In sum, Chaos is an unconventional undertaking to wrangle the chaos of our digital age.

Principles

Note: Due to its broad ambitions, and the generality incurred by acting more as a modular platform than an app to complete particular tasks, it may not be immediately clear what Chaos is all about. It may be useful to skip ahead to the Modules section, before returning here. Learning about those implementation details may help develop an appreciation for the important principles that deserve their place at the top of this report.

The Chaos project subscribes to a number of principles which guide its implementation, which were referenced in the introduction:

1. Control
2. Harmony
3. Simplicity

Control

Web apps are attractive options for companies looking to implement SaaS[1]-y business models, with, admittedly, benefits for the user too, such as constant updates, cloud backup and live collaboration. Increasingly, however, user data is locked behind company servers, stored in proprietary formats, difficult to access outside of the web app's natural context, analyzed by black-box ML models, and sold to the highest bidder.

Chaos intends to return control back to the user by accessing all web data (including social media accounts, creative tools and other products) on their behalf. Crucially, this access is done client side, which means that all credentials and retrieved data are stored safely on the user's operating system. In fact, Chaos pledges, to the maximum extent possible, not to spin up a single server of its own to mediate user data. Neither does it bake in any non-essential telemetry, nor does it lock-in users with proprietary protocols and formats; all data is stored in a transparent format (discussed later). Much of the code is and will continue to be open-sourced. Users are free to leverage Chaos' developer API to build their own modules that access data on yet unsupported services (also discussed later).

Harmony

We use dozens of native and web applications every day, switching hundreds of times between them, to accomplish our tasks. Chaos promises, to the maximum extent possible, a unified interface for managing all user data. If we take a step back, we realize that most web apps are attractive interfaces for essentially CRUD operations on lists of data objects, which I refer to in this document as *information atoms*. Whether emails organized in inboxes, lists of todos (Todoist, Wunderlist), cards in Kanban boards (Trello)

[1] https://en.wikipedia.org/wiki/Software_as_a_service

or notes (Docs, Notion), Chaos argues that there is nothing uniquely different about each atom, and there is opportunity to exploit their similarities.

Chaos harmonizes all user data together in the computer's file system using its proposed file format. It also provides a common interface for organizing, viewing and editing these files. With its modular architecture, it encourages developers to set up communication between seemingly disparate services (dragging an email thread onto a social media post, for example), making possible the sorts of interactions and interoperability that technology companies have little economic incentive to provide.

Simplicity

Amidst the dozens of apps we use everyday, Chaos aims to bring back simplicity to interactions with our data. It does not invent yet another paradigm or metaphor for the simple task of storing lists of user data. Additionally, it strips down seemingly disparate apps to their common core, and provides a unified interface that makes possible access to all of them. Where there are genuine advancements in user experience, Chaos shamelessly adopts, modifies and extends to provide a minimal, productive and accessible experience for its own users.

Nor does Chaos reinvent the wheel by rebuilding the primitives that are already present in the operating system. In this web app era, the OS is left virtually unused, and ever more interactions take place in a company's cloud. But the OS gives us many of the tools we need already; a robust windowing system, performant search, and, by definition, offline access. Chaos simply utilizes these features in clever ways to do the same for web app data what the OS has hitherto done for traditional files.

Related work

A comprehensive review of existing solutions was done, which was organized into three categories.

1. Web clippers
2. Document editors
3. Specialized web browsers

Web clippers

Web clippers save content from the web, often to another web app's data model and rarely to a file on the operating system. OneNote, Evernote, Walling, Are.na, Zotero and many more apps and services provide such a clipper. But these clippers are intended to archive static content, not dynamic web app data. These clippers are also one-size-fits-all and are not extensible for specific use cases.

Document editors

Certain document editing apps, such as Notion, incorporate embed functionality to integrate data across several web apps such as Drive, Slack etc. But this interoperability is still achieved via proprietary protocols, and the company's employees are responsible for developing these integrations. When economic incentives do not align, it is all too likely that these integrations will disappear. Some document editors such as Bear use a container format such as **`.textbundle`**, similar to the one proposed by Chaos, but these containers are not widely adopted and not generally applicable to all web app data, only text.

Specialized web browsers

Franz, Rambox and Station are examples of GUIs that seemingly unite multiple web apps in a single window. But on closer look, they are just specialized browsers that pin the common web apps into a sidebar (similar to tabs). There are few features (with perhaps the exception of unified search) to harmonize the multiple web apps together.

Architecture

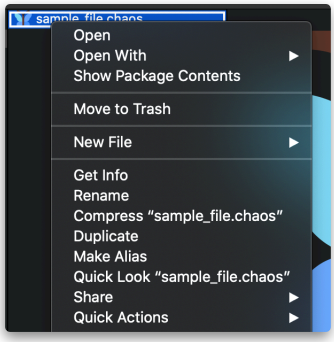
All referenced code can be found in the project's GitHub repository[2]: https://github.com/sarimabbas/chaos.

File format

The proposed **.chaos** file format stores web app data (although technically it can store any kind of data).

The **.chaos** specification was simultaneously simple and difficult to implement. Simple, since writing the specification simply required agreement on a set of conventions for how interfacing software should read and write **.chaos** files. Difficult, since the conventions needed to be flexible enough to meet the currently stated aims of Chaos, as well as any uses that develop for it in the future. Please see XKCD 927 about the dangers of this kind of wishful thinking[3].

A **.chaos** "file" is actually a folder. On macOS, folders with the **.chaos** extension are registered as document packages to give the illusion that the user is working with files in Finder[4]. By right-clicking the package, and clicking "Show Package Contents", the user can inspect its contents. I will subsequently use "package" and "file" interchangeably.



A minimally valid **.chaos** package has the following structure:

```
— my_website.chaos
—— manifest.json
```

The file package can also contain other assets: images, PDFs, other folders, binaries etc. The basic idea is that the GUI will read the **.chaos** package, look for a manifest, and refer

---

[2] https://github.com/sarimabbas/chaos

[3] https://xkcd.com/927/

[4] https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/DocumentPackages/DocumentPackages.html

to any assets stored in the package to do whatever rendering it needs to do. Here is an example of a website being archived within a **.chaos** package:

```
— my_website.chaos
—— manifest.json
—— assets
———— index.html
———— cat.png
```

This use of packages is nothing new. For instance, the **.textbundle**[5] specification was proposed to store images side-by-side with Markdown text. But where **.chaos** differs is in its generality: it's not just for text, but for any kind of digital data. You can use it to write notes, but also archive websites, credentials and configurations, chat messages, emails etc.

More about the manifest: it's a **.JSON** file that follows a structure of its own. The specification and validator is available at **github/chaos/atom.js**[6]. Here's a look:

```
static spec = Joi.object({
    // shared metadata for all atoms, where possible
    shared: {
      // descriptors (besides the filename)
      title: Joi.string().allow(""),
      description: Joi.string().allow(""),
      image: Joi.string().allow(""),
      // general organization (e.g. Kanban board)
      category: Joi.string().allow(""),
      tags: Joi.array().items(Joi.string().allow("")),
      // for todo functionality
      completed: Joi.boolean(),
      deadline: Joi.date().timestamp(), // integer UNIX time
    },
    module: {
      id: Joi.string().required(),
      name: Joi.string().required(),
      // module specific properties:
      // assetsPath: "",
      // apiToken: ""
    },
  });
```

The popular Joi library[7] was used to validate the specification in JavaScript. Since the specification is open-source, it can also be parsed and validated in other languages with types or a typing library.

---

[5] http://textbundle.org/

[6] https://github.com/sarimabbas/chaos/blob/master/src/backend/atom.js

[7] https://github.com/hapijs/joi

I would not consider this specification to be complete. It is a living document, that will evolve as the needs of users change, and will thus be prefixed with Semantic Versioning[8] to mitigate compatibility problems.

A lot of the properties come under a "shared" label. Chaos is driven by the recognition that most applications, no matter how avant-garde they seem, store pretty homogenous-looking data, usually just lists of atoms (Trello: cards, Excel: rows, Evernote: documents, Notion: documents, Docs: documents. Think of an app. Does it store lists of things? If so, its atoms are probably good candidates for being wrapped with **`.chaos`**). And they usually have properties like "title", "description" or "tags". We can wrap up this data inside **`.chaos`** packages, and refer to their common properties. You might already get a sense of why this is useful, but I'll be explicit when explaining the GUI design.

Chaos recognizes that every atom has its "secret sauce". An Instagram post might have a "likes" count, or a YouTube video might have a "comments" array. These properties are not general enough to be shared with other **`.chaos`** files. So they come under a "module" label instead. The creating module will store its unique ID and name (e.g. **`com.instagram.chaos`**, "Instagram") and these properties.

Graphical user interface (GUI)

Technologies

The Chaos interface is built with Electron[9], a framework that lets developers build cross-platform desktop applications with web technologies i.e. HTML, CSS and JavaScript (JS). It bundles Node.js[10] so that the web app can access the local file system and other system APIs. It also bundles the Chromium[11] browser so that the app is rendered consistently across platforms. Thanks to this, we get bundle sizes of approximately ~150-200 MB. Chromium is also a resource hog, so memory usage is generally quite high.

Earlier experimentation with the interface used PyWebView[12], a Python framework that swaps out Electron for a Flask[13] server and a system web view (e.g. WKWebView[14] in macOS). Bundle sizes came down to ~15 MB and memory usage was negligible. It makes

---

[8] https://semver.org/

[9] https://www.electronjs.org/

[10] https://nodejs.org/en/

[11] https://www.chromium.org/

[12] https://github.com/r0x0r/pywebview

[13] https://github.com/pallets/flask

[14] https://developer.apple.com/documentation/webkit/wkwebview

sense that this is the way of the future for web-based desktop apps, but the lack of polished system APIs makes development very difficult. Similar options investigated include Carlo[15] and zserge/webview[16], which are under active development but lack comprehensive documentation and a community. The footnote density of these previous two paragraphs is representative of the fact that web-based GUI development is a rapidly progressing area, but as of yet there are frustratingly few canonical ways to build reliable and performant cross-platform GUIs. In theory the Chaos interface can be built with any GUI framework, for instance the native SwiftUI framework for macOS, since the only requirement is that the interface respect the **`.chaos`** format conventions. For my prototype, however, I fell victim to the write-once deploy-anywhere dream and ultimately opted for Electron.

Vue[17] is also used: a JS frontend framework that adds convenient features like declarative rendering, components, scoped CSS, global state and routing. It sounds like jargon, but each of those characteristics reduced development time considerably. For instance, declarative rendering means that the view is a direct function of data, unlike vanilla JS where the DOM is mutated directly. This allows the interface to be reactive and reduces state-related bugs. Components allow the interface to be broken into reusable pieces, for example, a commonly used button component. Scoped CSS prevents global namespace clashes when styling individual components. Global state simplifies state management by allowing all components to interact as if they were communicating over a publish-subscribe model. And routing in a JS app gives the illusion of page navigation when in reality it is simply a way to swap out pieces of the DOM. While Vue is not necessary to build a web-based interface, it is highly productive. My command of Vue grew significantly over this semester, as I worked with recursive components, abused it to set up an event bus, and put together a primitive file watcher.

Lastly, the Tailwind CSS framework was used to style many of the components. Unlike other popular CSS frameworks like Bootstrap, it does not prescribe any opinionated components of its own, but rather provides utility classes that allow for rapid prototyping (sometimes this approach is called functional CSS). This, along with CSS variables, also helped me lay the groundwork for a flexible user theming system (discussed later).

Layout

Using CSS Grid[18], a multi-pane layout was built. The interface borrows heavily from the popular VSCode[19] editor with modifications. For instance, a left sidebar contains the

---

[15] https://github.com/GoogleChromeLabs/carlo

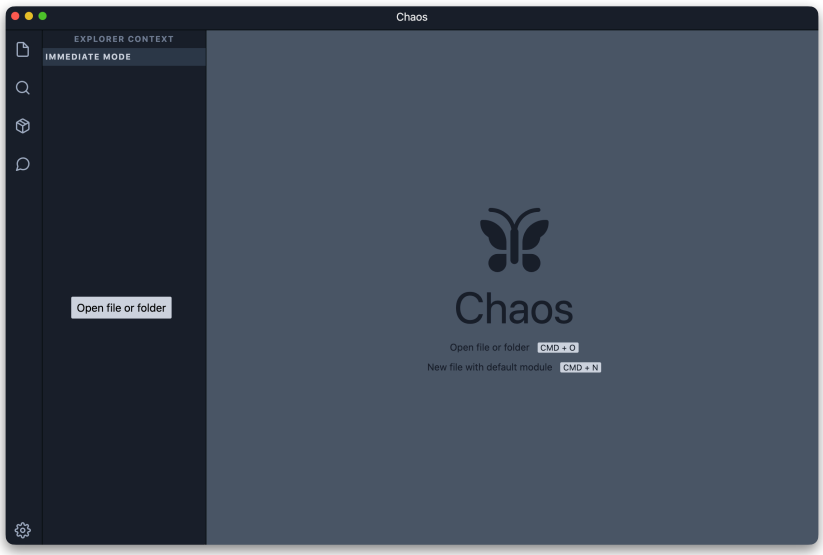
[16] https://github.com/zserge/webview

[17] https://vuejs.org/

[18] https://developer.mozilla.org/en-US/docs/Web/CSS/grid

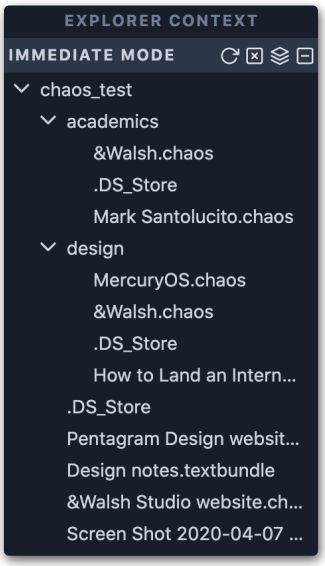
[19] https://code.visualstudio.com/

familiar file explorer, search, and module store. Where Chaos differs from VSCode is that, while the latter is exclusively an editor for source code, Chaos is a previewing and editing interface for diverse web-app data stored in **`.chaos`** files.



File explorer

The file explorer walks the OS file system and outputs details for each inode using the **`lstat`** system call. Currently this traversal is done with JavaScript, but may be replaced with a compiled routine (e.g. Rust) in the future to improve performance for large file trees. This is a common technique used in Electron apps.



The retrieved data is rendered recursively by a tree view component that was built in Vue over several weeks. Off-the-shelf components are available, but they lack flexibility e.g. hover effects, right-click context menus and multi-root rendering. The work on this

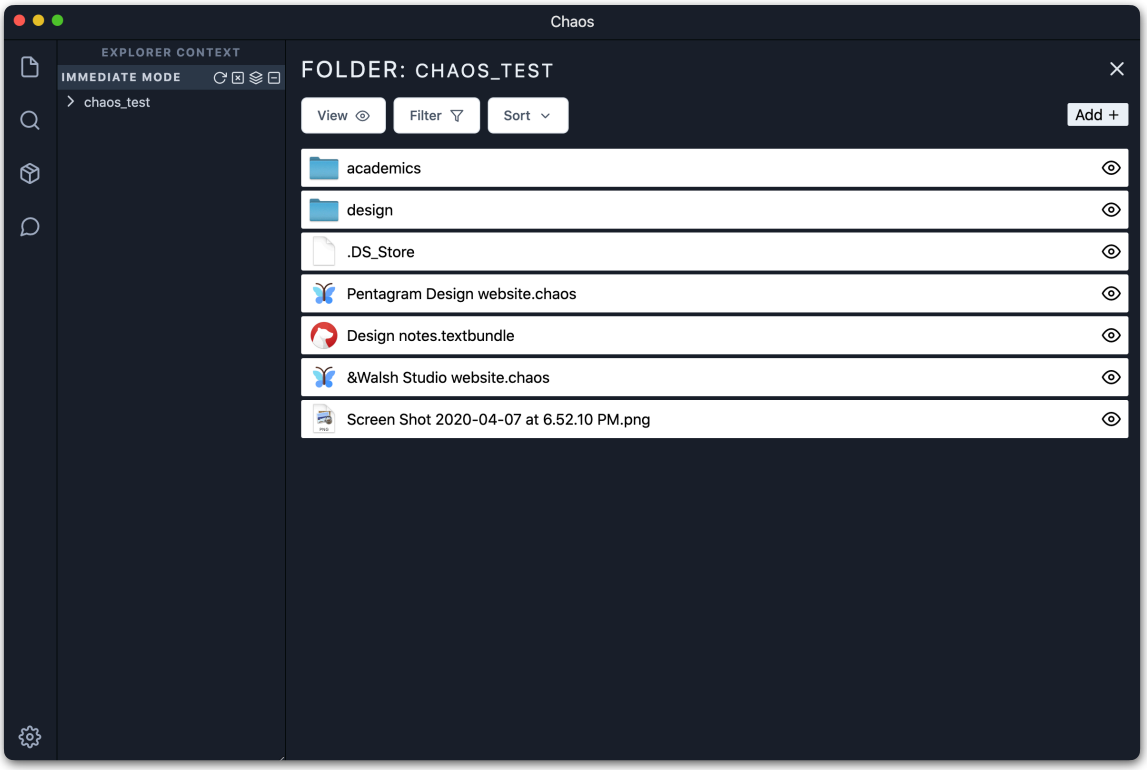
component is extensive enough that it warrants being spun out into its own open-source repository, and published on the Node Package Manager (NPM) registry for use by other developers.

An interesting feature is a “nested” mode, which fetches both a directory’s data and (recursively) the contents of its subdirectories. These contents are populated in the folder view (below). This is a useful organizational trick used in popular note-taking apps like Bear and Notion, which is easily replicated for the local filesystem with a small modification to the traversal algorithm.

Folder view

When a node in the tree view is clicked, and if it is a folder/directory, its contents are displayed in this view.

The folder can itself be viewed in different ways. By default, the contents are presented as a list. There is also an option to view as a grid, with rich text and image previews. The previews are made possible with the shared properties in the manifest of each **.chaos** file, although in the future these previews can be generated for any type of file using system APIs.



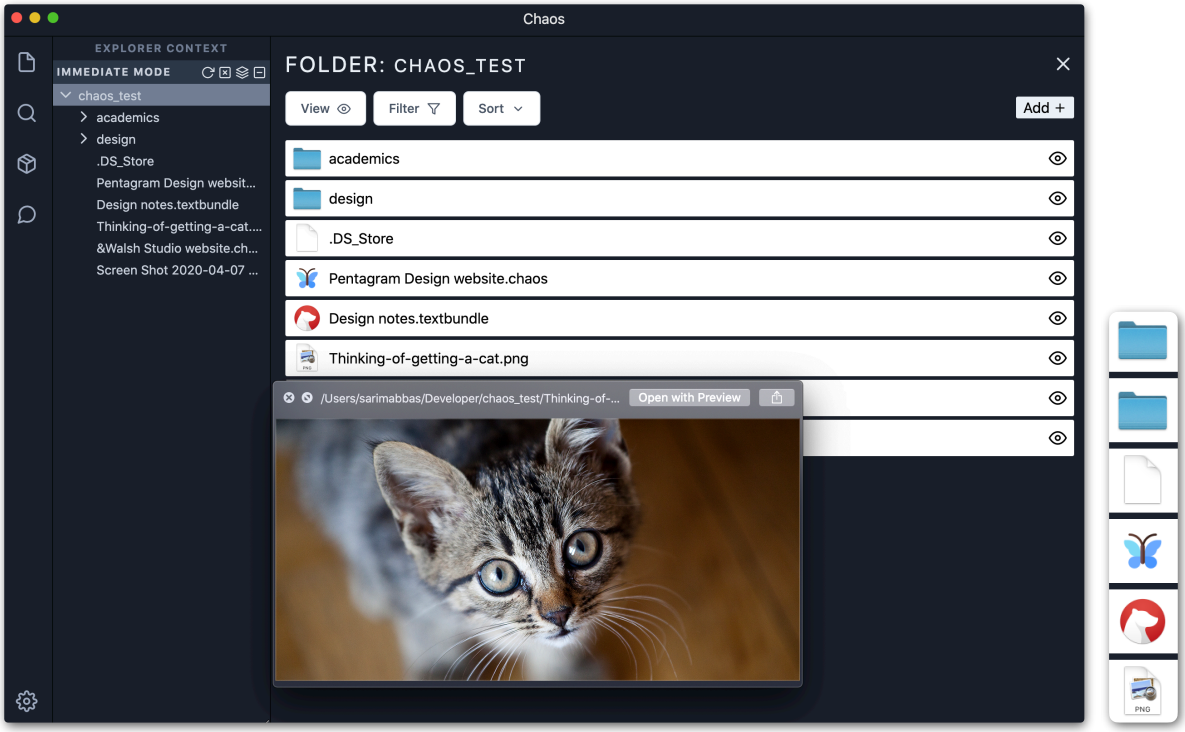
Borrowing from popular tools like Airtable and Notion, it is possible to introduce even more view options, for example, a Kanban view that utilizes the “category” property in the manifest to organize a folder’s contents into vertical columns, or, a Todo view that uses

the “deadline” timestamp to sort items by their due date. In other words, it is possible to get the defining visuals of popular apps for “free” by simply wrapping their surprisingly homogenous data in **.chaos** files, and using shared properties to construct these views. More importantly, this means that any data stored in a **.chaos** file can participate in these views; whether a note, website or email, all of them can have deadlines, categories and tags attached.

Borrowing further from Airtable, the folder view also has sort and filtering options to further organize its contents. With these features, and its emphasis on offline-first, file-system based data storage, Chaos essentially becomes an upgraded Finder. Indeed, I propose that the vision for Chaos should also be adopted by today’s operating systems, which should similarly outfit their file explorers to retrieve, store and organize web app data.

Any files which are not of the **.chaos** type (e.g. **.pdf**), can be previewed using the QuickLook API on macOS by clicking the “eye” icon. They can also be clicked to open them in the default application for that file type. This allows the user maximum flexibility in using the Chaos interface for their base organizational needs, as well as in editing unsupported file types with existing applications on their computer.

A technical aside: using a combination of system APIs, the relevant system icons for each file extension (e.g. **.chaos**, **.pdf**, **.png**) are retrieved, just as in Finder. Some engineering was done to make this retrieval as efficient as possible, using a cache of Base64-encoded icon images.



Modules

Modules provide the core of Chaos' functionality. They are most analogous to extensions in VSCode. Modules must broadly fulfill two requirements:

1. Provide a visual or programmatic interface for creating **`.chaos`** files
2. Visualize or render data using those files

For web apps whose data maps neatly onto information atoms (macOS refers to these as document-oriented apps), a module might connect to the web app on the user's behalf, fetch the data it needs, and render it for the user.

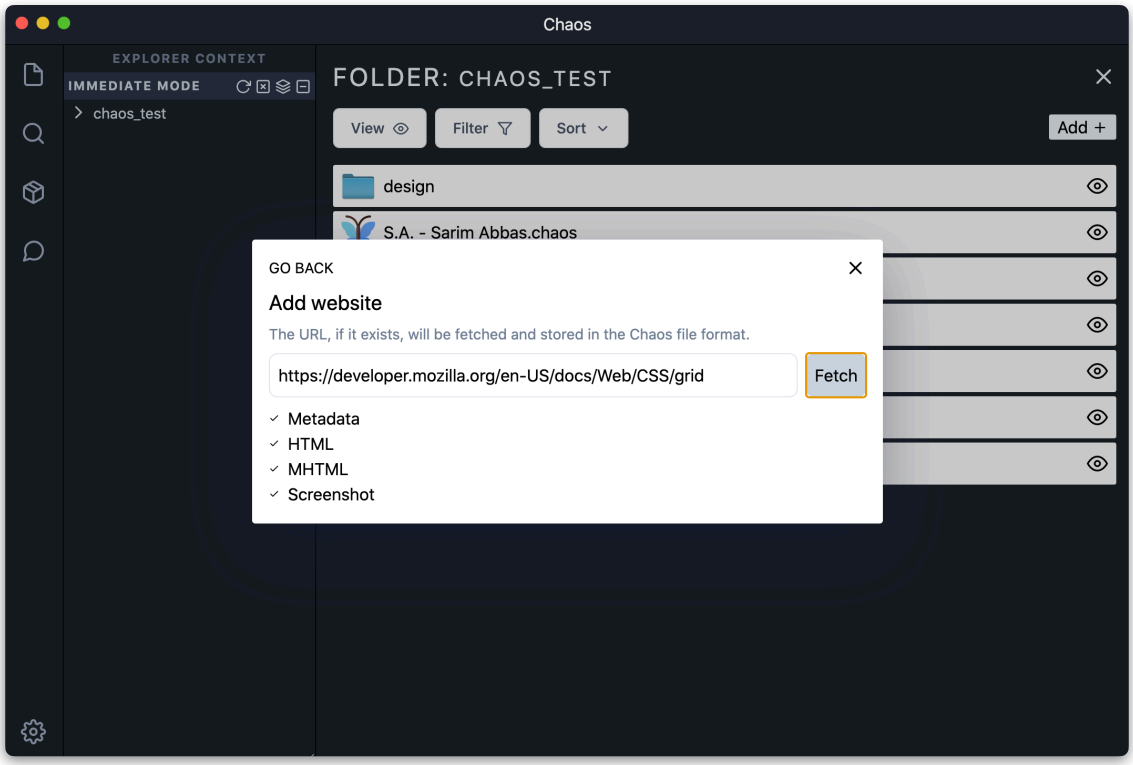
The exemplar modules below are currently they are baked into the core, but the codebase is oriented towards providing a robust developer API and module submission system in the future (see: Next steps).

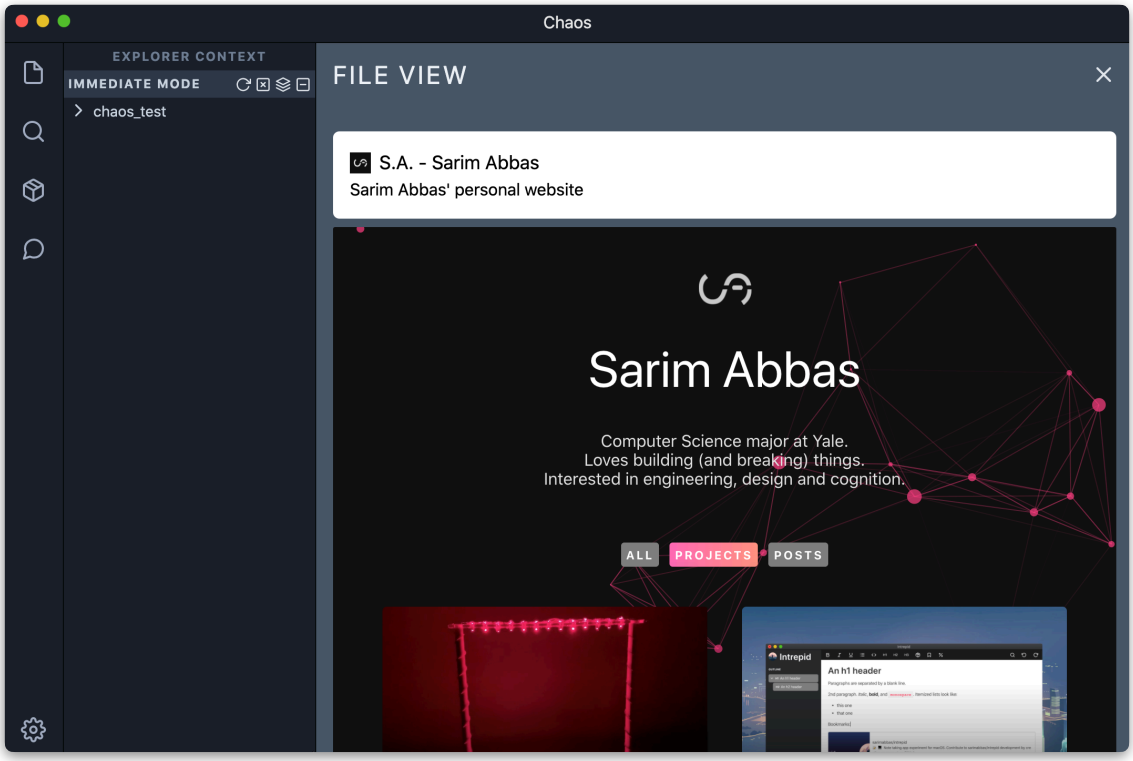
Website module

Here we see a concrete example of a module; its use of the **`.chaos`** file format, and visual interface. The website module was built as a proof-of-concept. It is intended as a bookmarking or Pocket[20] replacement, allowing the user to archive any link.
When clicking the "Add" button in the folder view, the user is presented with a list of modules that support adding information atoms. The user chooses "Website", and enters a link. The module then uses system and Electron APIs to retrieve the following: metadata (e.g. a title and description), a screenshot, PDF, MHTML and HTML archives. This allows both redundancy and versatility of access. For instance, the PDF can be opened for annotation, while the MHTML archive can be browsed offline. The redundancy also ensures that the dynamic, JavaScript-heavy websites of today are faithfully preserved.

This module is somewhat general, and more specialized solutions are needed for archiving specialized information atoms e.g. Google Docs. Nevertheless, the preview includes a live version of the link, which still allows for some of these specialized use cases (e.g. Docs can be edited in this way).

[20] https://getpocket.com/

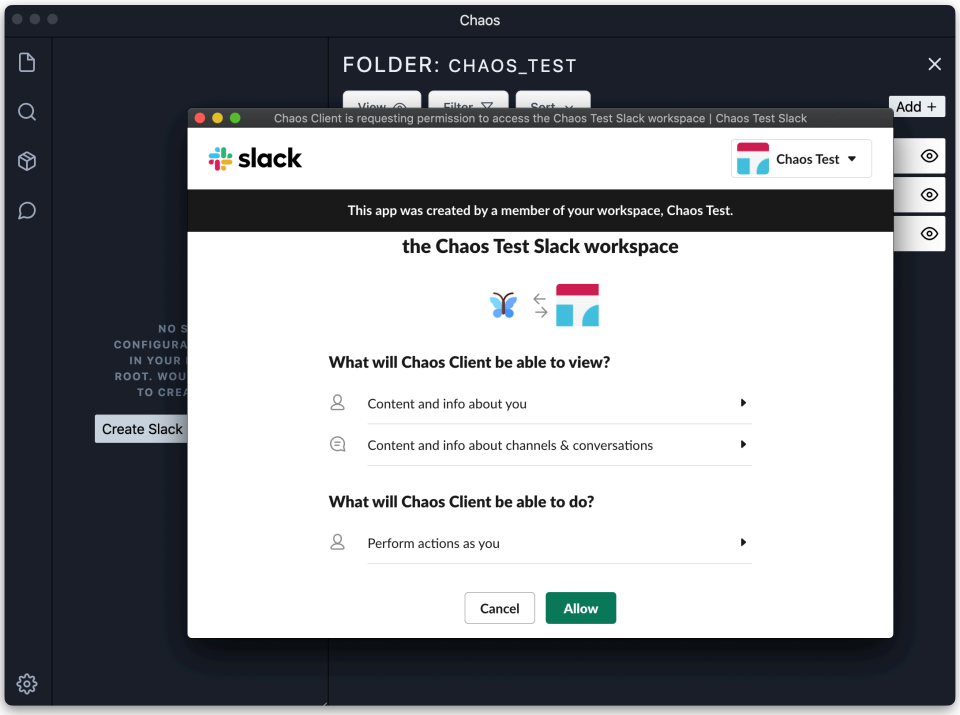

Chaos
EXPLORER CONTEXT
IMMEDIATE MODE
chaos_test
FOLDER: CHAOS_TEST
View
Filter
Sort
Add +
design
S.A. - Sarim Abbas.chaos
GO BACK
Add website
The URL, if it exists, will be fetched and stored in the Chaos file format.
https://developer.mozilla.org/en-US/docs/Web/CSS/grid
Fetch
Metadata
HTML
MHTML
Screenshot


Chaos
EXPLORER CONTEXT
IMMEDIATE MODE
chaos_test
FILE VIEW
S.A. - Sarim Abbas
Sarim Abbas' personal website
Sarim Abbas
Computer Science major at Yale.
Loves building (and breaking) things.
Interested in engineering, design and cognition.
ALL
PROJECTS
POSTS

Slack module

This module is an example of how the **`.chaos`** format can be leveraged not just for storing documents or information atoms, but rather credentials that can be used to render streams of data. In other words **`.chaos`** files can occupy the data-as-code[21] niche.

With a workspace open, the user can click the Chat icon in the sidebar to view the Slack pane. Slack is a popular messaging app used by teams. The module attempts to locate a valid Slack configuration for the workspace. If none is present, the user completes an OAuth flow to authenticate with a particular Slack team, and the relevant access tokens are stored in a **`slack.chaos`** file in the workspace root.

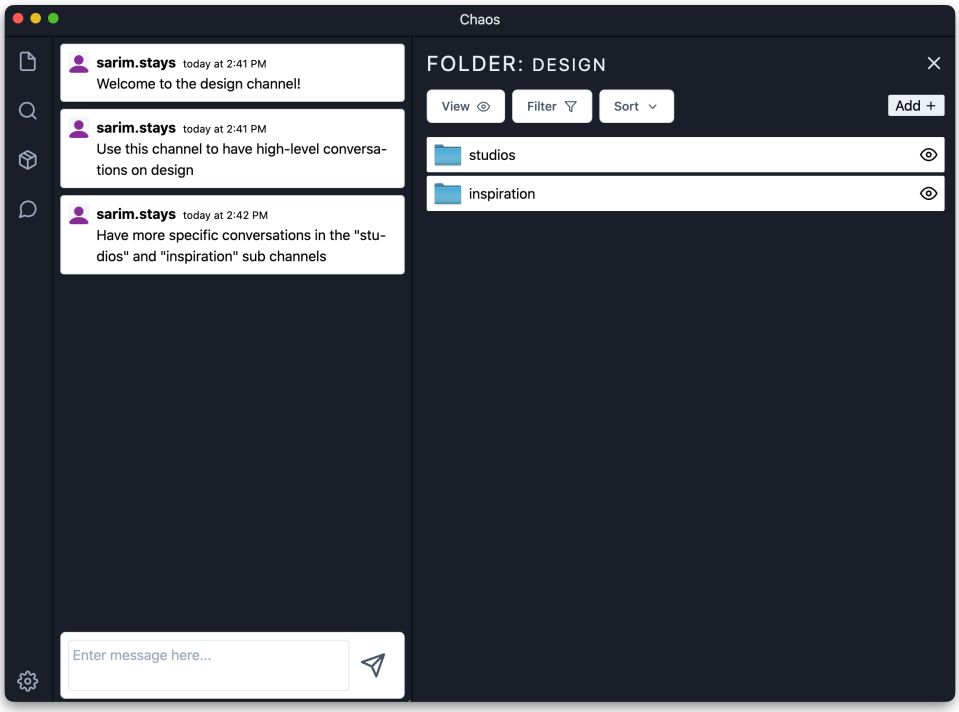


Each folder in the workspace mirrors a Slack channel, and the module makes intelligent guesses using folder and channel names to make this guessing happen. For instance, when a folder named “design” is opened, the module will retrieve the chat history for the “design” channel. The messages are retrieved in real-time using WebSockets[22], and messages can also be sent using the input field. The big idea is to keep chat history and files in the same place, because so often our conversations center around files anyway (e.g. “Can you take a look at that spreadsheet?”, “I updated the presentation” etc.)

---

[21] https://en.wikipedia.org/wiki/Code_as_data

[22] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API



Chaos modules can actually make existing web services more powerful. When the Slack module cannot find a channel synonymous with the currently selected folder, it offers to create it. But it does so in a nested fashion. For instance, if the “design/inspiration” folder is selected, the module will offer to create a Slack channel named “design___inspiration”. This effectively lets users retrofit Slack with nested channels, which is currently not possible with Slack proper.

Furthermore, in the future, the Slack module will be updated to retrieve messages for a channel recursively i.e. from all nested channels. This will allow the user an eagle-eye view of all messaging across the channel tree, and to zoom in on a particular conversation by just clicking to the relevant folder/channel. For e.g. consider the following folder/channel structure:

```
– design
—— studios
—— inspiration
```

By clicking on the parent “design” folder, a user will be able to see aggregate message history from across the “design”, “studios”, and “inspiration” channels. In order to further filter messages, a child folder can be clicked.

Usability testing

Protocol

The objective of the usability test was to identify and fix flaws in the interface, as well as prioritize features for next releases.

I recruited five participants to test a prototype release of the software over video chat. First, I provided context about the purpose and goals of the app. I then asked the participant to complete a number of tasks in sequence while sharing their screen (I also requested permission to record their interactions):

1. Open a folder as a workspace using the file explorer
2. Use the Website module to save a link
3. Use the Slack module to connect to a Slack workspace
4. Discover and use folder viewing options

During the test, I encouraged the participant to provide candid feedback, and to voice their thought process while using the app. I was careful in providing minimal guidance so that they could complete the tasks mostly by themselves.

At the end of the test, I debriefed the participant about what the study was for and how the collected data will be used, and I also assured anonymity.

Results

Measures of success were a mix of qualitative and quantitative, including subjective user discomfort, number of clarifying questions asked, time taken to complete tasks, and number of mis-clicks. Since the participant sample size was small, no hard statistics can be calculated, but there was still plenty of valuable feedback.

All participants completed the assigned tasks, with noticeable variation in success measures that was a function of their age, self-described tech savviness and familiarity with other productivity software.

Some of the biggest obstacles to task completion were incomplete components of the interface which acted as distractors from the task, for example, the “search” and "module store” components in the sidebar. For the next round of testing, these will be hidden so that participants are not confused when attempting to follow task instructions.

The test helped prioritize next features. Before the test, I had planned to develop a Notes module, but I noticed how participants expected common folder operations such as “rename”, “delete” and “copy” which were not present. And so, I now plan to implement these essential features before moving onto more ambitious additions of functionality.

Small pain points also emerged, which are easy wins to fix and will result in significant increase in polish. For example, one participant commented on how the file explorer component does not invite interaction since it is fully collapsed when the folder is first added to the workspace. By expanding the file tree by one level, it can improve visibility, and this is now planned for a future release. Similarly, another participant wanted to open the live version of a website (saved with the Website module) in their default web browser. Even though the functionality is present, it was not conspicuous, and this will be rectified in a future release.

Next steps

Views

In addition to the list and grid views, a Kanban view and todo view are also planned. The Kanban view will mimic the capabilities of Trello, allowing the user to organize their **`.chaos`** files in columns, using the “category” property in each file’s manifest. The todo view will mimic the capabilities of Todoist, and sort files by the “deadline” property in the manifest.

Modules

New modules

Additional modules are planned that highlight Chaos’ potential. A Markdown-based note-taking module, based on the existing app Intrepid[23], is being integrated into the interface. It will utilize the **`.chaos`** format to store both text and assets, similar to **`.textbundle`**.

Developer API

An unofficial API is already present which can be accessed in any Vue component using **`this.$chaos`**. In fact, the website and Slack modules use this API. It provides access to the file system and certain Electron capabilities for developers to leverage in their own modules. But security and privacy is a concern, and some work has to be done in this area before an official release.

Module store

Approaches are being investigated to allow developers to submit their modules to a registry, where they can be approved and made available for users to download and install in their Chaos interface.

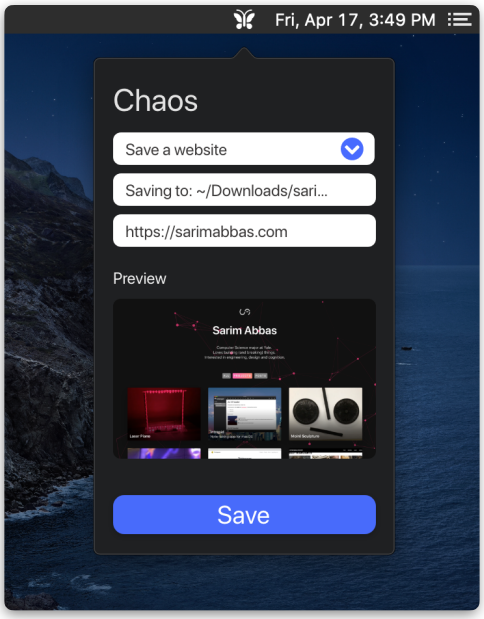
macOS improvements

Menu bar

In an effort to reduce friction, and prevent users from having to open the full GUI, a menu bar interface is planned, through which modules can expose functionality to create **`.chaos`** files. For instance, users will be able to quickly add web links via the website module in the menu bar.

---

[23] https://github.com/sarimabbas/intrepid



Quick look plugin

As a further attempt to reduce friction, a quick look plugin is planned that intelligently generates a preview for any **`.chaos`** file within Finder.

Public communications

Concrete use-cases (the interface, modules etc.) are being developed to make Chaos more than just a concept. As mentioned at the beginning of this report, it is not immediately clear what the system achieves without looking at some of its implementation details. Nevertheless, work needs to be done on refining the way the principles and goals of this project are presented to the public. There is real danger in adopting marketing language that is so ambiguous that it dissuades use of the system. Many productivity tools purport to do everything and solve all organizational problems but ultimately do not make an effective case for why their paradigm is uniquely better. It is very important that Chaos not go this route, and that its approach stands the test of time.

Chaos was recently accepted to the Beyond Boundaries symposium, an initiative by the Digital Humanities Lab at Yale University. The symposium is intended to highlight hybrid scholarship in the humanities. Chaos was presented as versatile tool to assist museum curators in archiving all forms of digital history: articles, streaming media, feeds of data and more. There is plenty of scope for museums to develop their own modules to render specialized visualizations (e.g. 3D fossil scans, photospheres etc.) from raw data, with the added benefit that all data lives offline and can be backed up with existing infrastructure (tapes, drives, NAS etc.). Additionally, modules were proposed that integrate diverse data to reveal interesting links. This has hitherto been the charge of experienced historians, but with the unified **`.chaos`** format, a module could automate this analysis using a similarity measure based on multiple metrics, such as time period, creator of the work,

and topic modeling. Such modules could power an intelligent curation mechanism that allows institutions to automatically organize their collections.

Perhaps the path forward in communicating Chaos to the public is to develop specialized narratives for each discipline that demonstrate its capabilities, as was done with Beyond Boundaries. This may best demonstrate how Chaos can occupy a unique place in users' daily workflows.

Acknowledgements

Many thanks to Mark and Professor Piskac for their support and in encouraging me to be ambitious with my work. My parents and Minaam, for being my sounding boards. My earliest prototype tester, Felicia. All my friends for their input and ideas.

References

- “Code as data,” Wikipedia. Jun. 04, 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Code_as_data&oldid=900197220.
- “Document Packages.” https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/DocumentPackages/DocumentPackages.html .
- “Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.” https://www.electronjs.org/ .
- GoogleChromeLabs/carlo. GoogleChromeLabs, https://github.com/GoogleChromeLabs/carlo 2020.
- “grid,” MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/CSS/grid .
- hapijs/joi. hapi.js, https://github.com/hapijs/joi 2020.
- Node.js, “Node.js,” Node.js. https://nodejs.org/en/ .
- pallets/flask. The Pallets Projects, https://github.com/pallets/flask 2020.
- “Pocket.” https://getpocket.com/ .
- Roman, r0×0r/pywebview. https://github.com/r0×0r/pywebview 2020.
- S. Abbas, sarimabbas/chaos. https://github.com/sarimabbas/chaos 2020.
- S. Abbas, sarimabbas/intrepid. https://github.com/sarimabbas/intrepid 2020.
- T. Preston-Werner, “Semantic Versioning 2.0.0,” Semantic Versioning. https://semver.org/ .
- “Software as a service,” Wikipedia. Apr. 07, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Software_as_a_service&oldid=949597498.
- “Standards,” xkcd. https://xkcd.com/927/ .
- “The Chromium Projects.” https://www.chromium.org/ .
- “The WebSocket API (WebSockets),” MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API .
- “Visual Studio Code - Code Editing. Redefined.” https://code.visualstudio.com/ .

- “Vue.js.” https://vuejs.org/ .
- “Welcome to TextBundle.org!” http://textbundle.org/ .
- “WKWebView - WebKit | Apple Developer Documentation.” https://developer.apple.com/documentation/webkit/wkwebview .
- S. Zaitsev, zserge/webview. https://github.com/zserge/webview 2020.