Native Slack app for macOS

Vision Statement

Quyen Tran, Luming Yin, Seong Moon
Jack Weinkselbaum, Kirill Korygin

1. Motivations

According to TechCrunch[1], Slack has 6M daily users and 1.5 million paying users. With that, Slack has become the industry standard for real-time team collaboration and communication around the world. Based on our own experience, our colleagues' experience and Slack users' written complaints on online communities such as Twitter, the official Slack app for macOS can be vastly improved to offer streamlined navigation and improved performance.

As with most Silicon Valley startups, the official Slack app for macOS was architected at a time where it was operating at a much smaller scale. Its design centers around individuals who have only joined one or two teams, hence navigating between conversations from different teams is cumbersome and slow. Its technical architecture, which uses Electron paired with a series of third-party node modules delivers subpar performance, uses a significant amount of computational resources and drains the battery quickly.

One of the main reasons Mac users love using their Mac is because of high-quality first-party and third-party apps. These apps are simple-to-use, powerful and power-efficient. Mac users are familiar with high quality, well-designed software that conforms to the macOS Human Interface Guidelines, which the Slack app fails to achieve in many ways. The 10-hour battery life amongst Mac laptops is possible because third-party apps are built against macOS natively - using AppKit and Cocoa, instead of spawning its own Chromium and Node instance. The official client aligns its interest in cross-platform compatibility. In other words, it is optimized for ubiquity and ease-of-development instead of simplicity for the users. Because Slack's API is mostly undocumented, there is no third party client that is compatible with existing Slack teams and channels. As a result, existing solutions are not fitting for our purpose. Table 1, shown below, outlines the problem our product will work to address.

Table 1. Problem Statement

The problem of	Users of Slack for Mac waits for a long time for the app to start, and the official app’s hungriness slows down other running applications on the computer.
Affects	Slack for Mac users, especially those who are a part of multiple teams or use the app heavily.
The impact of which is	Team members waste time waiting for the Slack app to load instead of collaborating and communicating.

[1] https://techcrunch.com/2017/09/12/with-more-than-6m-daily-users-slack-opens-up-cross-organization-teams/

A successful solution would be	A native macOS Slack app with a streamlined UI, improved responsiveness, and reduced battery consumption

Our product is designed around the macOS Human Interface Guidelines, being flexible, expandable, capable and focused. Our solution provides multiple ways to organize and view messages from each channel, including a unified timeline, a unified source list and a conventional hierarchy akin to the official client. Its unified source list is expandable, enabling it to display all channels from every team the user has joined so that users do not have to constantly switch between teams to view messages from different channels. This streamlines the navigation between different teams, which is a constant pain point for power users and allows users to focus on the important matters - collaboration and communication.

Because our solution is built natively, we can apply a wide range of techniques to speed up its launch time and performance. Because we no longer need to spawn a new Chromium instance per team and evaluate each team's transcript in a separate JavaScript context, its launch time, CPU and memory usage are significantly lowered. This ensures Slack does not consume excessive battery energy and does not slow down other apps. Our solution also caches the state of each team (such as team member list, channel list and message transcript), therefore enable the app to launch as soon as a WebSocket connection with the server has been formed, rather than launching only after dozens of REST request calls has been responded. A preliminary implementation shows these techniques can allow our app to launch around ten times faster than the official app. Table 2, shown below, describes our product position.

Table 2. Product Position

For	Slack for Mac users
Who	Communicate with their co-workers or team members from multiple teams
Our system	macOS app built with natively with AppKit
That	Allows power Slack users to navigate between multiple teams and channels more easily and efficiently
Unlike	Slack’s official Electron-based cross-platform desktop app
Our product	Is more efficient both from a user experience perspective and from a technical performance perspective

2. Users

While our solution is optimized for those with high workloads in regards to Slack, our solution targets all Slack for Mac users - whether they have just joined their first team, or are organizers of multiple teams, our solution needs to be both intuitive, powerful and performant.

Because users of our solution are mostly existing Slack users, our app's interface needs to take some familiar design cues from the existing Slack web app, such as the tone-on-tone contrasting color scheme between source list and the conversation transcript. Our app should also be approachable to all users who have used the built-in Messages app on a Mac, which means no unexpected use of system UI widgets. For example, by default, we need to respect the expected behavior of NSTextFields, where pressing the return key sends the message, rather than creating a second line of an existing message.

As many potential users may not be experienced in computer science, the interface needs to be tolerant of mistakes, where all actions must be easily recoverable. Therefore, elements common in pro-level software, such as panels that can be torn apart from the main interface to be put on an external display, should not be implemented. For sidebars and panels, they should be expanded and collapsed inline, rather than through an external window.

3. Constraints

There are three factors which constrain our design and development:
1. Encryption of cached local data;
2. The solution’s exclusivity to macOS users;
3. Slack's undocumented API that may be subject to breaking changes.

Because enterprise and team communication can involve confidential information and trade secrets, the solution cannot collect data to be sent remotely to any server other than Slack's official server for analytical purposes. This prevents us from utilizing any analytics frameworks and toolchains, which may make debugging more difficult. Additionally, since the native client caches certain information - such as recent chat transcripts, list of teams, channels and members, they must be encrypted with industrial-level algorithms. We plan to implement 256-bit AES encryption of cached data with SQCipher on top of Apple's CoreData data persistency layer, so that even the end-point device is compromised, the chat transcripts and relevant metadata stay secure.

As indicated in our motivations, we want our solution to feel at home for Mac users, taking advantage of native technologies to improve responsiveness, reduce performance and energy

overhead. Therefore, our solution will be built natively with AppKit and Cocoa, two technologies that are only compatible with computers running macOS. Hence, we do not foresee ourselves porting the solution to other platforms, such as UWP on Windows 10 or WPF on earlier versions of Windows.

Because our solution still needs to be compatible with all existing Slack teams, we need to reverse-engineer Slack's undocumented API, rather than coming up with our own and running our own servers. While Slack provides a limited set of documentation for creating bots that integrate with its service, there is no public documentation on implementing a third party client. We need to use man-in-the-middle proxy tools to sniff the network traffic sent out by Slack's official API in order to create the solution.

While there are no costs, licensing limitations or dependency on hardware, there may be dependencies on Slack's undocumented official API. Because Slack's API is undocumented, Slack makes no promise to keep its API compatible in future revisions to its services. Therefore, technically, Slack's API may change at any time and the change may break our solution. However, the track record throughout the past few years suggests there are rarely breaking changes to its API, and even when there is, the changes are minimal. If the track record is indicative of Slack's engineering team's future behavior, our solution should be functional for the foreseeable future.