Kanishk Kakar
kanishk.kakar@gmail.com
github.com/kanishk98
GMT +05:30
India, fluent in English

Polishing Zulip (Electron)

Making the desktop client an obvious choice for Zulip users

ABSTRACT

With its innovative threading model and robust webapp, Zulip has received a lot of praise from remote teams that use it. While the desktop app is certainly complete in terms of features, it needs some polish and certain standout features to make it an obvious choice for a Zulip user to install.

In this proposal, I suggest the implementation of multiple features to achieve the above goal.

PROPOSED DELIVERABLES

By the end of the summer, I intend to have implemented the following features:

Enterprise deployment

Currently, there is no Zulip-enabled way for admins to deploy the app with custom settings for multiple users in an enterprise setting. After discussions with the community, I've been working with Vipul Sharma to implement a system that allows the admin to write a script for configuring the app as they require via a .json file in the root directory.

My role so far while developing this feature has been to add an **`EnterpriseUtil`** module that configures settings at various places in the app and allows admins to also configure whether keeping a setting admin-only is required or not.

I expect to have completed this feature before the community bonding period begins.

WIP PR #681

Replacing `<webview>` with `BrowserView`

We currently use `<webview>` for rendering all content except the sidebar in the app window. However, the Electron team has warned developers against using `<webview>` because of certain persistent bugs. In a discussion here, it was agreed that we should go with

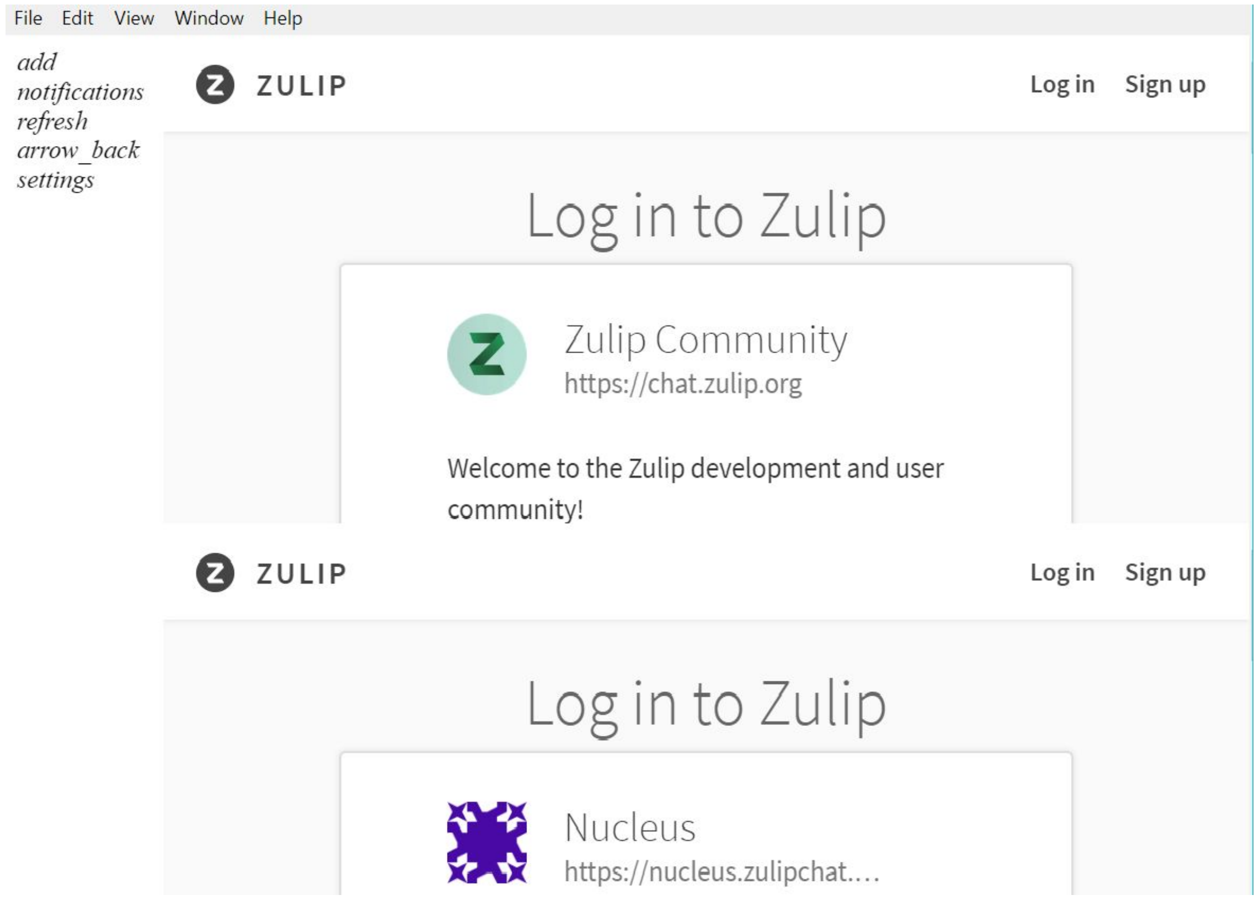
`BrowserView` instead because it has been reported to boost performance and address several issues related to rendering embedding content. While we haven't seen complete failures of the renderer like Slack has, it would be wise to shift before `<webview>` prevents us from updating Electron versions.

Moving to BrowserView is complicated because of the differences between webview and BrowserView. I've listed the challenges below and explained how I'll be tackling them.

Supporting multiple server tabs in one window

While Slack's post about this migration was helpful, it didn't explain how to handle this issue because they use multiple windows for multiple workspaces. We should be able to use the same window for multiple servers because of this PR getting merged into Electron 5.0.0-beta.1.

I'll first work around this by creating a new BrowserView for the sidebar along with one for each server tab. I tried to implement a very basic implementation of this last night, and here's a screenshot for your reference:



This shows two chat screens on the same page as proof-of-concept: we can make n + 1 BrowserViews in the main process, one for the sidebar and n for the number of servers the user has. On switching tabs, we can simply run `win.removeBrowserView(oldView)` and `win.addBrowserView(newView)`.

Listeners and custom CSS
Most listeners and webview-related tasks can be handled by using the `browserView.webContents` object. This will have to be supplemented by passing messages via IPC from the renderer to the main process. For example, when a click is detected on a button in the sidebar, that message is sent to the main process, where we then perform the required actions.

Communication
The BrowserViews will communicate with each other using the main process as the common medium. Actions will be registered by the renderer processes and then acted upon by the main process (meaning that `ipcRenderer.on('event')` calls will mostly be minimised). This'll help keep a clear structure in the app.

Database migration

As we add more features like the one above that require increasingly complex logic for handling the user's app data, we'll see that the current method of using node-json-db is insufficient for our needs. After discussion with Akash Nimare, I suggest moving to lowdb since that gives us access to the entire lodash API and just like node-json-db, is a small local JSON database that fits our use case.

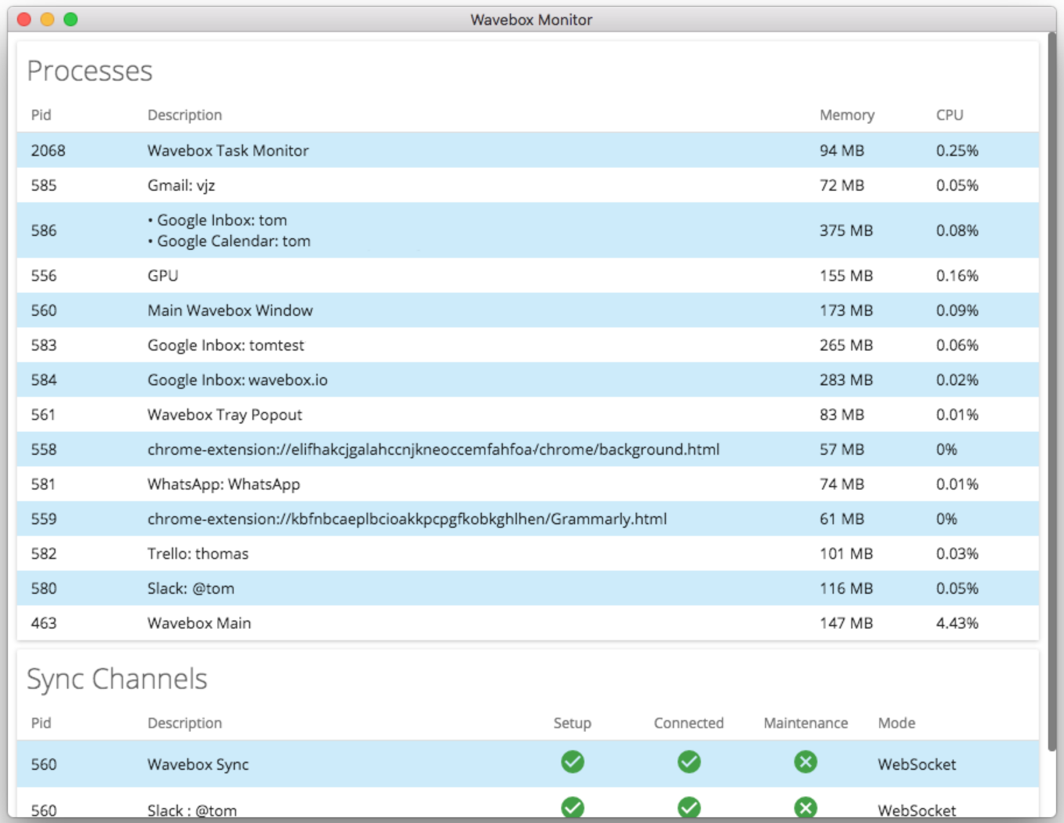
Automated testing

We have currently set up Travis CI only for style checks and linting. This PR has already addressed the lack of unit tests. It'll help maintainers a lot if they can simply check for unit test failures before reviewing a PR. Since the tests have been written using `karma`, implementing automated tests shouldn't be difficult.

Along with setting up a workflow for automated testing using Travis, I shall also be writing tests for the new features I build over the summer and investigating whether we need more tests for the ones that already exist.

Task manager

There have been multiple issues (#695, #213) about high memory/CPU usage. While these are mostly upstream issues in Electron or Chromium, it would be a great benefit to be able to see statistics of the app's usage resources. I suggest we do something similar to Wavebox and add a task manager:


Wavebox Monitor

Processes

Pid	Description	Memory	CPU
2068	Wavebox Task Monitor	94 MB	0.25%
585	Gmail: vjz	72 MB	0.05%
586	• Google Inbox: tom • Google Calendar: tom	375 MB	0.08%
556	GPU	155 MB	0.16%
560	Main Wavebox Window	173 MB	0.09%
583	Google Inbox: tomtest	265 MB	0.06%
584	Google Inbox: wavebox.io	283 MB	0.02%
561	Wavebox Tray Popout	83 MB	0.01%
558	chrome-extension://elifhakcjgalahccnjkneoccemfahfoa/chrome/background.html	57 MB	0%
581	WhatsApp: WhatsApp	74 MB	0.01%
559	chrome-extension://kbfnbcaeplbcioakkpcpgfkobkghlhen/Grammarly.html	61 MB	0%
582	Trello: thomas	101 MB	0.03%
580	Slack: @tom	116 MB	0.05%
463	Wavebox Main	147 MB	4.43%

Sync Channels

Pid	Description	Setup	Connected	Maintenance	Mode
560	Wavebox Sync	✓	✓	✗	WebSocket
560	Slack : @tom	✓	✓	✗	WebSocket

Since Electron depends on Chromium, memory usage stats reported by native OS process monitors like Windows Task Manager suffer from double-counting: they count memory shared among multiple renderer processes multiple times. The Chromium Blog covers this here.
In addition, it's beneficial for our users who file bug reports to be able to quickly look at resource usage without having to open native process monitors. As we head towards a restructuring of the app with `BrowserView`, such information will be incredibly useful in assessing if our work has paid off.

Conversion to TypeScript

Work for supporting TypeScript in the codebase has already been done in PR #610. My plan is not to move the entire codebase to TypeScript in one go once this PR is merged, but start with the features I'm directly working on, and then make a gradual move to TS by changing the files I fix bugs in and so on. I will therefore not be mentioning this in my schedule below; it'll happen throughout the summer.

Issues to solve

There are multiple issues reported to the Zulip repository. I intend to solve some of the most critical as follows:

1. Google login opens in app, not browser
   Some users may be concerned about not logging in to Google via their browser (mostly because of saved login sessions and passwords). I'll try to figure out if OAuth can redirect from the browser to the app somehow to indicate auth state.
2. Inappropriate "Not a Zulip server error" with locked-down hosts
   This issue needs further testing to make sure it hasn't regressed after v1.4.0. (The OP has stopped responding, and I'll check the validation logic and compare it with current tests in the app).
3. Zulip client won't connect after hibernation
   Like the two issues above, this one also requires comprehensive testing. Since it has occurred sporadically across different operating systems and network configurations, this'll be at the highest priority for me during this sprint of the coding period.
4. Sidebar count won't open if you get too far behind
   Initially, we could not retrieve unread message count for a server if the "Welcome...catch up" modal was shown, which meant that the user might think they're up to date on servers with a lot of unread messages. After this discussion, I should be able to fix this easily.
5. Stop checking from error from zulip.ogg if 400 error received
   I intend to collaborate with the contributor to this PR when I'm writing unit tests to help them solve this issue (as per the discussion, the validation logic itself is correct, it just needs to be tested).
6. Online check ignores proxy settings
   Somewhat related to issue 3 above. I'll follow a similar plan here, and discuss with the maintainers if adding unit tests is required.
7. Add a setting to change spellcheck language
   Related with a couple of other high-priority issues (#662 and #646). I'll be tackling this first, going to #662 and then #646 using what I learn from the main solution.

Completing pending PRs

I've sent in quite a few challenging pull requests. These include making server tabs draggable, adding a browser-like loading icon to the sidebar, fixing icon reload issues, a URL entry bar, and so on. Regardless of my application result, I intend to complete these by April 25 so that the accepted GSoC candidate can focus on their proposal instead of dealing with unfinished PRs.

PLANNED SCHEDULE

During the GSoC coding period, I will have no other professional commitments. I expect to work 8-10 hours a day during this time, including weekends. I'll be available for almost the entirety of the summer after my selection. I've noted the exceptions below.

Please note that I'll be spending all time from May 16 to July 21 incrementally working on the important issues listed above. Since they require testing and discussion, I don't think it's a great idea to try fixing them in a single sprint.

Here's how I plan to spend my time working with Zulip:

Time	**Task**
Before GSoC (April 10 - April 25)	● Fix assigned issues ● Improve pending PRs as per recent code reviews ● Finish enterprise deployment
Community bonding period (May 6 - May 27) I'll be unavailable to code from April 26 - May 15 on account of my final exams. However, I will stay in touch with the community at czo.	● Research more about implementation of BrowserView, see if Zulip is extremely dependent on a <webview> API and work with the upstream repositories for workaround ● Migrate database to lowdb
May 27 - June 3	● Write unit tests for database actions ● Design front-end for sidebar as BrowserView
June 4 - June 22	● Implement addition/deletion of servers as BrowserViews ● Integrate switching of servers with the sidebar
June 23 - June 24	● Prepare and submit evaluation 1
June 25 - July 1	● Add Task Manager ● Examine memory usage with increase in the number of BrowserViews using Task Manager ● Submit evaluation 1
July 2 - July 12	● Set up automated testing with Travis CI

	● Testing includes support for tests written so far in #526
July 12 - July 21	● Finish up work on important issues listed
July 22 - August 1	● Investigate analytics tools for logging outlier statistics reported by Task Manager ● Integrate this into the app if possible ● (Before July 26) submit evaluation 2
August 2 - August 19	● Buffer period for unforeseen hiccups ● Prepare final evaluation
August 19 - August 26	● Submit final evaluation

CONTRIBUTIONS TO ZULIP

I'm proud to have been a regular contributor to Zulip, both in terms of code and discussions. My ***merged PRs*** are:

- Default to starting app on login
- Switch to next server on Ctrl + Tab
- Fix context menu indexing
- Add context menu to letter icon
- Disable beta updates if auto updates disabled

Nearly merged/approved PRs:

- Make org tabs draggable in sidebar
- Add option to find accounts by email
- Add option to hide menu bar to View menu

Complete but open PRs, waiting for further review/discussion:

- Add zulip:// URI scheme for navigating within app
- Revert to fallback icon only if needed
- Trim domain to first word in server URL
- Add loading indicator to sidebar
- Add URL copy/paster to left sidebar
- Fix flashing icons of servers in sidebar

WIP PRs:

- Add custom configurations for enterprise (collaborative effort with Akash, Priyank, and Vipul)
- Mute organization badge and server count (depends on a web app-enabled setting)

I've also opened a few ***issues*** to report bugs or enhance functionality. They are:

- Can't run Travis checks locally [Windows]
- Cursor does not focus in Settings box after soft reload (Ctrl/Cmd + R)
- Individual screens for showing network error

WORK EXPERIENCE

After my first year of college, I worked with Prof. S. Mathur as a data analytics intern and used statistics to offer insights into certain case studies. After my second year, I worked with Reap Benefit Foundation to work on an Android app that would detect potholes on Indian roads.

As an intern at RBF, I re-structured the app to eliminate the need for storing and uploading large log files containing phone sensor data by standardising readings across different devices and processing the files offline. On average, this resulted in a 100x reduction of storage space taken by the app.
Initially, the app was only intended for usage in cars. I wrote a new autoencoder algorithm that could account for outliers and classify potholes on any vehicle type, and also moved that offline by using TensorFlow for Android.

OPEN-SOURCE EXPERIENCE

Before contributing to Zulip, I contributed to GitHub Desktop (also an Electron app). Here's a list of my contributions there:

- [Issue] Auth error prompt when cloning non-existent repo
- [PR] Add compare-to-branch to MenuIDs
- [PR] Fixed auth error prompt when cloning non-existent repo

I have also created a downloader library for Android that's targeted at emerging markets in discussion with the Mozilla community.

Road Quality Audit (the pothole detector app I worked on during my second internship) is an open-source product.

I've also worked on several personal projects, all of which are open-source:

- Nucleus
  Social networking app exclusively for college students. Allows them to chat anonymously with new people, find anyone in college without a phone number, and post questions anonymously and get anonymous votes on them. Built using React Native and GraphQL.

- Transaction scheduler
  Demonstration of a scheduling algorithm for a multi-user database system. Locking in databases is a complex problem, and I attempted to implement and improve upon a research paper on contention-aware lock scheduling for transactional databases. Uses the batched Largest Dependency Set First algorithm. Written in Python.

- Kanishk's GraphQL to-do app
  Built as a demonstration of the capabilities of Hasura's GraphQL engine. The app was reviewed and praised by Tanmai Gopal, the CEO of Hasura. You may check it out in action or read a tutorial explaining how I built the app. I used React.js for the web app.

- Dell Inventory Manager
  A Dell hackathon project to decide what to do with aging inventory. I worked on the back-end and front-end infrastructure of the app, while the machine learning model and data preprocessing was handled by other members of my team. Built using React.js, Express, Flask, and Keras.

- SKCH-CBM Marker
  Android solution for SKCH School, Bangalore to mark exact locations of their bus stops so they can distribute the collected locations to parents and students.

WHY ZULIP

I first learned about Zulip when I came to college and saw other students putting Zulip stickers on their laptops. I checked it out online, learnt about the threading model and the inspiration behind it, but didn't really understand how it would help me or how I could help the Zulip community. A couple of years, multiple projects, and lots of collaborative work later, I came to realise the value that Zulip brings to the table as an enabler of communication. I'm someone who loves to build systems that help people interact (Nucleus is an example) and I think that contributing to the desktop app, something I use daily now for communication, would be a great use of my time and my way of giving back to the community that has taught me the most about Electron development.

WHY ME

My entry into development was as an Android developer. I started pretty late; in the middle of my second year in college. Since then, I have made it a point to learn, build, and iterate over my projects quickly; in a year, I have come from not knowing what a REST API was to working with GraphQL companies and creating tutorials about their technology.
I have significant experience in developing complex apps - both Nucleus and the Pothole Detector app had a lot of moving parts and required me to carefully think about app architecture. I'm proud of the effort I can put in and am excited by challenging work environments. With a possible restructuring of the desktop app around the corner, I believe these qualities make me a good fit for Zulip.

POST GSoC

I'm confident that even after my work with Zulip over the summer, there will be a lot of room for improvement in the desktop app. I plan to branch out a little into the Zulip ecosystem after the summer because I'm also really interested in how the back-end of a messaging platform as massive as this works at scale.