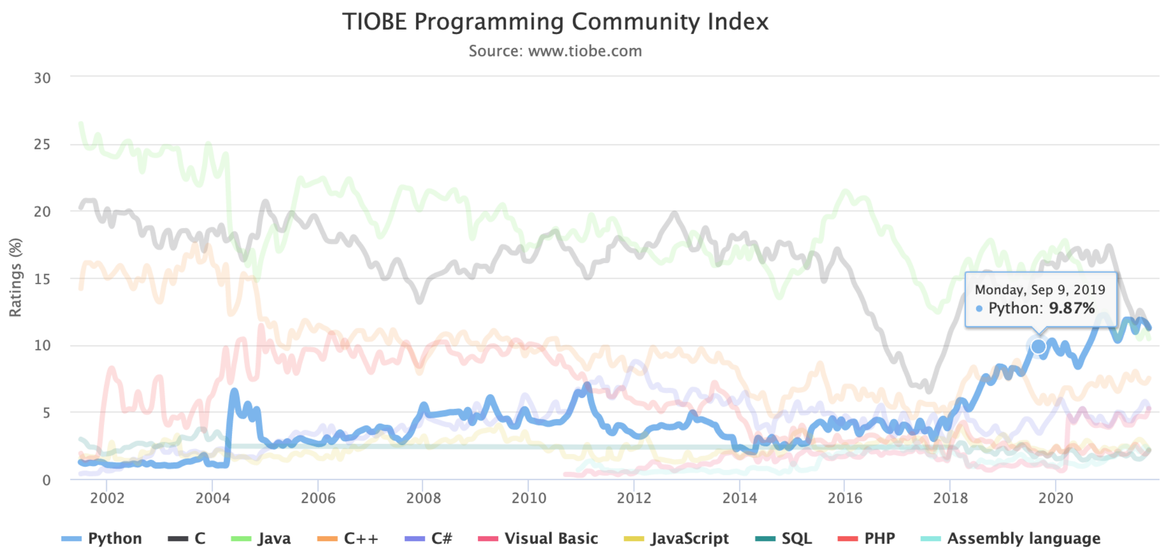
Textual

Python recently overtook C and Java as the most popular programming language and is often described as “[at least] the second best language for everything”. Far from derogatory, this is a testament to the language's flexibility: if you know Python you can apply it to virtually anything.



Consequently there is a lot of Python code being written, not all of which by developers with Python as their primary skill set.

Much of that code is inaccessible to the non-technical user until the developer implements a *user interface* (UI). Herein I propose that the standard methods for building UIs in Python and other languages are prohibitively expensive and time consuming, to the extent that many UIs are simply not being created.

The lack of a UI means that critical tasks may be performed manually. Additionally, without a UI there is a lack of visibility on many systems that an organisation may depend on.

I propose that the solution is to build on the *terminal* as a means of delivery for these user interfaces. Supported by all operating systems, terminal software is in many ways more ubiquitous than the web browser and yet unfamiliar to most non-technical users.

I will present a solution using *Textual*, a modern terminal framework powered by Python that seamlessly integrates terminal applications with the desktop, bridging the gap between developer and end-user.

Textual will be the fastest and easiest way to build and deploy Python code.

Problem 1

“User Interfaces take a long time to do well.”

Broadly speaking there are 2 ways to build an UI: desktop application or web application. Both require a significant investment in time and effort.

Python has the cross-platform desktop UI library “tkinter” library built in. Apps built with this library don’t look native and aren’t used a great deal outside of education. The most popular alternatives are wxPython and QT for Python. Both these libraries can produce slick native (or native looking) applications. Downsides are that they are heavy-weight and require a large C binary component. They are also large and complex APIs with a steep learning curve.

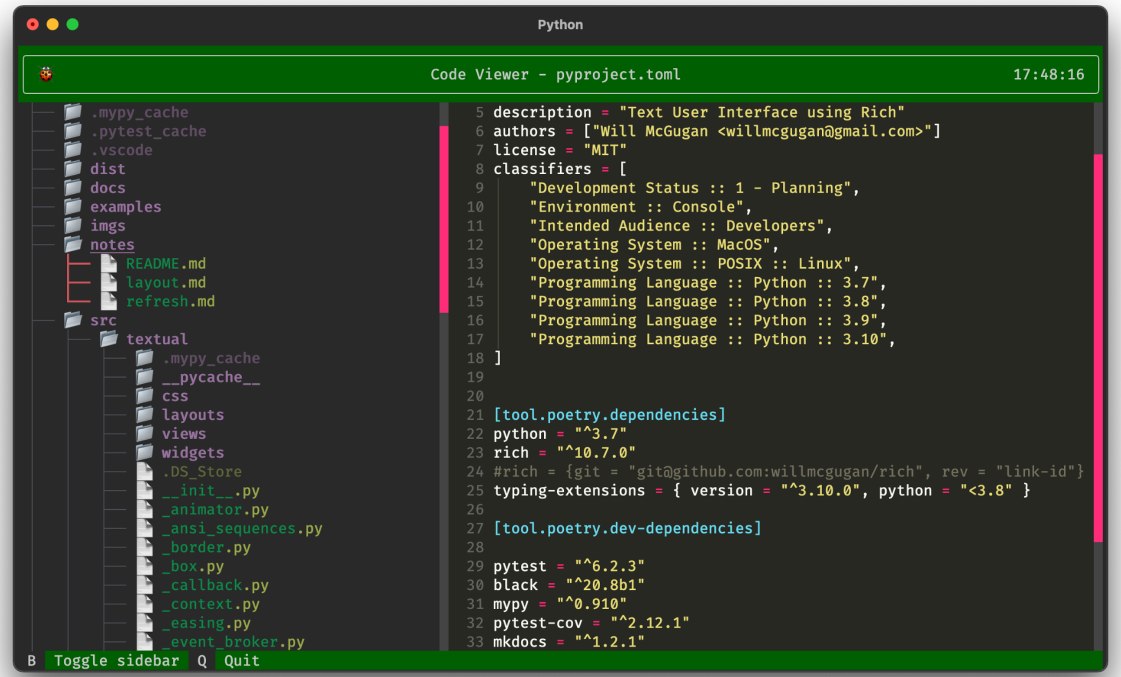
In general, Desktop applications require a significant amount of experience to do well, and are complicated by the requirement to make them cross-platform. Long term maintenance is typically required.

A web application is a more modern solution but requires specialised and cross-disciplinary skills. Even in-house tools require the skillsets of a back-end developer(s), front-end developer(s), and designer(s).

Python is a popular tool for web development and offers such libraries as Django, and Flask, amongst others. They are very helpful for the component of a web-app that runs on the server, but still require the developer to also be proficient in a minimum of HTML and CSS, typically also requiring knowledge of Javascript. Additionally, a designer would be required for all but the most basic web applications.

Solution 1 - Textual

Textual is a Python framework for building applications within the terminal. Such applications are built on a grid of characters but present many familiar UI elements such as scrollbars, tree controls, panels, and menus. See below for an example of a Textual application:



```
.mypy_cache
.pytest_cache
.vscode
dist
docs
examples
imgs
notes
  README.md
  layout.md
  refresh.md
src
  textual
    .mypy_cache
    __pycache__
    css
    layouts
    views
    widgets
    .DS_Store
    __init__.py
    _animator.py
    _ansi_sequences.py
    _border.py
    _box.py
    _callback.py
    _context.py
    _easing.py
    _event_broker.py
```

```
description = "Text User Interface using Rich"
authors = ["Will McGugan <willmcgugan@gmail.com>"]
license = "MIT"
classifiers = [
    "Development Status :: 1 - Planning",
    "Environment :: Console",
    "Intended Audience :: Developers",
    "Operating System :: MacOS",
    "Operating System :: POSIX :: Linux",
    "Programming Language :: Python :: 3.7",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
]


[tool.poetry.dependencies]
python = "^3.7"
rich = "^10.7.0"
#rich = {git = "git@github.com:willmcgugan/rich", rev = "link-id"}
typing-extensions = { version = "^3.10.0", python = "<3.8" }

[tool.poetry.dev-dependencies]

pytest = "^6.2.3"
black = "^20.8b1"
mypy = "^0.910"
pytest-cov = "^2.12.1"
mkdocs = "^1.2.1"
```

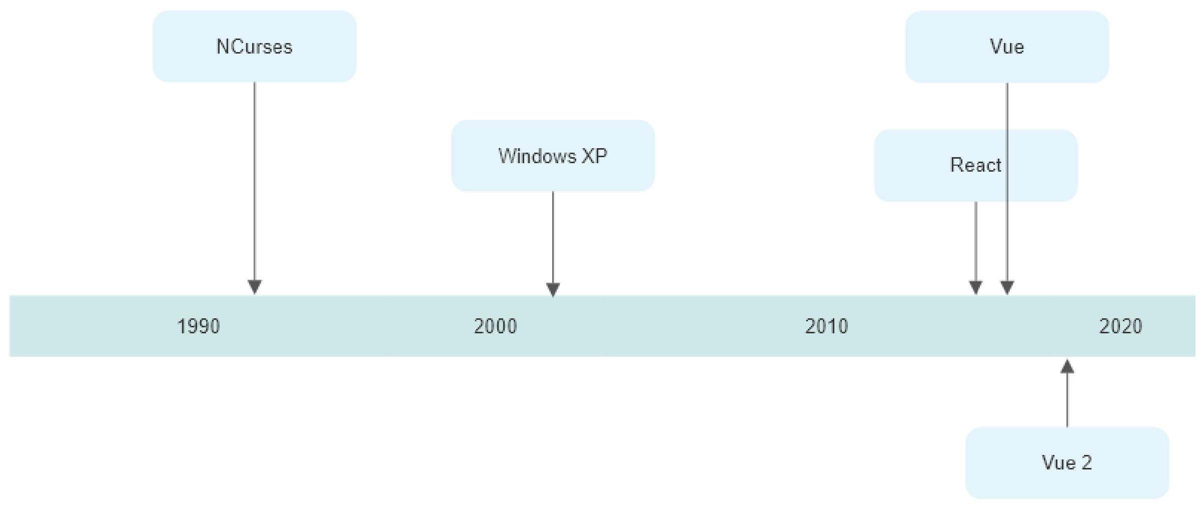
Building a Textual application can be done exclusively with Python, and requires little specialized knowledge. Textual apps are also cross platform, and will run identically on Windows, Linux and MacOS.

Textual borrows a few concepts from the web world for rapid development. In particular, *reactivity*. Found in the Vue and React web frameworks, reactivity allows the developer to make changes to the data and have the interface update accordingly. This feature permits simple expressive code without learning a large API.

Components of a Textual application (widgets) work in this way, and can be distributed independently. It is expected that Textual will offer a large number of components, both official and third-party.

There are other frameworks and toolkits for building apps within the terminal. The most notable of which, and an industry standard, is Curses, a library designed in the early 90s.

While some newer projects attempt to improve upon Curses, they tend to follow the familiar design patterns set out decades ago. In practical terms there has been very little innovation in this area to date. Consequently developing terminal applications is considered somewhat antiquated and avoided by most developers.



Since Curses was released, building user interfaces has been the subject of intense development and research, particularly in the last 5 to 10 years in the web world. Very little of that progress has made it to terminal based applications.

Paradoxically, although terminal toolkits haven't kept pace with modern developments, the terminal software that powers them is distinctly up-to-date. The terminal software that comes as default on many systems is powered by the same technology as video games, supporting 16.7 million colors and a fast refresh rate. And yet it is rare to see any project that takes advantage of these capabilities.

While still under active development, Textual already makes better use of the capabilities of the hardware it runs on, providing a user experience closer to that of the web than typical terminal apps.

Textual combines recent developments in Python with modern innovations from web development to create an API which is comfortable and familiar for any developer to pick up and yet can produce production-ready apps with minimal effort.

Problem 2

“Distribution is difficult.”

Once an application has been built, it needs to reach its intended audience.

Distributing a desktop application is fraught with problems in compatibility, security, and physically getting the installers to those that need it. Many organisations will restrict users from installing software due to these issues.

Web applications have the edge in distribution in that they are available instantly and have less security concerns, but require investment in infrastructure and maintenance. Web applications must also carefully manage users and permissions which is typically an ongoing pain-point.

Solution 2 - Textual Cloud Service

While Textual can run on any device with Python installed, there is no integrated way of distributing such applications.

This can be solved with a cloud service which transforms the terminal application into a web application. With a single switch, the developer can retrieve a public URL where their app runs within a web-page.

Additionally, web-facing Textual apps will be built as “Progressive Web Applications”, a technology supported by all the major browser vendors allowing a web application to be installed on the desktop or home screen. When installed in this way, a Textual application can look and feel very much like a native application, even on phone and tablet.

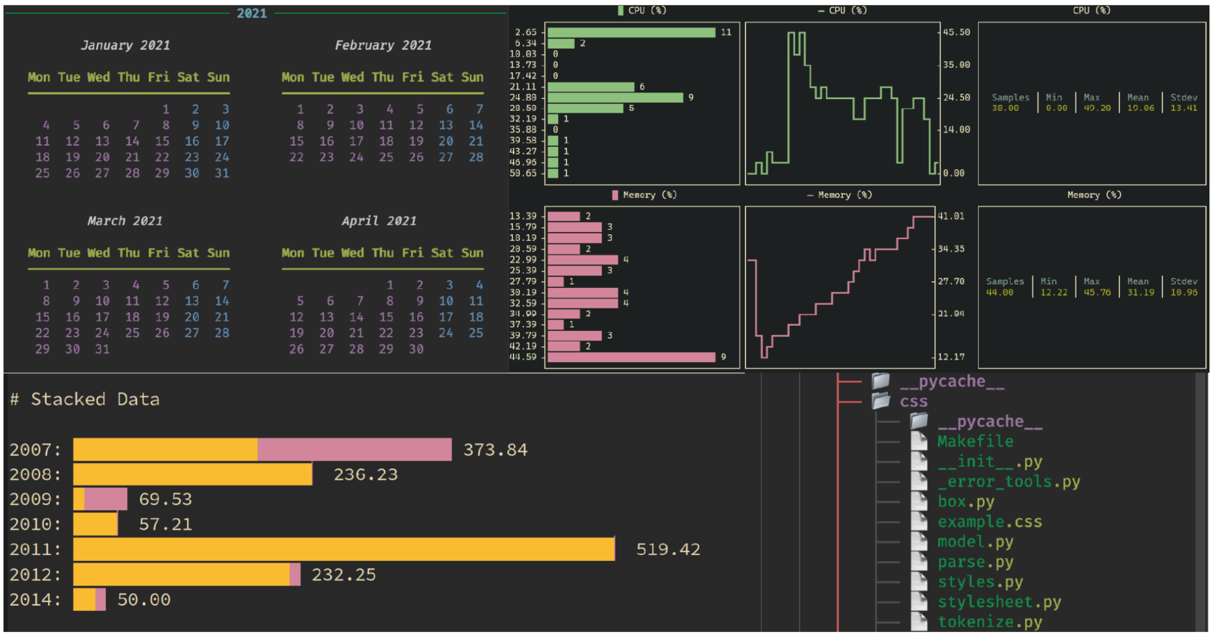
In addition to the base tier service of exposing applications on the web, the Textual Cloud Service can offer a number of value-add services:

- **Authentication** - The cloud service can provide authentication services for applications using industry standards, such as OAuth2 and two factor authentication.
- **Team support** - Support for more finely grained permissions for larger organisations.
- **End to end encryption** - Additional security.
- **Management interface** - An advanced interface capable of managing multiple applications in one.

Anatomy of a Terminal App

A terminal presents a grid of characters: letters, numbers, digits, emoji, etc., which may have a different foreground and background color, and a small selection of styles such as bold, italic and underline.

Graphics in terminal apps are generally limited to symbols for drawing lines and boxes, but can none-the-less be used to render plots and diagrams (see below).



```
                                   2021

        January 2021                         February 2021

 Mon Tue Wed Thu Fri Sat Sun          Mon Tue Wed Thu Fri Sat Sun

                   1   2   3            1   2   3   4   5   6   7
   4   5   6   7   8   9  10            8   9  10  11  12  13  14
  11  12  13  14  15  16  17           15  16  17  18  19  20  21
  18  19  20  21  22  23  24           22  23  24  25  26  27  28
  25  26  27  28  29  30  31

         March 2021                           April 2021

 Mon Tue Wed Thu Fri Sat Sun          Mon Tue Wed Thu Fri Sat Sun

   1   2   3   4   5   6   7                        1   2   3   4
   8   9  10  11  12  13  14            5   6   7   8   9  10  11
  15  16  17  18  19  20  21           12  13  14  15  16  17  18
  22  23  24  25  26  27  28           19  20  21  22  23  24  25
  29  30  31                           26  27  28  29  30
```

```
  CPU (%)                 CPU (%)                      CPU (%)
 2.65  11                        45.50
 6.34  2                         35.00      Samples | Min  | Max   | Mean  | Stdev
10.03  0                         24.50      38.00   | 0.00 | 49.20 | 19.06 | 13.41
13.73  0                         14.00
17.42  0                          0.00
21.11  6
24.80  9
28.50  5
32.19  1
35.88  0
39.58  1
43.27  1
46.96  1
50.65  1

  Memory (%)              Memory (%)                   Memory (%)
13.39  2                         41.01
15.79  3                         34.35      Samples | Min   | Max   | Mean  | Stdev
18.19  3                         27.70      44.00   | 12.22 | 45.76 | 31.19 | 10.96
20.59  2                         21.04
22.99  4                         12.17
25.39  3
27.79  1
30.19  4
32.59  4
34.99  2
37.39  1
39.79  3
42.19  2
44.59  9
```

```
# Stacked Data

2007:  373.84
2008:  236.23
2009:  69.53
2010:  57.21
2011:  519.42
2012:  232.25
2014:  50.00
```

```
├── __pycache__
├── css
│   ├── __pycache__
│   ├── Makefile
│   ├── __init__.py
│   ├── _error_tools.py
│   ├── box.py
│   ├── example.css
│   ├── model.py
│   ├── parse.py
│   ├── styles.py
│   ├── stylesheet.py
│   ├── tokenize.py
```

Due to their low-graphics nature, terminal apps are unlikely to be suitable for applications which require branding and richer visuals, but are wholly suitable for information dense interfaces or applications where visuals aren't the top priority.

Rather than a limitation, the reduced graphical capabilities of terminal apps can be seen as a benefit. The developer of a Textual application has far less to consider when it comes to styling. Additionally, many developers express a preference for terminal applications and would appreciate the same aesthetic in the web-browser and desktop.

Terminal apps have long been capable of supporting mouse input, some even support touch gestures. Yet few applications make any particular use of the mouse. The Textual framework has good support for the mouse, with scrollbars, links, and buttons. Ultimately such applications will function similar enough to web or desktop platforms to be familiar to any user.

Low Code / No Code

It is practical for a Textual application to be built not with Python code, but with a configuration file. This more declarative approach can assemble pre-existing components in novel ways while still providing sophisticated functionality. Such a declarative approach will appeal to those wishing to build a UI who perhaps don't have Python in their primary skillset, or at all.

Textual Enhancements

Bringing terminal applications to the web presents a number of potential enhancements to the terminal experience.

Graphics

Graphics aren’t well supported in terminals. While there are non-standard extensions, most terminals are only capable of displaying low-resolution images. A web-facing terminal application has the opportunity to replace such images with higher resolution versions.

Accessibility

A very reasonable criticism of terminal applications is that they are not accessible to vision impared users. A web-facing terminal application can remedy this by offering a selection of color themes and text sizes. Additionally, a textual application can also provide information which the web-facing application can use to enable screen-reader support.

Downloads and Uploads

Given that textual apps have access to a local filesystem (unlike a web application), a web-facing application should have the capability of both sending a file to the server and receiving a file from it. The developer will be able to decide what files and directories are exposed.

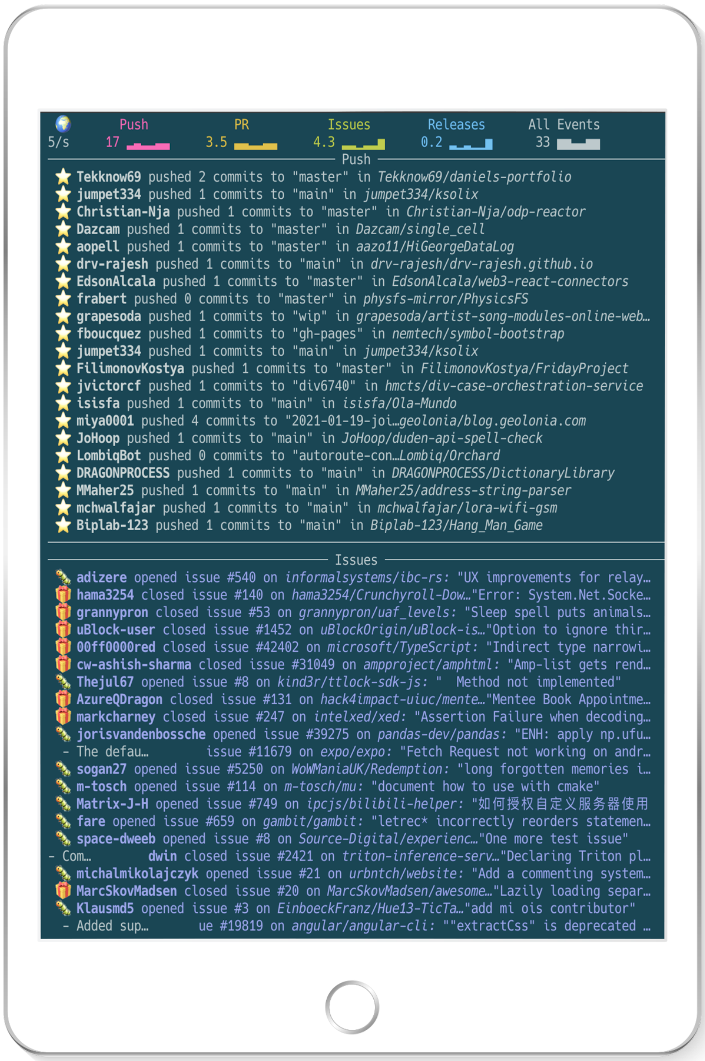
Smoother Scrolling

Due to its grid based nature, scrolling on terminals is not as smooth as more modern alternatives. Web-facing Textual apps can improve upon this and implement pixel perfect scrolling.

Textual Mockup

Web facing Textual applications inherit a retro-aesthetic from terminals, but don't require any particular knowledge of terminals to operate.

The mockup below shows how a text based terminal based application may look running on a tablet. It will be possible to integrate touch controls and gestures with apps, which wouldn't typically work in standard terminal software.



```
        Push          PR           Issues        Releases      All Events
5/s     17            3.5          4.3           0.2           33
──────────────────────────────── Push ────────────────────────────────
 Tekknow69 pushed 2 commits to "master" in Tekknow69/daniels-portfolio
 jumpet334 pushed 1 commits to "main" in jumpet334/ksolix
 Christian-Nja pushed 1 commits to "master" in Christian-Nja/odp-reactor
 Dazcam pushed 1 commits to "master" in Dazcam/single_cell
 aopell pushed 1 commits to "master" in aazo11/HiGeorgeDataLog
 drv-rajesh pushed 1 commits to "main" in drv-rajesh/drv-rajesh.github.io
 EdsonAlcala pushed 1 commits to "master" in EdsonAlcala/web3-react-connectors
 frabert pushed 0 commits to "master" in physfs-mirror/PhysicsFS
 grapesoda pushed 1 commits to "wip" in grapesoda/artist-song-modules-online-web…
 fboucquez pushed 1 commits to "gh-pages" in nemtech/symbol-bootstrap
 jumpet334 pushed 1 commits to "main" in jumpet334/ksolix
 FilimonovKostya pushed 1 commits to "master" in FilimonovKostya/FridayProject
 jvictorcf pushed 1 commits to "div6740" in hmcts/div-case-orchestration-service
 isisfa pushed 1 commits to "main" in isisfa/Ola-Mundo
 miya0001 pushed 4 commits to "2021-01-19-joi…geolonia/blog.geolonia.com
 JoHoop pushed 1 commits to "main" in JoHoop/duden-api-spell-check
 LombiqBot pushed 0 commits to "autoroute-con…Lombiq/Orchard
 DRAGONPROCESS pushed 1 commits to "main" in DRAGONPROCESS/DictionaryLibrary
 MMaher25 pushed 1 commits to "main" in MMaher25/address-string-parser
 mchwalfajar pushed 1 commits to "main" in mchwalfajar/lora-wifi-gsm
 Biplab-123 pushed 1 commits to "main" in Biplab-123/Hang_Man_Game
──────────────────────────────────────────────────────────────────────

─────────────────────────────── Issues ───────────────────────────────
 adizere opened issue #540 on informalsystems/ibc-rs: "UX improvements for relay…
 hama3254 closed issue #140 on hama3254/Crunchyroll-Dow…"Error: System.Net.Socke…
 grannypron closed issue #53 on grannypron/uaf_levels: "Sleep spell puts animals…
 uBlock-user closed issue #1452 on uBlockOrigin/uBlock-is…"Option to ignore thir…
 00ff0000red closed issue #42402 on microsoft/TypeScript: "Indirect type narrowi…
 cw-ashish-sharma closed issue #31049 on ampproject/amphtml: "Amp-list gets rend…
 Thejul67 opened issue #8 on kind3r/ttlock-sdk-js: "  Method not implemented"
 AzureQDragon closed issue #131 on hack4impact-uiuc/mente…"Mentee Book Appointme…
 markcharney closed issue #247 on intelxed/xed: "Assertion Failure when decoding…
 jorisvandenbossche opened issue #39275 on pandas-dev/pandas: "ENH: apply np.ufu…
 - The defau…        issue #11679 on expo/expo: "Fetch Request not working on andr…
 sogan27 opened issue #5250 on WoWManiaUK/Redemption: "long forgotten memories i…
 m-tosch opened issue #114 on m-tosch/mu: "document how to use with cmake"
 Matrix-J-H opened issue #749 on ipcjs/bilibili-helper: "如何授权自定义服务器使用
 fare opened issue #659 on gambit/gambit: "letrec* incorrectly reorders statemen…
 space-dweeb opened issue #8 on Source-Digital/experienc…"One more test issue"
- Com…        dwin closed issue #2421 on triton-inference-serv…"Declaring Triton pl…
 michalmikolajczyk opened issue #21 on urbntch/website: "Add a commenting system…
 MarcSkovMadsen closed issue #20 on MarcSkovMadsen/awesome…"Lazily loading separ…
 Klausmd5 opened issue #3 on EinboeckFranz/Hue13-TicTa…"add mi ois contributor"
 - Added sup…        ue #19819 on angular/angular-cli: ""extractCss" is deprecated …
```

Textual Applications

There are a number of areas where Textual apps (with cloud service) present a significant advantage over current technologies.

Internal Tooling

Textual apps can be an entry point into an organisation’s infrastructure, providing an interface to existing processes and tools. The application can respond to actions within the UI and to other triggers, such as watching a file or API. And since the application has access to the filesystem it can make changes to configuration.

Operational Analytics

The realtime nature of Textual apps makes them particularly useful for monitoring the running of various processes. Log files can be tailed and analyzed in real time, presenting an aggregated interface with text, charts, and plots.

Device Configuration

Since Textual apps don’t need to be registered, and there is no limit to the number created, they can be installed on devices such as routers and single board computers. This would allow for such devices to be remotely monitored and configured

Jupyter Extensions

The developers of Jupyter have expressed an interest in embedding Textual apps within Jupyter notebooks (a tool popularised by data scientists and machine learning engineers). Building Jupyter extensions with Python and Textual would allow more Jupyter users to contribute to the ecosystem. Currently web-development skills are required to build extensions, which are far from universal.

Textual apps can also expose data which is not readily available via an API, such as files on a server on the intranet--which can be the case for laboratories running specialized equipment.

Embedded Applications

Textual applications can be embedded within web pages, providing additional services to back-end tools. For instance displaying server logs, editing configuration, and restarting processes.