Building a Web Frontend for Federated Communication with Brython

Jérôme Poisson (Goffi)

2024-02-04, FOSDEM'24

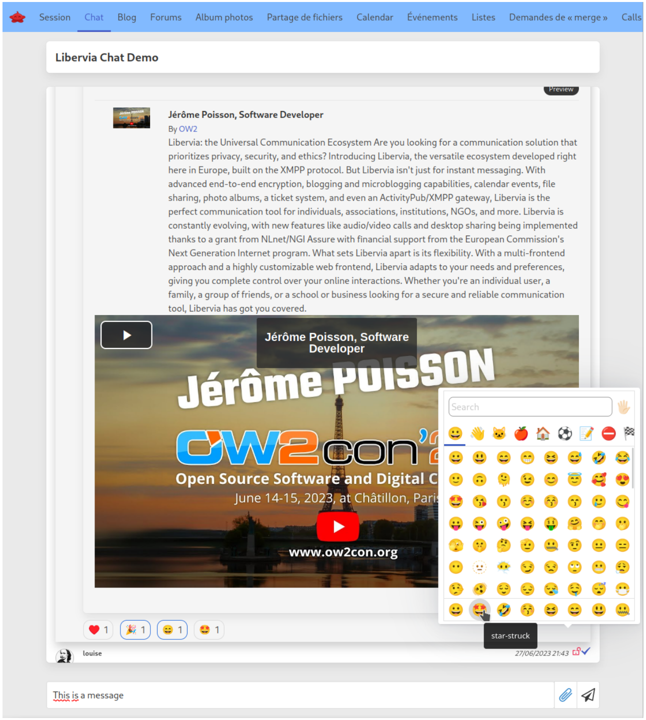
Libervia

- Universal communication ecosystem
- Built on XMPP
- Features: chat, blogging, A/V calls, calendar events, forums, etc.
- End-to-end encryption (planned for web frontend)
- Multi-frontends
- Compatible with ActivityPub

Why Python in the Browser?

- No context switching
- Quick and easy development
- High code reusability
- Stable Ecosystem

Libervia Web Screenshot



Libervia Chat Demo

Jérôme Poisson, Software Developer

By OW2

Libervia: the Universal Communication Ecosystem Are you looking for a communication solution that prioritizes privacy, security, and ethics? Introducing Libervia, the versatile ecosystem developed right here in Europe, built on the XMPP protocol. But Libervia isn't just for instant messaging. With advanced end-to-end encryption, blogging and microblogging capabilities, calendar events, file sharing, photo albums, a ticket system, and even an ActivityPub/XMPP gateway, Libervia is the perfect communication tool for individuals, associations, institutions, NGOs, and more. Libervia is constantly evolving, with new features like audio/video calls and desktop sharing being implemented thanks to a grant from NLnet/NGI Assure with financial support from the European Commission's Next Generation Internet program. What sets Libervia apart is its flexibility. With a multi-frontend approach and a highly customizable web frontend, Libervia adapts to your needs and preferences, giving you complete control over your online interactions. Whether you're an individual user, a family, a group of friends, or a school or business looking for a secure and reliable communication tool, Libervia has got you covered.

Jérôme Poisson, Software Developer
Jérôme POISSON
OW2con'2
Open Source Software and Digital C
June 14-15, 2023, at Châtillon, Paris
www.ow2con.org
Search
star-struck

❤️ 1 🎉 1 😄 1 🤩 1

louise

27/06/2023 21:43

This is a message

Alternatives

Pyjamas/PyJS

- Used until Libervia 0.6
- port of GWT to Python
- Python-to-JavaScript transpiler
- Heavy
- Similar to desktop development
- Supports Python 2 only
- Project is no longer active

Transcrypt

- Python-to-JavaScript transpiler
- Lightweight
- Not fully Python compatible
- No port of standard libraries (use instead JS modules)

Pyodide

- CPython ported to WebAssembly
- Heavy
- Fully compatible with CPython
- Supports numerous packages
- Notably great for scientific packages

PyScript

- Introduced after Brython was chosen
- Uses WebAssembly, relies on Pyodide or MicroPython
- Offers web integration
- Choice between full Python compatibility (Pyodide, heavier) or lighter version (MicroPython, less compatible)
- Seems easier to use then Pyodide

Other

- Skulpt
  - similar to Brython
  - not yet Python 3
- PyPy.js
  - PyPy ported to WebAssembly (emscripten)
  - unmaintained

Brython

- Transpiles Python to JavaScript directly in the browser
- Caches the transpiled code
- Includes a compatibility layer
- Real Python, strong compatibility
- Up-to-date with Python releases
- Most standard libraries are available
- Supports pure Python packages
- Enables dynamic JS transpilation
- Allows calling JavaScript's eval
- Facilitates direct use of JS code in Python and vice versa
- Welcoming, responsive and supportive community

It's Python, for real!

```
import antigravity
```

How the Web Frontend Works

Overview

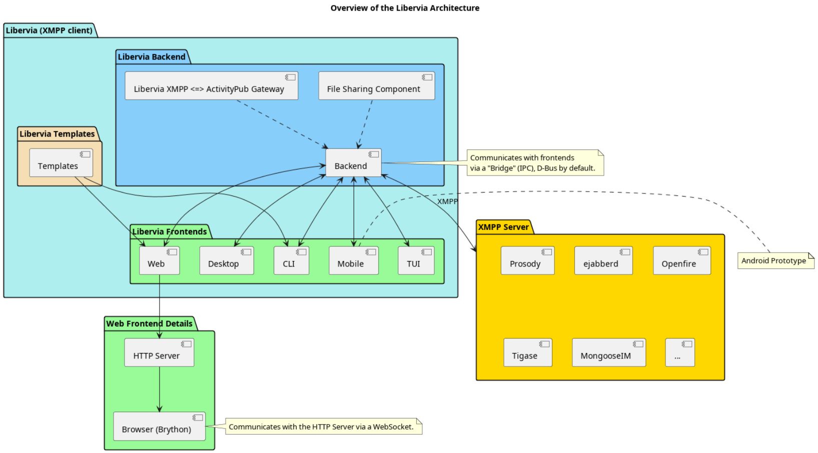


Figure 2: Libervia Architecture Overview

Goals of the Web Frontend

- Progressive enhancement
  - Mostly functional in static environments
  - Utilizes JavaScript if available
- Ease of development and maintainability
- Code reuse

Templating

- Jinja2 (Python)
- Nunjucks (JS)
- Both are mostly compatible
- Implements missing filters/directives in Brython
- Ensures compatibility with backend templates
- Supports easy theming
- Templates can be utilized by the CLI frontend:
  - static websites
  - data formatting

Libervia “Pages”

- Each directory corresponds to an HTML path
- Page code in `page_meta.py` includes:
  - Name
  - Access policy
  - Template
  - Methods for:
    - URL parsing
    - Data preparation
    - Handling POST requests
    - And other things
- Browser-specific code in `_browser` directory
- Final files hierarchy automatically generated

Code Example

Minimal Page

```python
from libervia.web.server.constants import Const as C

name = "calls"
access = C.PAGES_ACCESS_PROFILE
template = "call/call.html"
```

Browser Code

```python
import json

from bridge import AsyncBridge as Bridge, BridgeException
from browser import document, aio
import dialog

bridge = Bridge()

# on_delete see next slide

for elt in document.select('.action_delete'):
    elt.bind("click", lambda evt: aio.run(on_delete(evt)))
```

```
async def on_delete(evt):
    evt.stopPropagation()
    evt.preventDefault()
    target = evt.currentTarget
    item_elt = target.closest('.item')
    item_elt.classList.add("selected_for_deletion")
    item = json.loads(item_elt.dataset.item)

    confirmed = await dialog.Confirm(
        f"List {item['name']!r} will be deleted, are you sure?",
        ok_label="delete",
    ).ashow()

    if not confirmed:
        item_elt.classList.remove("selected_for_deletion")
        return

    try:
        await bridge.interest_retract("", item['id'])
    except BridgeException as e:
        dialog.notification.show(
            f"Can't remove list {item['name']!r} from personal interests: {e}",
            "error"
        )
    else:
        print(f"{item['name']!r} removed successfuly from list of interests")
        # deletion effect
```

Demo Video

Debugging

Debugging

- Real Python tracebacks
- Sometimes JS exceptions
- We can use breakpoint and pdb!
- `interpreter.Inspector`

Performance Considerations

- Same order of magnitude as CPython (according to documentation)
- Slower than JavaScript due to compilation + compatibility layer
  - compiled JS is cached
- `brython_stdlib.js` is big (∼4.5 Mb) but can be reduced or fully removed
- Loading time (first time, then cache)
- from my experience and use case: absolutely acceptable, and I've yet to optimize

Roadmap

- **Enhance Brython Integration**:
  - Blog to social network UX evolution.
- **Backend Code Reusability**:
  - Foundation for e2ee.
- **Implement Full e2ee On Demand**:
  - Secure user communication.
- **Experiment with Innovative Use of Python in Browser**:
  - The possibilities are vast: education, science, automation, etc.

Conclusion

Conclusion

Brython stands as a robust solution for integrating Python into web development. It bridges the gap between backend and frontend, promoting code reuse and efficiency.

Thank You!

- Brython: https://brython.info
- Libervia: https://libervia.org
- blog: https://www.goffi.org
- XMPP room: libervia@chat.jabberfr.org
- ActivityPub: @goffi@mastodon.social

Thank you for attending, happy web hacking with Brython!