Tremola/TinySSB Software-Architektur Dec 2024

christian.tschudin@unibas.ch

`https://github.com/ssbc/tinySSB`

- currently only for Android (BLE, Wifi und WebSocket) and select embedded devices (ESP32-based)

- additionallly: development version for Chrome browser with simulated replication (from browser tab to browser tab)

- partly done: JavaScript version of the lower layers using the *library from SocketSupply - unfinished work*
  —> potentially enables to run tinySSB on MacOS, iOS, Windows and Android

tinySSB - SW Architecture (2024)

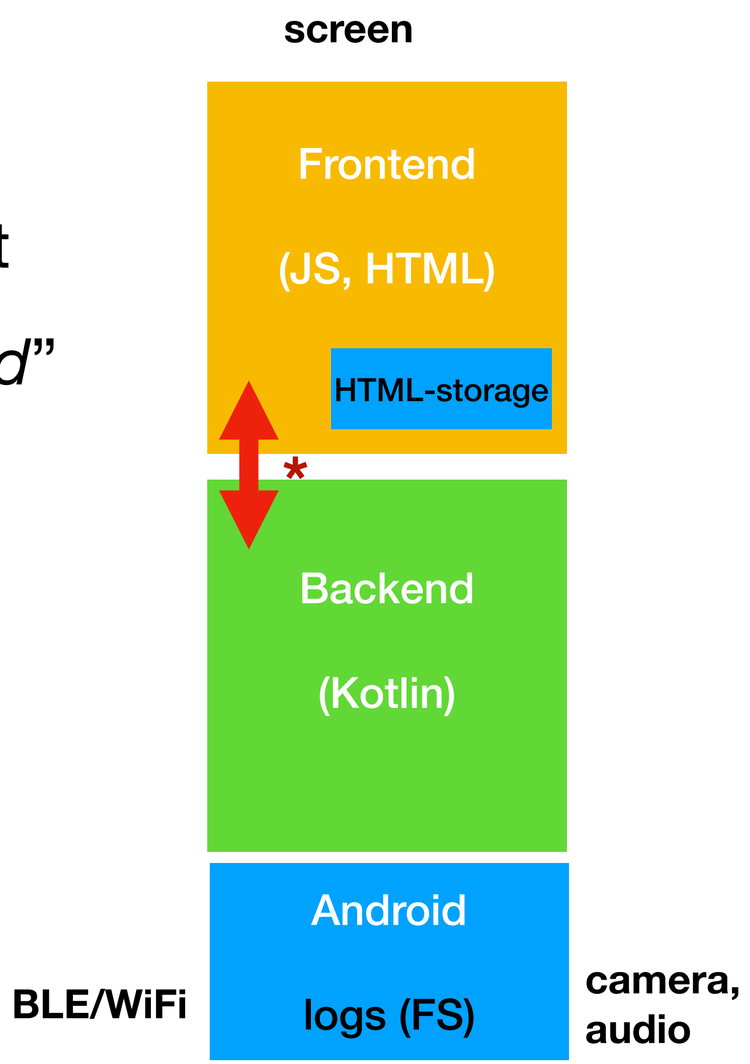
Two components

Frontend:

- GUI and application logic, in HTML and JavaScript
- receives, triggers the writing of log entries, “*cooked*”

Backend:

- responsible for wire format (binary encoding)
- digitally signs new log entries
- storage of “*raw*” log entries
- handles replication protocol



tinySSB SW Architecture (contd)

Event driven software. Execution paths:

(0) “ready” signal from WebKit frontend, once it’s initialized

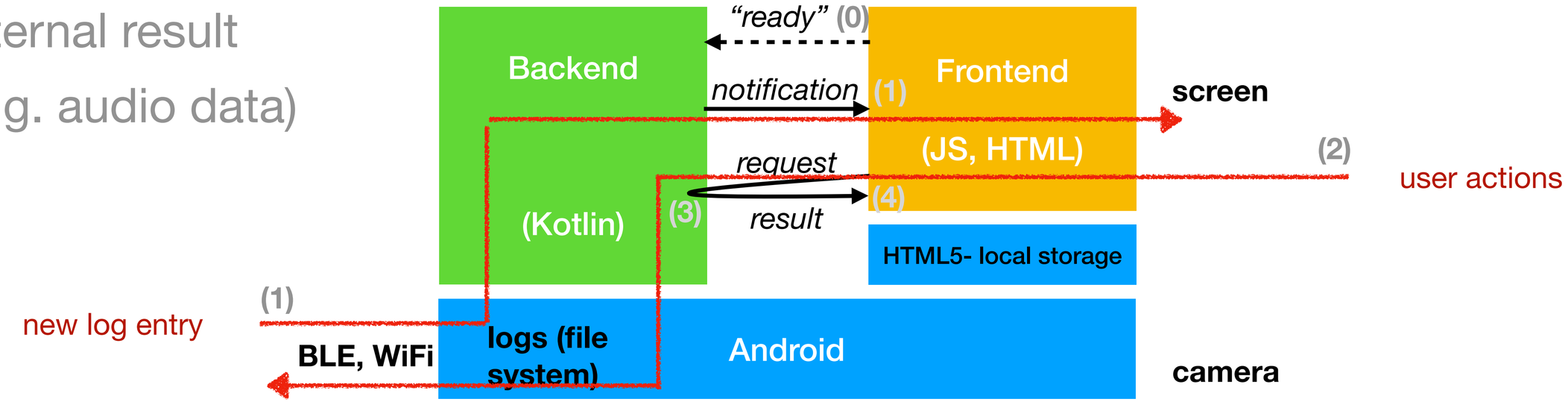
(1) network event (received new log entry), leads to upcalling the frontend

(2) user actions (HTML triggers)

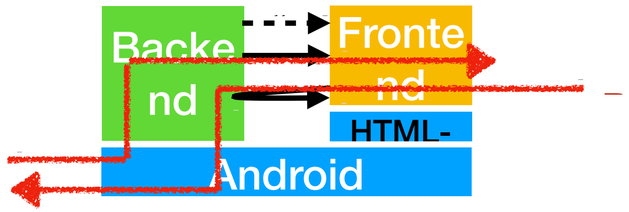
(3) internal service request (e.g. “start audio recording”, “sign and publish”)

(4) internal result

(e.g. audio data)



Processing User Input



Example: “sending and receiving a chat message”

a) click on send button 

b) Frontend calls the backend, using JavaScript: Parameters must be stringified

```
backend(“publ:post [] btoa(text) btoa(voice)”)
```

c) Backend creates a list data structure, first field identifies the app (TextAndVoice):

```
['TAV', [], “text”, byte_array]
```

This list is BIPF-encoded, cryptographically signed,
turned into one or more 120B data packets. Replication protocol kicks in.

d) On receiving a new log entry e, the backend makes an upcall into the frontend

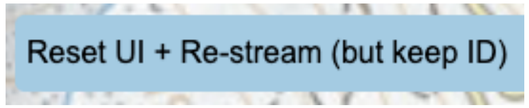
```
b2f_new_event(e)
```

note: e is a dict, contains de-BIPFed data, signature was stripped

The frontend filters on ‘TAV’, renders the message in the chat’s HTML table

The `tremola` Dictionary for “Cooked App Data”

There is no database yet: the frontend keeps an object (dict) called `tremola` that is persisted in the HTML5 local storage area

- Each app puts its vital state inside that `tremola` object
- Periodically, this object is persisted (local store of HTML5)
- At init time, the `tremola` object is loaded from the HTML5 store
  - contains decoded chat messages (no need to re-read the logs)
  and we wait for new incoming log entries
- Developer hack: click on “Settings -> 

in order to erase the `tremola` object and then replay all log entries (to re-build `tremola`)
  (beware of side effect like re-sending receipt confirmations …)

A Simulated Backend (for Chrome)

Tool to develop application logic in JS, without recompiling the Android app

Approach: “virtual backend” (see next slide)

- requires a Chrome-Browser (and/or Firefox?), no Kotlin or Android
- features the “Kanban mini-app” of Mr Heisch (BSc thesis)
  as an example about how to install a mini-app

App logic in JS and HTML can later be inserted into the Android tinySSB app, should work unchanged

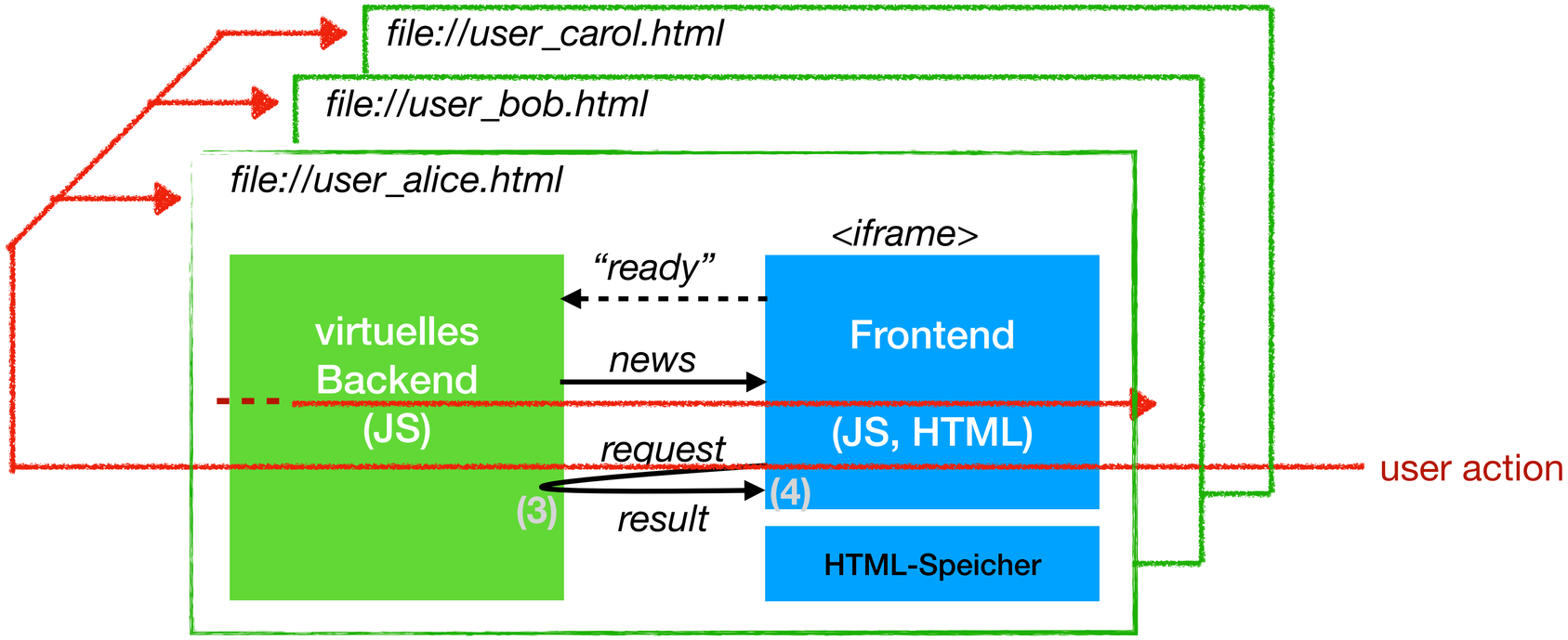
virtual Backend (in JS+HTML)

Replaces Kolin and Android part

Using Chrome browser tabs:

- one tab per user (Alice, Bob, Carol)
- inter-tab comm via
  "browser broadcast"
- users Alice, Bob
  and Carol are
  preinstalled, have
  fake public key

- msg queue per user



Chrome Virtual Backend has off/online toggle

State of the msg queue
(in- and outgoing messages

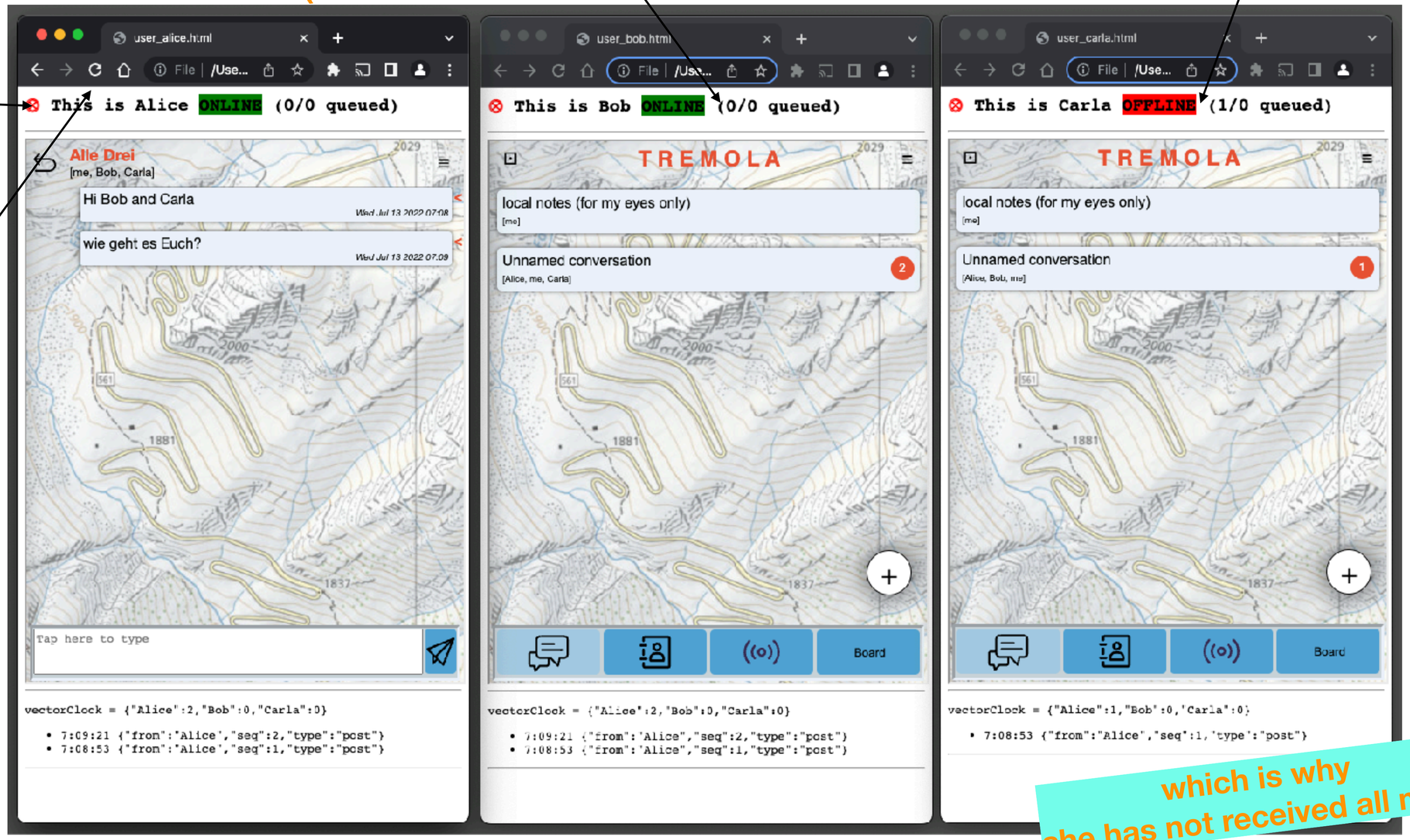
Carol is offline (toggle) …

Reset button: erases store
of ALL three users …

.. must the reload the SW in
each of the three tabs!

Vector clock: highest seen
sequenz number so far

List of receive log entries



which is why
she has not received all msgs

Log-Eintrag vom Backend zum Frontend

A log entry e is delivered as a dictionary:

```
▼ 2:
  ▶ header: {tst: 1716416562050, ref: 107305, fid: '@AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=.ed25519', seq: 10}
  ▶ public: (4) ['TAV', 'hello world', null, 1716416562]
```

Header: Metadaten des Log-Eintrags
(Timestamp, Hashwert, Absender, seq-Nr)
Public: content of the log entry, as a list (de-BIPF-ed)

Log entry a list with:
`'TAV'`, `'KAN'` - tag that identifies the mini-app for which this entry is meant for, followed by arbitrary many args (no formal schema doc, so far - it's ad-hoc)

Example 2:

```
▼ 2:
  ▶ header: {tst: 1716416928065, ref: 643024, fid: '@AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=.ed25519', seq: 11}
  ▶ public: (5) ['KAN', 'null', 'null', 'board/create', 'new Kanban board']
```

Walk Through for the IAM mini-app (“I am”)

The IAM mini app: can define a display name for one’s own public key, will be told to the whole world

format of log entry: `[“IAM”,“Alice”]` (log’s feed ID is the public key)

Two event handlers for the mini-app:

on user input:
  if my display name changed then
    - Frontend sends `"iam QWxpY2U=“` to Backend (base64-encoded display name)
    - Backend creates log entry with content `[“IAM”,“Alice”]`

on incoming event `[“IAM”,“Bob Jr.”]` in Bob’s log:
  if Bob’s display name not set manually then
    set it to received value

Properties of the Virtual Backend

The three browser tabs do not replace three real, independent devices:

- all three tabs must be open *before* an app action can be placed
- reason: a straggler tab cannot request old (missed) log entries
  (no sync protocol is implemented, just the push of new log entries)

Steps after modifications to the software (JS or HTML):

- push the reload button on each of the three tabs, **and**
- push one of the “Reset” buttons (this re-initializes all three clients)

When a tab “goes offline”, all log entries for and from it are queued and will be released (in both directions) when the tab is again put in online mode.

Use “Developer Tools” for debugging

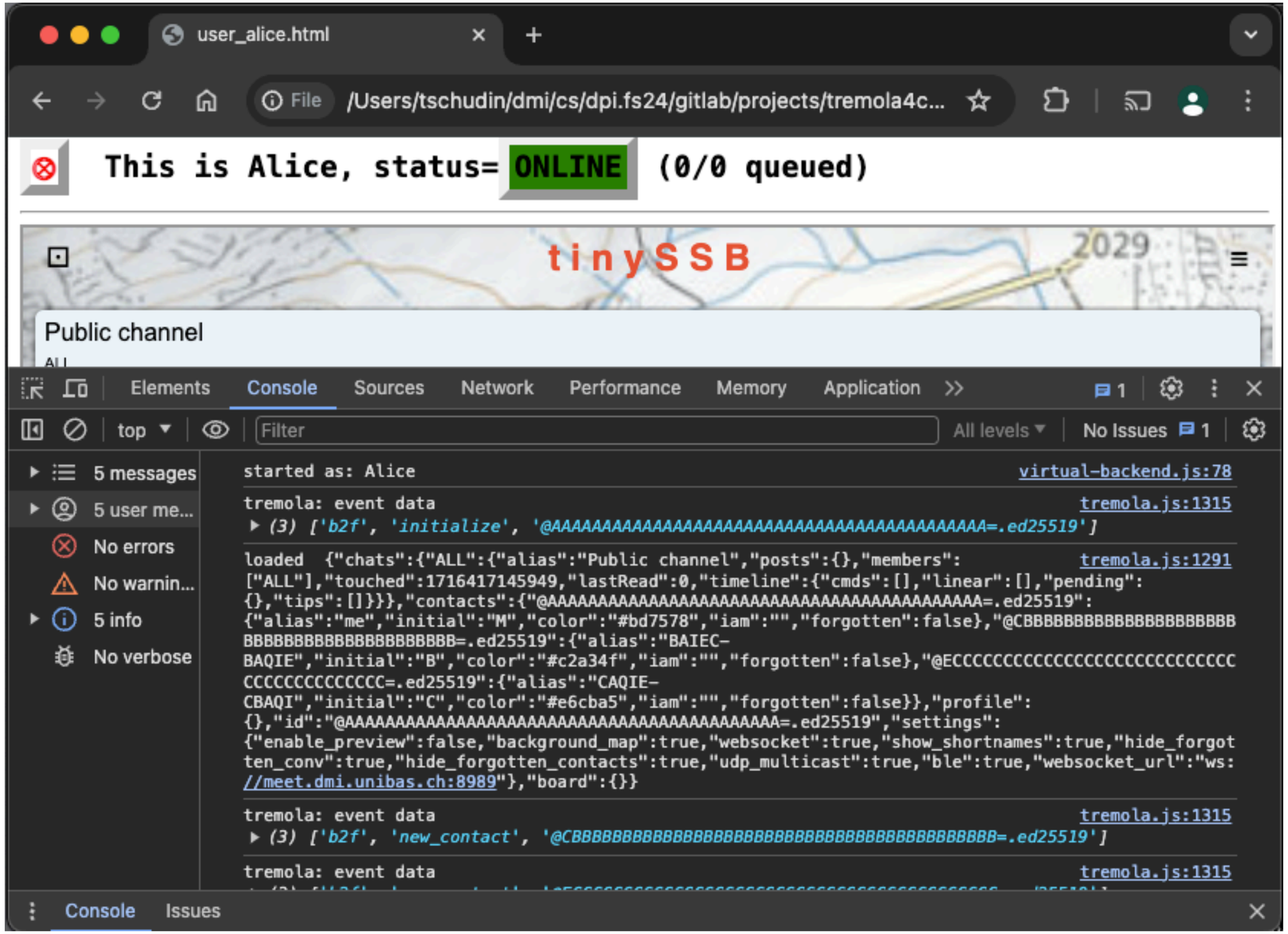
Open with:

View ->

Developer ->

Developer Tools

Shown: —> content of the `tremola` “cooked state” that is also persisted.



Outlook for the Android SW Architecture: split of backend into “foreground” and “main”

Problem of the current Android app (Dec 2024):

- BLE and ws sync is only active when the app is open and visible to the user

Ongoing project: running the sync protocol in the background. In Android, such a service is called “foreground” 8-)