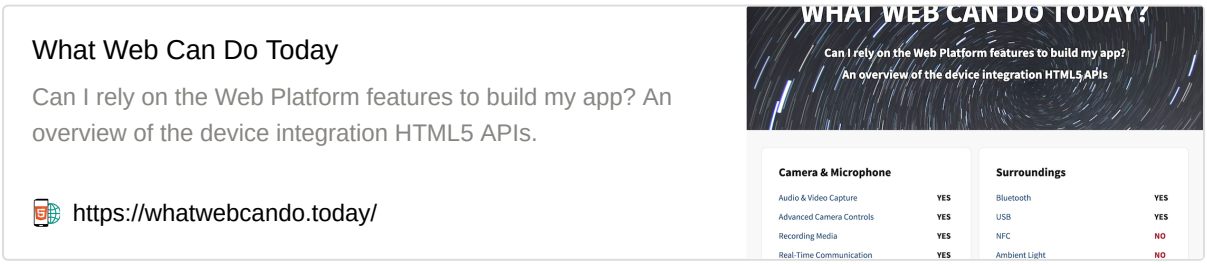
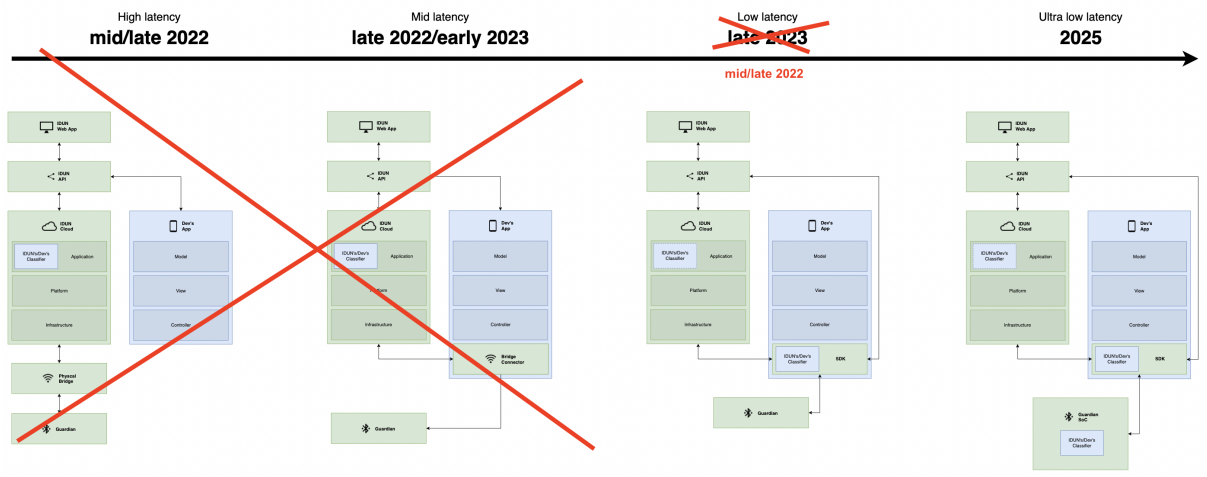
Web-first approach proposal

In order to accelerate the speed of development in all software aspects here at IDUN, we have proposed to take a web-first platform approach to our efforts (also called web-native approach). This means that we skip two steps on the initial software architecture roadmap and go straight to the point where we have a mobile SDK, mobile app and classifiers that are also used for mobile apps – all within the web ecosystem. The mobile app will also already communicate directly with our device via Bluetooth.



Benefits compared to the previous version

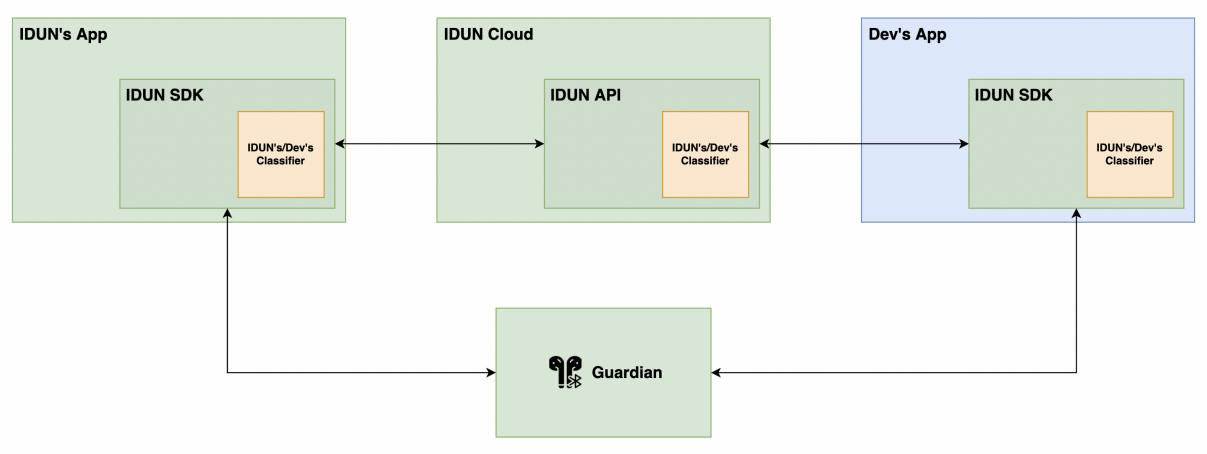


First, we can have a fully mobile version with low latency as early as 2022, compared to late 2023, so we **shave off 1.5 years from the original roadmap**. Also, we can get rid of the physical Bridge and associated codebase that is already being phased out, too – no refactoring or anything like that is required, the firmware people can now solely focus on the firmware. Also, we are developing our web application while simultaneously creating the SDK and mobile application, which drastically reduces development time. And because we rely on web technologies, we

don't need to outsource or hire native app developers for the upcoming ventures in the original roadmap since the knowledge already exists inside the company. Last but not least, the fact that we are developing an SDK that we also use for our web application means that we are customers of our own SDK for a core product here at IDUN, which is an underestimated benefit over developing an SDK that we don't use to build an internal core product like the web app.

What are we doing differently than before?

We are going away from the Bridge from now on and focusing on direct communication between the Web App and the BrainBox via the modern Web Bluetooth API. We are also creating a Progressive Web App (PWA) from our Web App, which allows it to run without an internet connection and lets the user install our app on their device so that it looks and feels like a native app (on mobile and desktop). Since the code that connects to the BrainBox runs on the end user's device, we already have low latency in visualising and displaying the data in the Web App – so the data can already be visualised and even processed before it is actually sent to the cloud. To go one step further, we also want to use WebAssembly, a low-level compilation target for high-level languages like Python, Rust, etc. So we can develop classifiers in Python, compile them in WebAssembly and send a lightweight and highly efficient byte format over the internet to users' browsers with native performance. This makes it possible to classify data from the bridge on end-user devices without an active internet connection with low latency, while controlling and hiding the intellectual property of our code. Since the PWA is web-based, we can also deploy the lightweight AWS Kinesis SDK asynchronously over the internet, while as of before users of our SDK or users of third-party developer apps with our SDK would have had to download many MBs just for the native AWS Kinesis SDK. So the streaming challenge is already solved in a serverless, high-performant and scalable way without having to create a custom solution. Here is an architectural overview diagram of the new holistic software architecture:



What are the downsides?

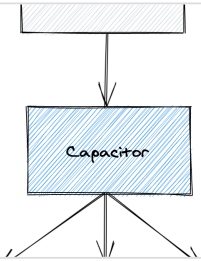
We use an experimental technology: Web Bluetooth. The specification is still under development by the W3C community group and browser compatibility is below the 90 % threshold. Apple is lagging behind the most, as the company has been against web technologies since the death of Steve Jobs and relied more on its monopoly in the App Store. Recent court cases and also releases in the Safari ecosystem have shown that Apple has started to adopt modern browser standards and APIs, so it is only a matter of time before PWAs and Web Bluetooth have production-ready compatibility on all major devices and platforms.

Until then, we will focus on developing our PWA for Android, Windows and macOS. If iOS needs to be supported, we will continue to go the App Store route via Capacitor.js. Third-party developers are therefore best advised to simply build a PWA or use a cross-platform technology like React Native or Ionic to build their apps and use our JavaScript (TypeScript/Node) SDK. Our SDK also works in other frameworks, even native ones (Swift and Java etc.) with some extra effort – it's just important to know that we focus on our JavaScript SDK first and in a later phase, as the software team grows, we can extend our SDK to port it to other languages and frameworks as well (even beyond mobile, e.g. for a game engine like Unity etc.).

Capacitor: Turn Your Web App into a Mobile App

Learn what is CapacitorsJS and how it works. Alongside, a step-by-step guide to convert your web app into an Android or an iOS app

https://javascript.plainenglish.io/capacitor-turn-your-web-app-into-a-mobile-app-4d114249e55b



List of famous apps using modern web technologies such as PWA and WebAssembly

- Google Maps
- Figma
- Uber
- Spotify
- YT Music
- Twitter
- Draw.io
- Stackblitz
- Google Stadia
- Facebook Gaming
- Microsoft Xbox Cloud
- etc.

Who uses Capacitor? Is it production-ready?

Capacitor reached already version 3.0 in 2021 and version 4.0 is on its way. The repository as well as the community is very active and well maintained. Companies such as Microsoft, Target, Burger King, Disney, Home Depot and General Electric use Capacitor in production.

What is the motivation for us to start with the mobile SDK/progressive web app and not with the cloud platform?

- In terms of current skillset, we are much faster starting with this approach until we add engineering staff (e.g. senior full-stack or senior software engineer) for the cloud development efforts.
- It gives us already sufficient visibility of the software product to the outside world, that is investors/external partners/customers. Also this integrates previous efforts of the offline solution with the refactored web app, the features are (but not limited to):
    - Live streaming
    - Impedance Check
    - Data quality assessment
    - Demos with classifiers
    - **Future**: Recording (needs cloud integration) etc.
        - Some of the persistency can be achieved by the web app as well, but this should just be used for caching and buffering
- The PWA will support us as well in in-house development tremendously (as it did with the offline solution, e.g., experiments in PoCs MUST be top notch stable, flawless as it can be) and has the added benefit of being easily understood by externals.
- Above all, we implicitly work on our device in terms of on-device edge computing which is a topic in multiple years ahead, furthermore the hardware and electronics team can free up resources from bridge software development (because the bridge is inherently in the PWA) and focus on future versions of the Guardian hardware. The software team will have strong development links with the hardware people
- Provision of the cloud platform might be of high relevance in the future to generate revenue (**if and only if** external partners are willing to integrate their

efforts and modules on our cloud) but up to this point uncertainty in development time in this regard is rather high and also the amount of collaborations

- One of the biggest problems we faced (and will always face in the future) was “why doesn’t this work?” or “why is it so buggy?” or “why did it work yesterday, but not today?”. We do not have sufficient time to properly test software – although we should. There is obviously a trade-off between quality and quantity and the amount of tech debt we create. So going this way as mentioned above, steps of software development is clearer and aims to make development in each iteration more robust as well as we can traceback errors more easily (which is essential as the full-stack gets more and more complicated): As we are not a pure software company but aim to go full-stack (here, meaning hardware + software), the development cycle should **ALWAYS** start from the material to some output on the web app. This is or should be repeated every time. Every single change in the chain will also alternate the visible output at the end of the chain. We need to make this more robust.

  The proposal is, we start from the core (materials + hardware) and build around it, each time adding a lego block

  1. **The core**: Stable offline solution (Relevant for material + hardware testing and neuroscience) - which we have now
  2. **The shell**: Stable PWA fulfilling the requirements of the offline solution and the actual web app, but making a step towards the customer/stakeholder for ease of use and demonstrating the potential of the Guardian device in its whole breath via robust/creative demo applications
  3. **The colouring of the shell**: If we know what is possible with the PWA and if we then know what we need to add on top of it, e.g., recording of the data or adding customer ML models, it is much easier to identify the features which need certain aspects of the cloud infrastructure and we can also better identify the potential of what external developers can achieve. (Additionally, it gives us more time and identify the missing skillsets we need to add to the team)
  4. **Breaking out the shell:** Being ready and completing all previous steps, we are then moving towards the cloud API to provide it as a service

Stuff to consider

- **Q:** Background service capability of BLE, tombstoning. See attached issue:

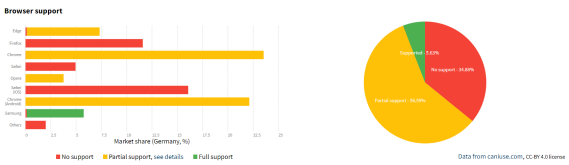
  > https://github.com/WebBluetoothCG/web-bluetooth/issues/422

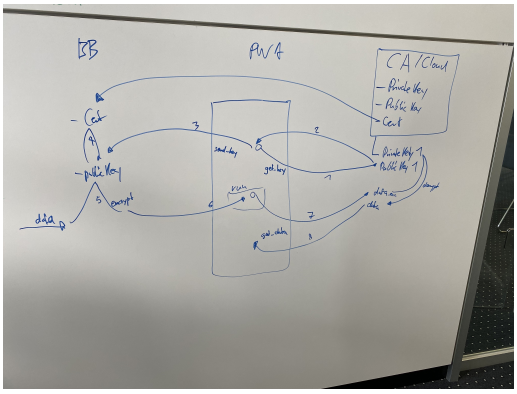
  A: Work in progress, we would need Web Bluetooth to be available in the Service Worker context, but at the moment this is not implemented yet. For that, we would need to go through the app store and use something like this with Capacitor:

  > https://github.com/capacitor-community/bluetooth-le

- **Q:** Bonding and encryption
  A: Web Bluetooth works in a sandboxed environment (the browser engine) and is – according to the implementation docs – very safe. The firmware on the device needs to implement security measures, but this is in general a discussion we need to have: encrypted streams, only streaming data if an access key has been provided which would expire after a given timeframe etc.

- **Q:** How to enforce to use the cloud even with a sophisticated SDK
  A: We only send encrypted data from the Guardian to the connected Bluetooth device, meaning that the connected device can receive data, but cannot do anything with the data since it's encrypted. For that we would utilise an end-to-end encryption with private key provided statically on the device in the firmware before shipping and through a public key provided by the API from the cloud. Here's a overview of how this flow would work:





Feedback Niclas Granqvist

- Niclas Granqvist (He/Him) 22:15

  Thank you Moritz! I do have been aware of this API since years now. I have never needed to use it myself or in any project. For Bluetooth as such it's just an API. The question is mainly wether it works and is fast enough. I do think that it probably works and is good enough for what you are doing, mainly small demos I suppose. I would move forward and try it a little bit and if it works then settle for it. I don't think that the fact that you have to use Chrome or something like that is an issue. You can always choose your browser.

- Assuming that it works I think that this can really be a good strategy and save some development time. I suppose you anyway mainly display data in the browser.

- We mainly use an app to display data. We are using flutter which is a multi-platform approach.

- I don't recommend to use flutter if you don't want to spend a lot of development time. As such, when you are comfortable with flutter it is actually the fastest and one of the best APIs to use on Android and iOS.

Issues during the first initial MVP implementation (PWA+Bluetooth+ Data Processing)

- **Data processing** on the web browser (at least the requirements that we have) using Python to web assembly (wasm) is not straightforward to do. While it makes sense to run simple calculations and convert basic python functions to wasm and using glue code to actually do something with it on the browser will

work just fine, adding comprehensive packages like **scipy** or **numpy** cannot be used as out of the box and need to be transpiled as well. Also, each added python module will add overhead and increase the size of the packages that need to be installed (can go up to 100 MB). In pyodide this process only needs to be done once, not sure about the wasm file itself

- Some fruits for thought:
  - Data processing on the web browser is in general a temporary transition because at some point we will move on the edge data processing. It does not make sense to build the foundation for getting python packages run on the web browser and spend a lot of time in this, so the focus should not be translating complex cpython to wasm. ⇒ at this stage for quick MVP it might be worth to encapsulate the functionality of pyodide (which is a python interpreter runtime to web) and test whether the PWA can handle python data processing + classifier output, however here again pyodide is not production ready and in the experimental stage
  - TBC, not done yet, still investigating some stuff