Bringing Android 13 to Tesla vehicles with Flutter Web

FlutterCon 2023

Michał Gapiński - Senior Software Engineer - HappyByte GmbH

Who I am

- Based in Szczecin, Poland
- 26
- SWE with experience in Embedded and Mobile
- Tinkering with AOSP since 2011
- Transitioned to Flutter from iOS in 2020, never going back

Tesla infotainment limitations (12.2021)

- Lack of PL navigation voice
- Outdated maps in PL
- Lack of 3rd party chargers (ABRP)
- No info about speed cameras
- No Apple Music integration
- Limited BT audio (Playlists, up next etc)
- Limited HV battery diagnostics

What is the solution?

Provide support for CarPlay/Android Auto without performing any modifications that would affect the vehicle warranty.

Finding the right approach

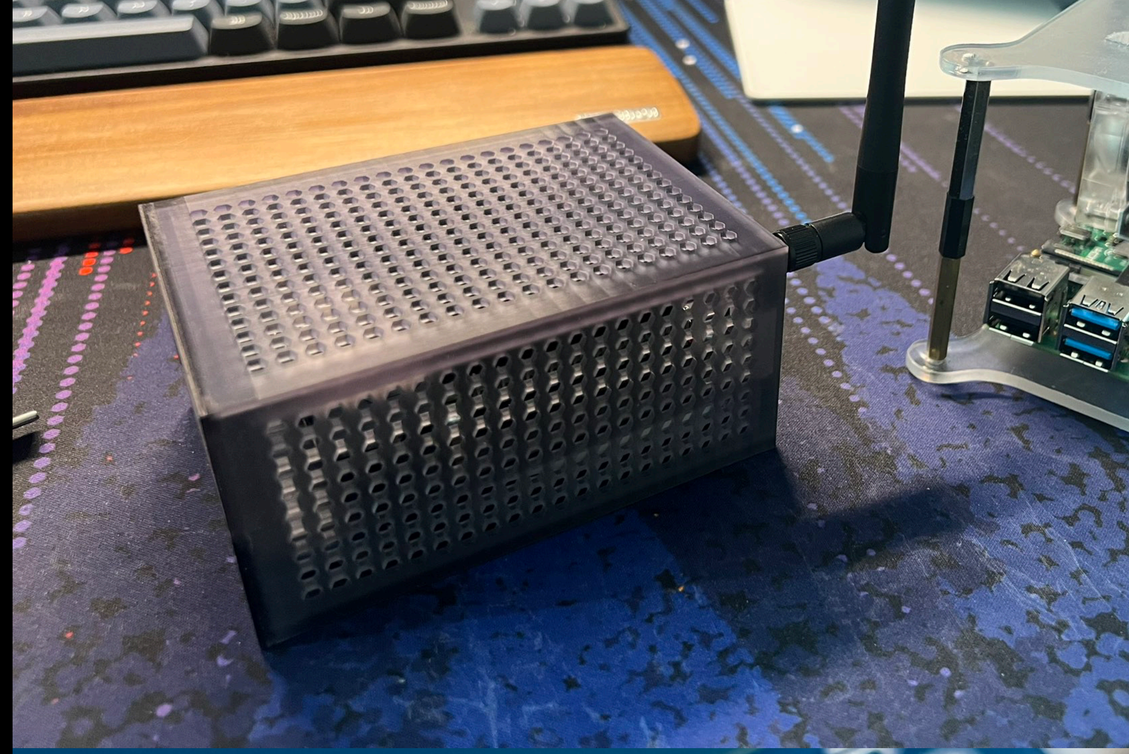
- CarPlay is proprietary and requires the MFI chip for the handshake
- Tesla infotainment does not support 3rd party apps; even if it would, the hardware access would be restricted
- The only approach that does not involve rooting the OS is the Chromium browser that works in motion (in most parts of the world) :
  - WebGL is supported - Flutter Web runs very well
  - Tesla regularly updates the browser (support WasmGC & WebGPU will be there at some point)

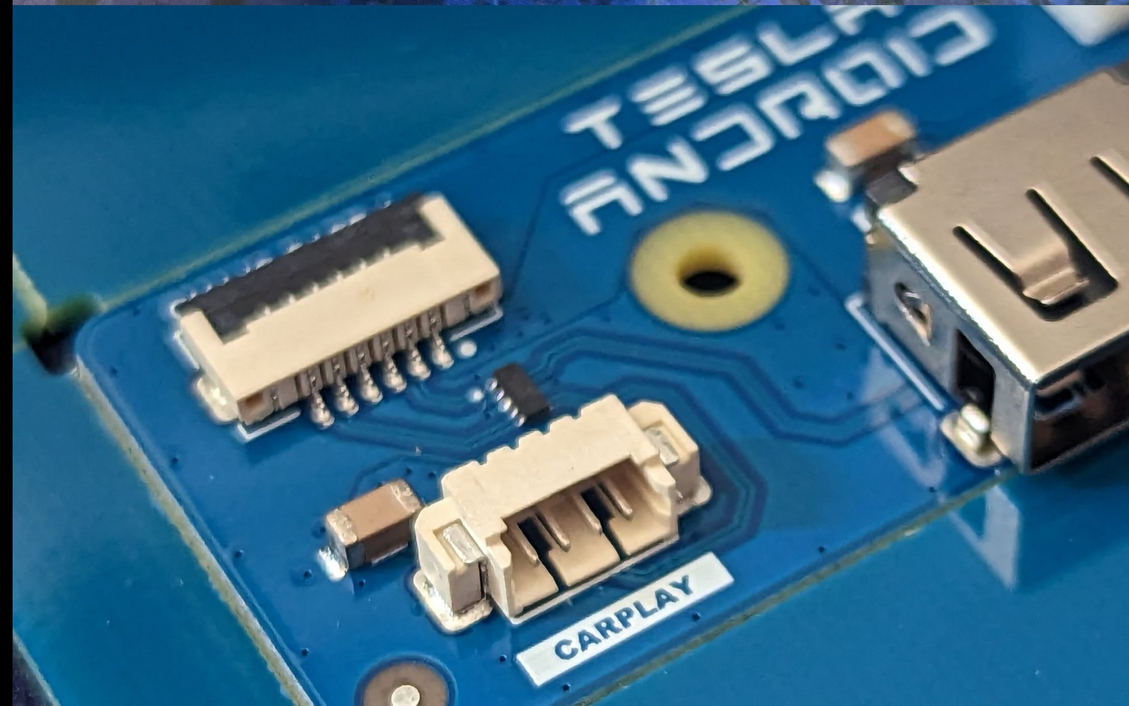
Why Android?

- Need to use 1st party client apps for CarPlay interfaces
- I did not want to reverse-engineer anything
- Higher probability of achieving a fully functional solution
- 3rd party apps support
- Full utilization of the SBC, providing the most value the end user
- Porting standard Linux software is „easy”

Hardware requirements

- Raspberry Pi 4 / Compute Module 4
  - Compatible with mainline kernel/mesa3d
  - Powered via USB-C PD
  - Decent Wi-Fi interface
- USB LTE Modem
- 3rd party CarPlay interface for Android headunits





Proof of concept

- Zero focus on required hardware (multiple SBCs, HDMI capture interfaces etc)
- Won’t move into the MVP stage until all the potential blockers are identified and resolved
- Get it done in 2 weeks

Next steps

- Published the POC when it became usable
- Tons of positive feedback from the community
- Decided to build something the average Joe could get running in 15 minutes

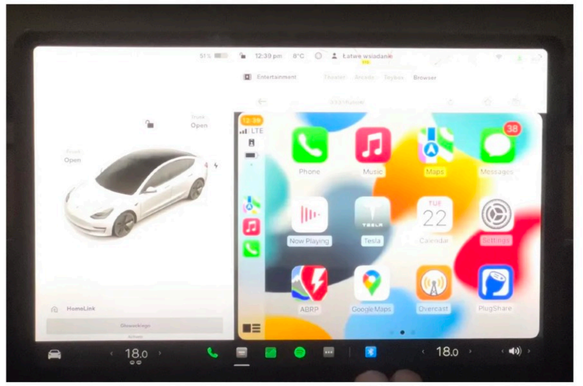
The Verge

Menu

TRANSPO / TECH / GADGETS

You can hack Apple CarPlay into a Tesla using – what else – Android / Plus one or two hundred bucks in hardware

By MITCHELL CLARK
May 9, 2022 at 9:03 PM GMT+2



Thanks to web standards. Image: Michał Gapiński

Michał Gapiński
@mikegapinski · Follow

The resolution is now perfect 😍😍😍 #teslaCarPlay

6:12 PM · Jan 14, 2022

461 Reply Share

Read 37 replies

461 51 37

Impressions
307K

Engagements
20,022

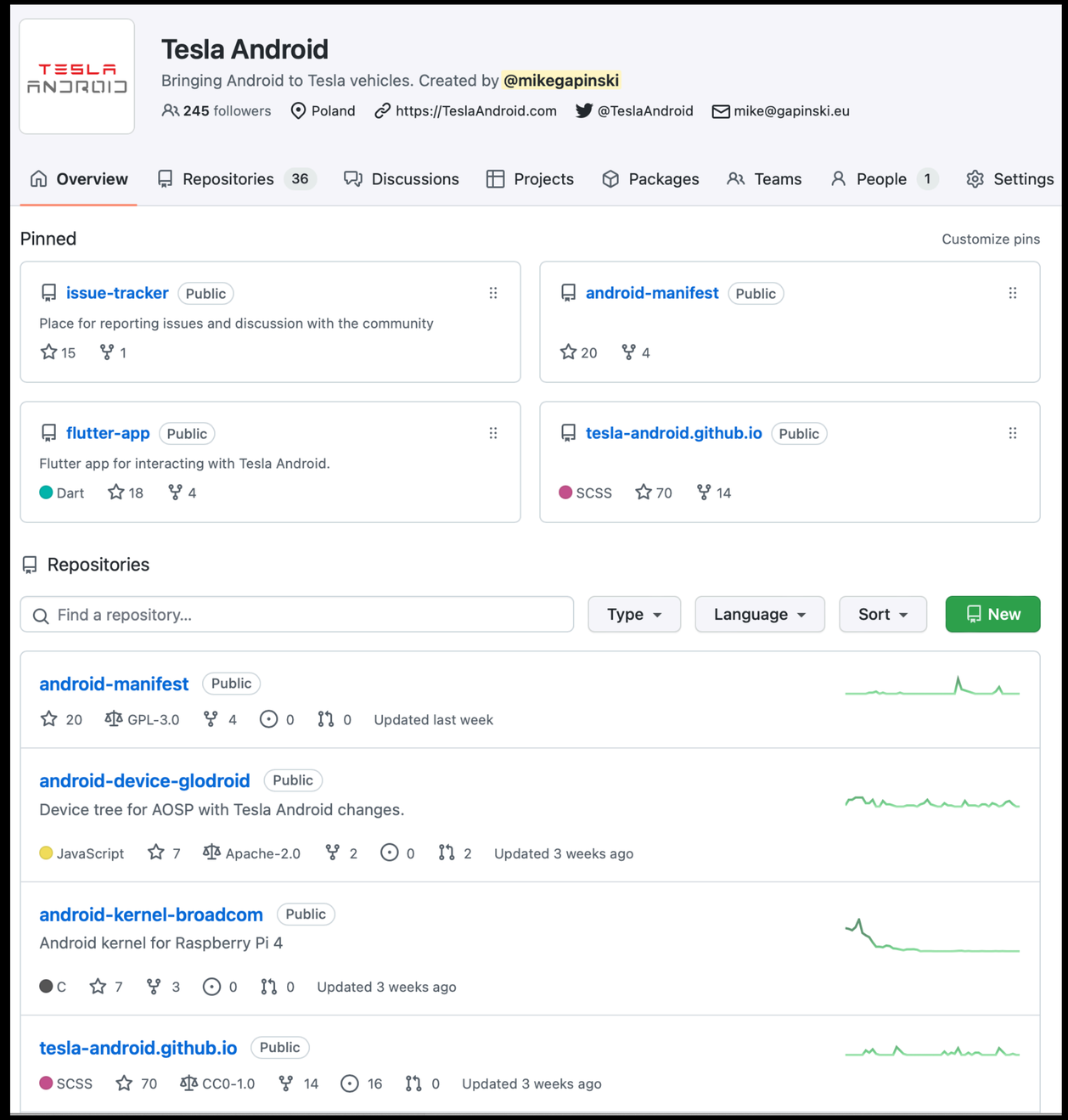
Detail expands
7,192

Core software components

Everything is open-source

GPL-3 license

The most commented bug took 200 comments and a few months to resolve



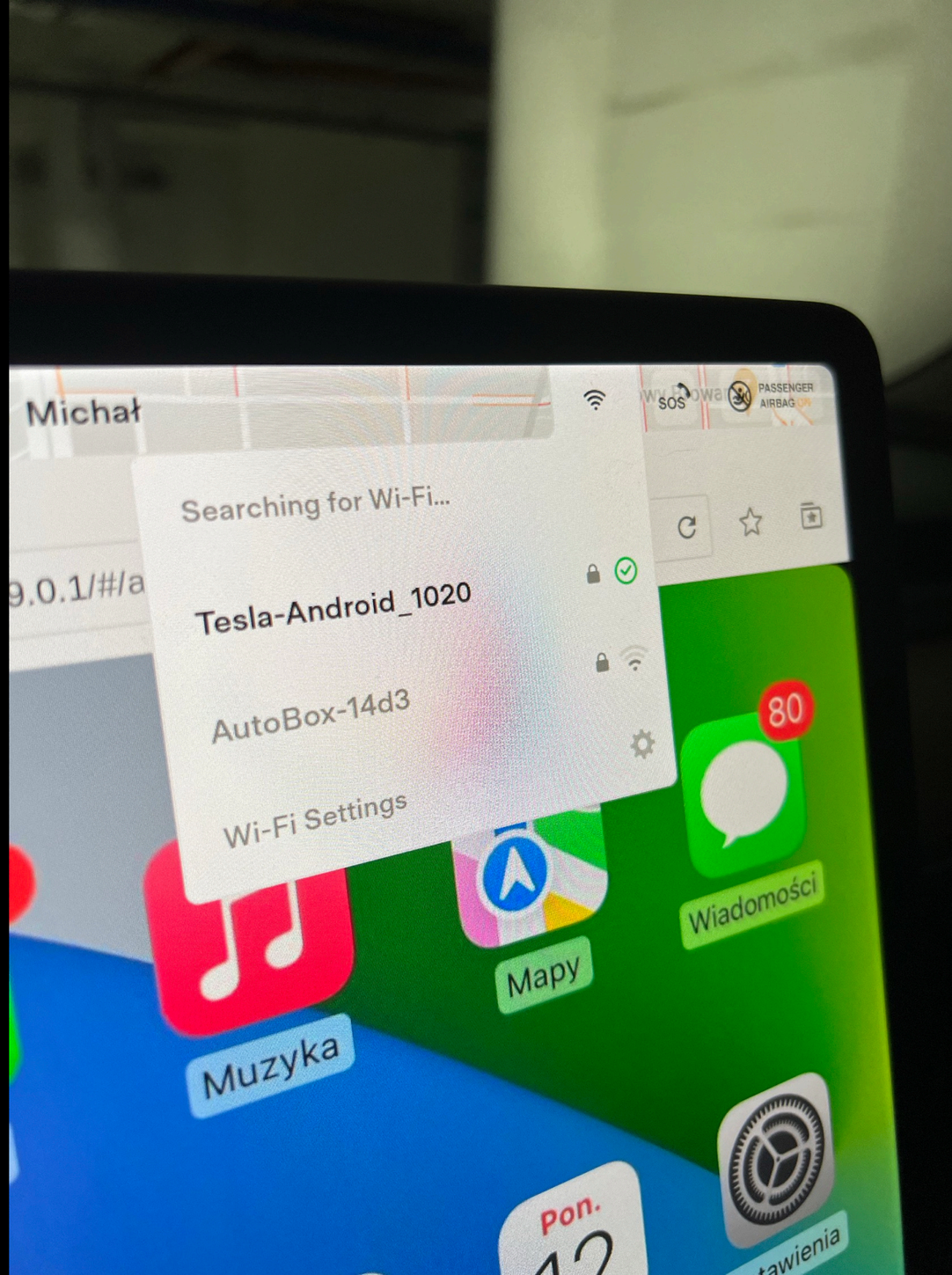
Networking

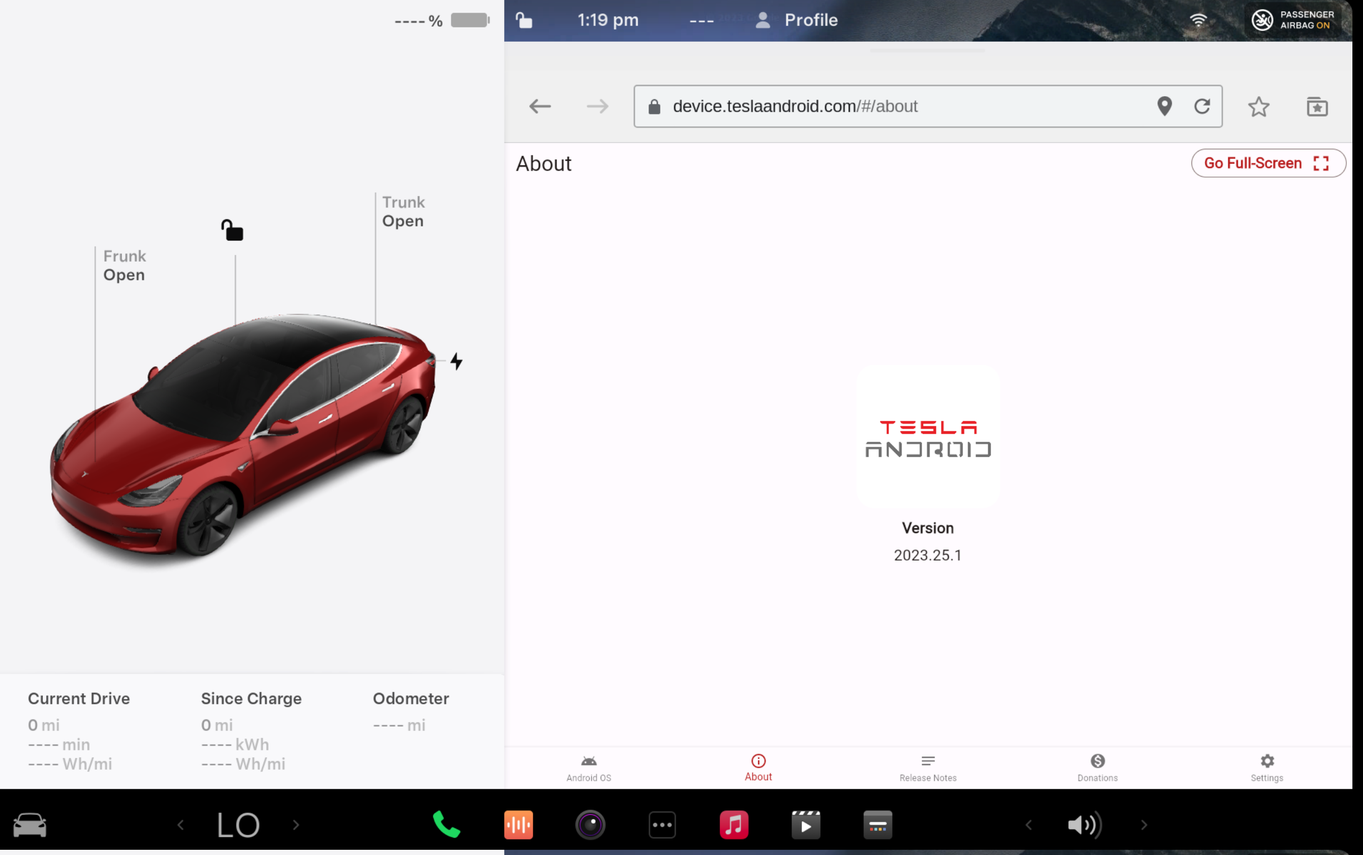
- Permanent Wi-Fi hotspot
  (AC 5GHz, 40MHz channel)
- lighttpd ported to the AOSP build system hosts the Flutter app and acts as a TLS termination proxy for all WebSockets and the REST API used for configuration
- iptables handle redirecting the traffic from a public address to LAN
- DNS spoofing:
  - connman - connectivity check bypass
  - full screen launcher for Tesla Theatre

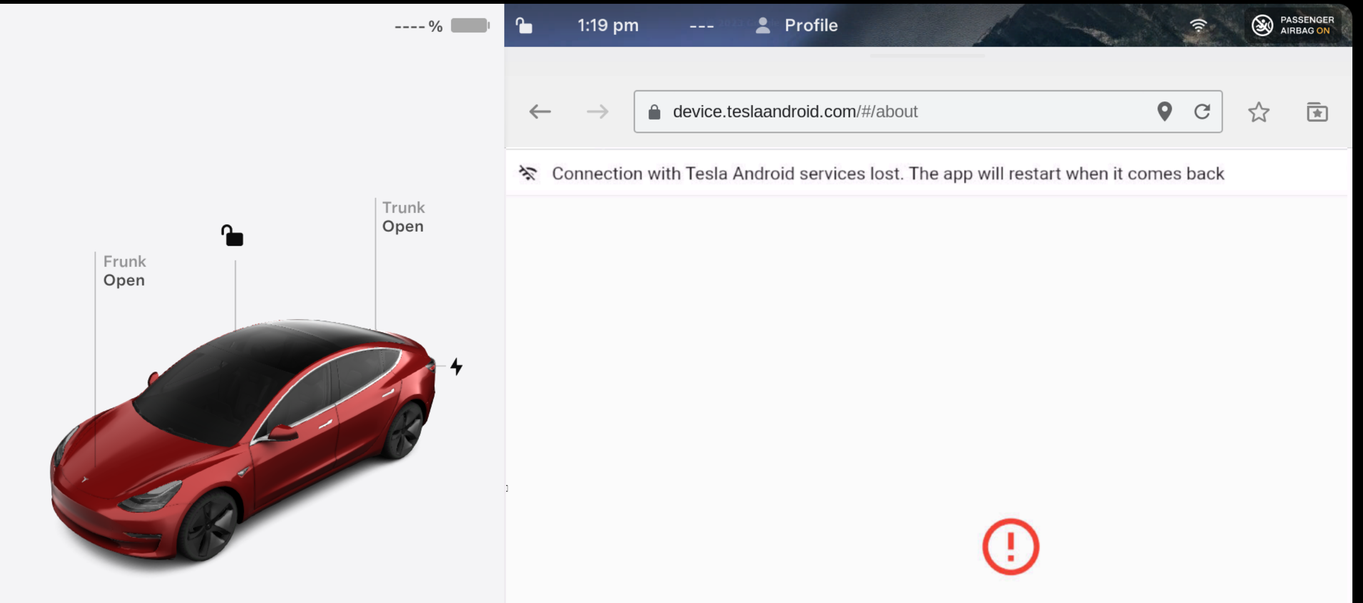
```
@@ -257,6 +270,42 @@ int TetherController::startTethering(bool usingLegacyDnsProxy, int num_addrs, ch
                                     dhcp_ranges[addrIndex + 1]));
     }

+    argVector.push_back("--address=/www.youtu.be/104.248.101.213");
+    argVector.push_back("--address=/fullscreen.device.teslaandroid.com/104.248.101.213");
+    argVector.push_back("--address=/www.fullscreen.device.teslaandroid.com/104.248.101.213");
+    argVector.push_back("--address=/device.teslaandroid.com/104.248.101.213");
+    argVector.push_back("--address=/www.device.teslaandroid.com/104.248.101.213");
+
+    // FIXME: Remove deprecated endpoints in Sept 2023
+    argVector.push_back("--address=/fullscreen.app.teslaandroid.com/104.248.101.213");
+    argVector.push_back("--address=/www.fullscreen.app.teslaandroid.com/104.248.101.213");
+    argVector.push_back("--address=/app.teslaandroid.com/104.248.101.213");
+    argVector.push_back("--address=/www.app.teslaandroid.com/104.248.101.213");
+
+
+    if(offlineModeEnabled == 1) {
+       argVector.push_back("--address=/connman.vn.tesla.services/104.248.101.213");
+        argVector.push_back("--address=/www.teslamotors.com/104.248.101.213");
+        argVector.push_back("--address=/hermes-prd.vn.tesla.services/104.248.101.213");
+        argVector.push_back("--address=/connman.vn.cloud.tesla.cn/104.248.101.213");
+        argVector.push_back("--address=/www.tesla.cn/104.248.101.213");
+        argVector.push_back("--address=/hermes-prd.vn.cloud.tesla.cn/104.248.101.213");
+    }
+
+    if(offlineModeTelemetryEnabled == 0) {
+       argVector.push_back("--address=/telemetry-prd.vn.tesla.services/104.248.101.213");
+        argVector.push_back("--address=/telemetry-prd.ap.tesla.services/104.248.101.213");
+       argVector.push_back("--address=/apf-api.prd.vn.cloud.tesla.com/104.248.101.213");
+       argVector.push_back("--address=/x1.ap.tesla.services/104.248.101.213");
+       argVector.push_back("--address=/s3.ap.tesla.services/104.248.101.213");
+       argVector.push_back("--address=/tesla-hermes-snapshot-eu.s3-eu-central-1.amazonaws.com/104.248.101.213");
+       argVector.push_back("--address=/tesla-hermes-snapshot.s3-us-west-2.amazonaws.com/104.248.101.213");
+    }
+
+    if(offlineModeTeslaFirmwareDownload == 0) {
+       argVector.push_back("--address=/va.teslamotors.com/104.248.101.213");
+    }
+
     std::vector<char*> args(argVector.size() + 1);
     for (unsigned i = 0; i < argVector.size(); i++) {
         args[i] = (char*)argVector[i].c_str();
@@ -557,6 +606,26 @@ int TetherController::setDefaults() {
         return res;
     }

+    std::string sourceIp = "172.16.0.1";
+    std::string destinationIp = "104.248.101.213";
+    std::string interface = "wlan0";
+
+    std::vector<std::string> natCommands = {
+        "*nat",
+        StringPrintf("-A PREROUTING -d %s -i %s -p tcp -j DNAT --to-destination %s", destinationIp.c_str(), inter
+        StringPrintf("-A PREROUTING -d %s -i %s -p udp -j DNAT --to-destination %s", destinationIp.c_str(), inter
+        StringPrintf("-A POSTROUTING -s %s -p tcp -j SNAT --to-source %s", sourceIp.c_str(), destinationIp.c_str(
+        StringPrintf("-A POSTROUTING -s %s -p udp -j SNAT --to-source %s", sourceIp.c_str(), destinationIp.c_str(
+       "COMMIT\n"
+    };
+
+
```

Networking

- Several modifications to Android frameworks, SystemUI, and Settings. Removal of everything related to Wi-Fi
- Kernel changes to drivers used by USB LTE interfaces to get them initialized and recognized as valid internet sources for Hotspot
- Got IPETH (iOS USB tethering) to work on Android
- Custom implementation for device initialization forced by lack of udev rules (AOSP libusb without hotplug events)







Flutter Frontend

Tesla specific requirements

Connectivity check

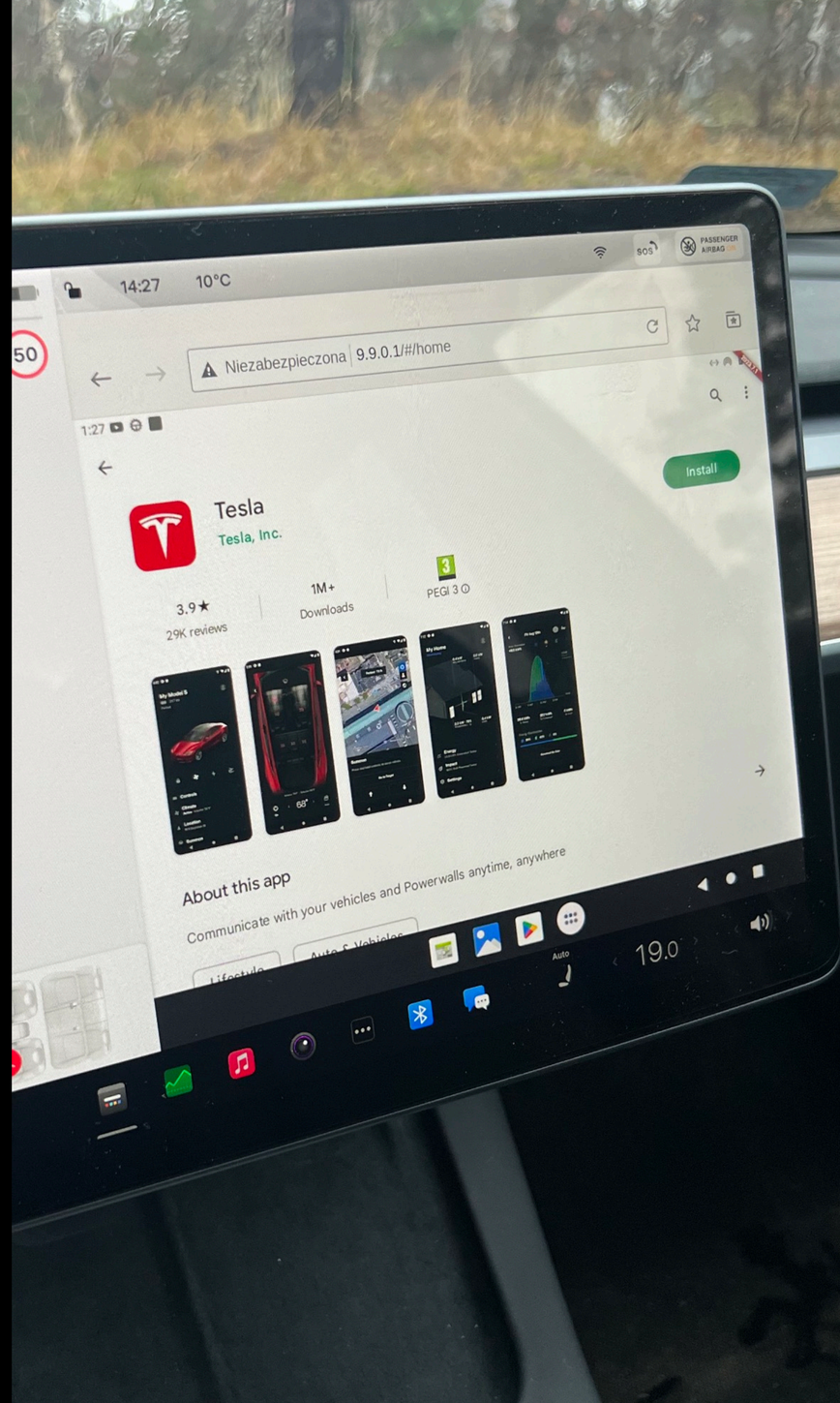
- WebSockets synchronised using the Health Check endpoint
- All Cubits/Transport classes are deallocated when the device is not accessible (avoids unnecessary polling)
- The Android OS module is reinstantiated after the Wi-Fi connection is established

WebGL performance limitations

- OOM issues with Full Self Driving visualisations active, browser gets killed by the Linux host
- canvasKitMaximumSurfaces need to be set 4
- Every unecessary redraw decreases performance

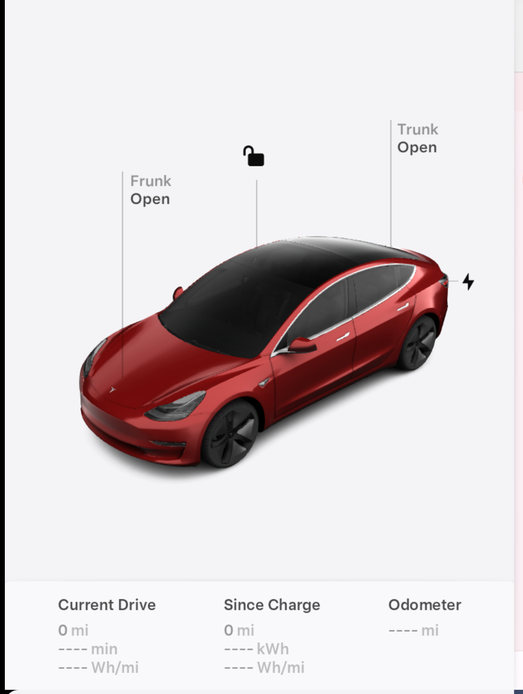
Virtual display

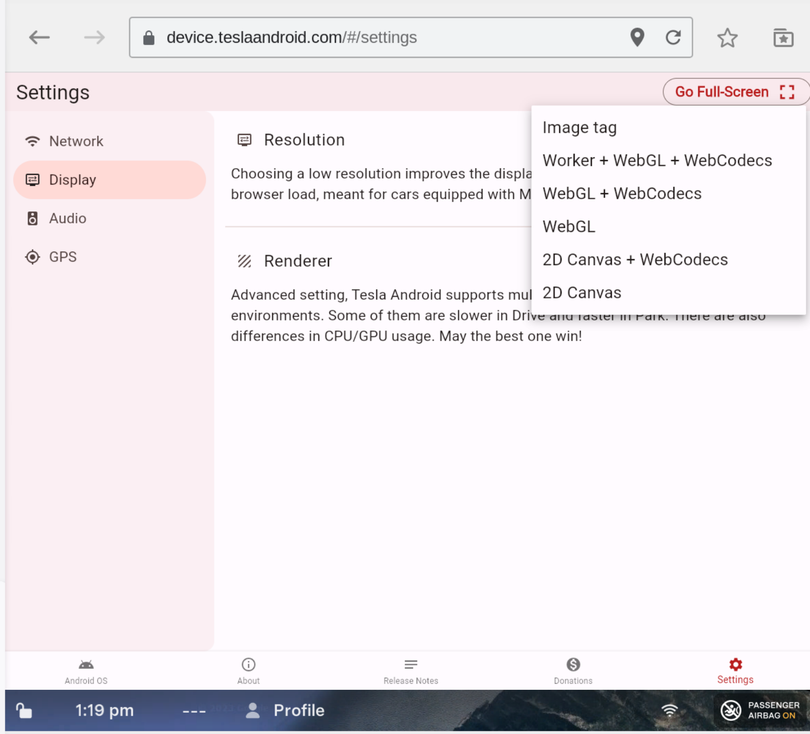
- It relies on a headless operation mode in the drm_hwcomposer - available upstream
- Dynamic resize via display hotplug event triggered via a system property - ready to submit
- RAW frames are taken from the SurfaceFlinger, encoded with the Broadcom HW Encoder and streamed via WebSockets
- SurfaceFlinger was modified to support screen capture of secure layers (DRM support for the Netflix app, HDCP bypass)

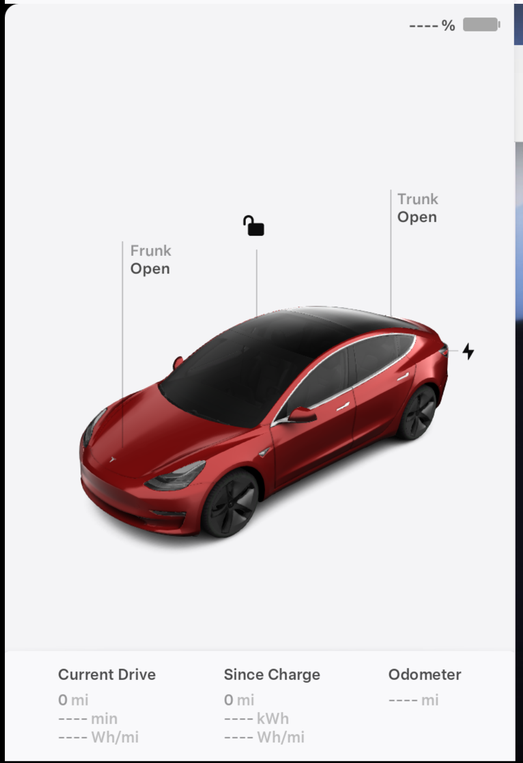


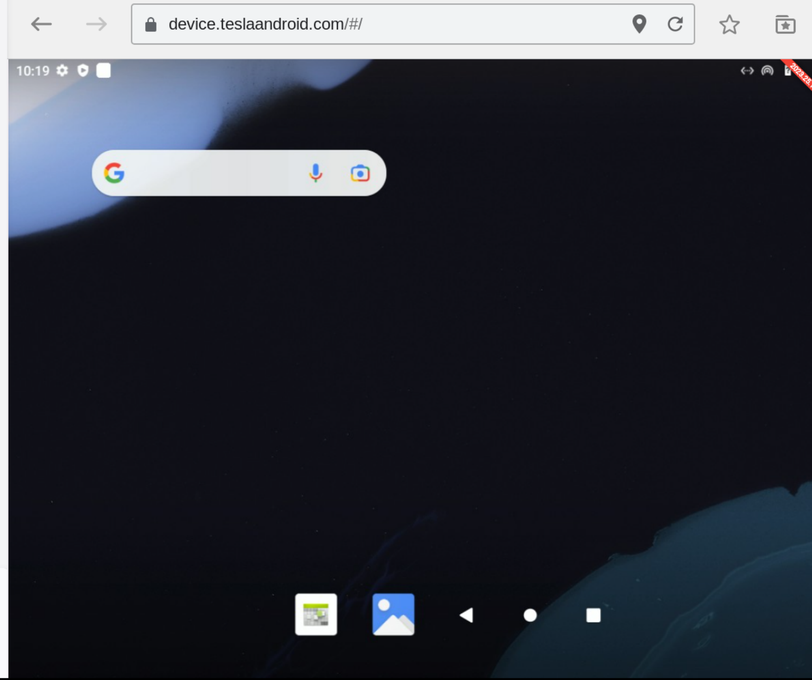
Virtual display
Frontend implementation

- Size from LayoutBuilder, dynamic resize
- Managed by a single cubit
- Optimal resolution calculated after each resize
- Configurable in the settings (resolution presets)
- Using WS transport, bytes sent to PlatformView via JS Interop
- Multiple renderers:
  - WebGL + WebCodecs (fastest, low CPU usage)
  - 2D Canvas + WebCodecs
  - WebGL
  - 2d Canvas
  - Image tag









Virtual touchscreen

- The kernel module exposes a character device that receives multitouch events
- Cross-platform, does not have limitations of standard Android Accessibility API
- Flutter Frontend maintains the state of each multi-touch slot and generates events compatible with the Linux Multi-touch (MT) Protocol
- A lightweight system daemon acts as a proxy between the browser and the kernel module

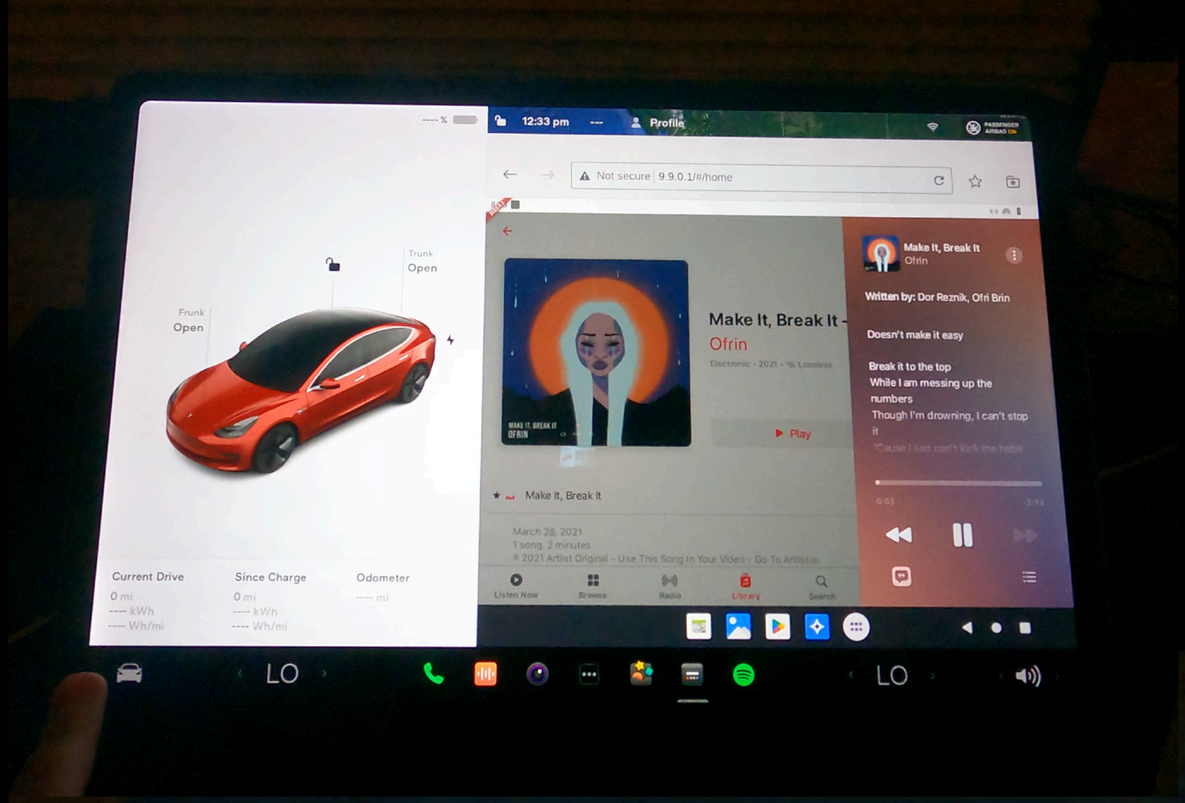
```dart
@injectable
class TouchscreenCubit extends Cubit<bool> with Logger {
  final List<VirtualTouchscreenSlotState> slotsState =
      VirtualTouchscreenSlotState.generateSlots();
  final TouchScreenTransport _transport;

  TouchscreenCubit(this._transport) : super(false) {
    _transport.connect();
  }

  @override
  Future<void> close() {
    log("close");
    _transport.disconnect();
    return super.close();
  }

  void handlePointerDownEvent(
      PointerDownEvent event, BoxConstraints constraints, Size touchscreenSize) {
    final scaledPointerPosition =
        _scalePointerPosition(event.localPosition, constraints, touchscreenSize);
    final slot = _getFirstUnusedSlot();
    if (slot == null) return;
    slot.trackingId = event.pointer;
    slot.position = scaledPointerPosition;

    log("Pointer down, assigned slot ${slot.slotIndex}, trackingId ${slot.trackingId}");

    final command = VirtualTouchScreenCommand(
        absMtSlot: slot.slotIndex,
        absMtTrackingId: slot.trackingId,
        absMtPositionX: slot.position.dx.toInt(),
        absMtPositionY: slot.position.dy.toInt(),
        synReport: true);

    _transport.sendCommand(command);
  }
```

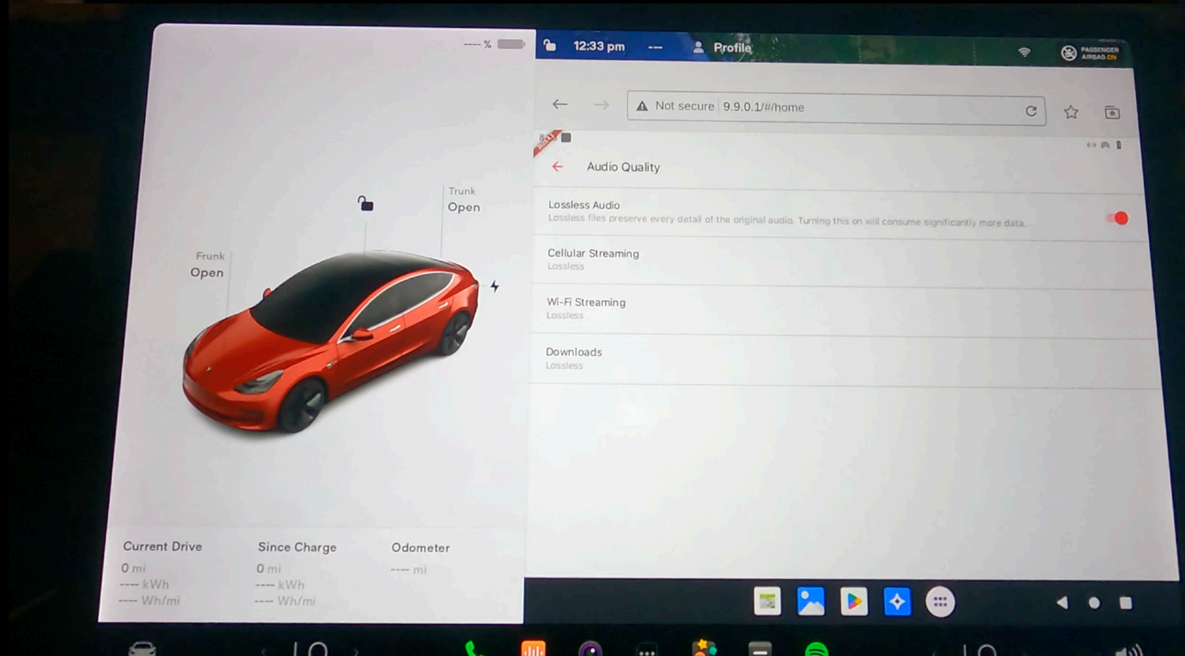
```dart
class VirtualTouchScreenCommand {
  final int? absMtSlot; //ABS_MT_POSITION_X
  final int? absMtTrackingId; //ABS_MT_TRACKING_ID
  final int? absMtPositionX; //ABS_MT_POSITION_X
  final int? absMtPositionY; //ABS_MT_POSITION_Y
  final bool synReport; //SYN_REPORT

  VirtualTouchScreenCommand({
    this.absMtSlot,
    this.absMtTrackingId,
    this.absMtPositionX,
    this.absMtPositionY,
    this.synReport = false,
  });
```

```dart
class VirtualTouchscreenSlotState {
  int slotIndex; // 0-9
  int trackingId; // active pointer index or -1
  Offset position;

  VirtualTouchscreenSlotState.initial({required this.slotIndex})
      : trackingId = -1,
        position = Offset.zero;

  static List<VirtualTouchscreenSlotState> generateSlots() {
    return List.generate(
      10,
      (index) => VirtualTouchscreenSlotState.initial(slotIndex: index),
    );
  }
}
```

Sound

- BT audio is not available in Tesla Theatre
- The capture of unprocessed audio frames takes place before they get to the HAL in AudioFlinger, no DRM restrictions
- Highest quality possible - Stereo 48KHz PCM stream sent directly to the browser
- WS transport, AudioContext fed with data va JS interop
- Made lossless Apple Music playback possible (Amazing in the MY3/Y LR and Palladium S/X)





GPS

- Hardware GPS from the Autopilot computer is accessible in the browser
- Manual polling with high frequency always provides a fresh set of coordinates (it is not always the case, especially on mobile platforms)
- Location data is fed to AOSP via WebSockets
- Updates are consumed by a modified version of the Goldfish(Emulator) GPS HAL.
- Disadvantages: speed needs to be approximated. Not always accurate due to GPS jitter

AOSP port features

- Google Play Services are fully functional, Tesla Android injects a device fingerprint from a certified device and satisfies the Device Attestation checks used by SafetyNet. Most apps can run without issues
- DRM playback is supported
- The OS comes without root, supports Android Verified Boot
- Seamless A/B OTA is supported, updates are installed when the car is being driven

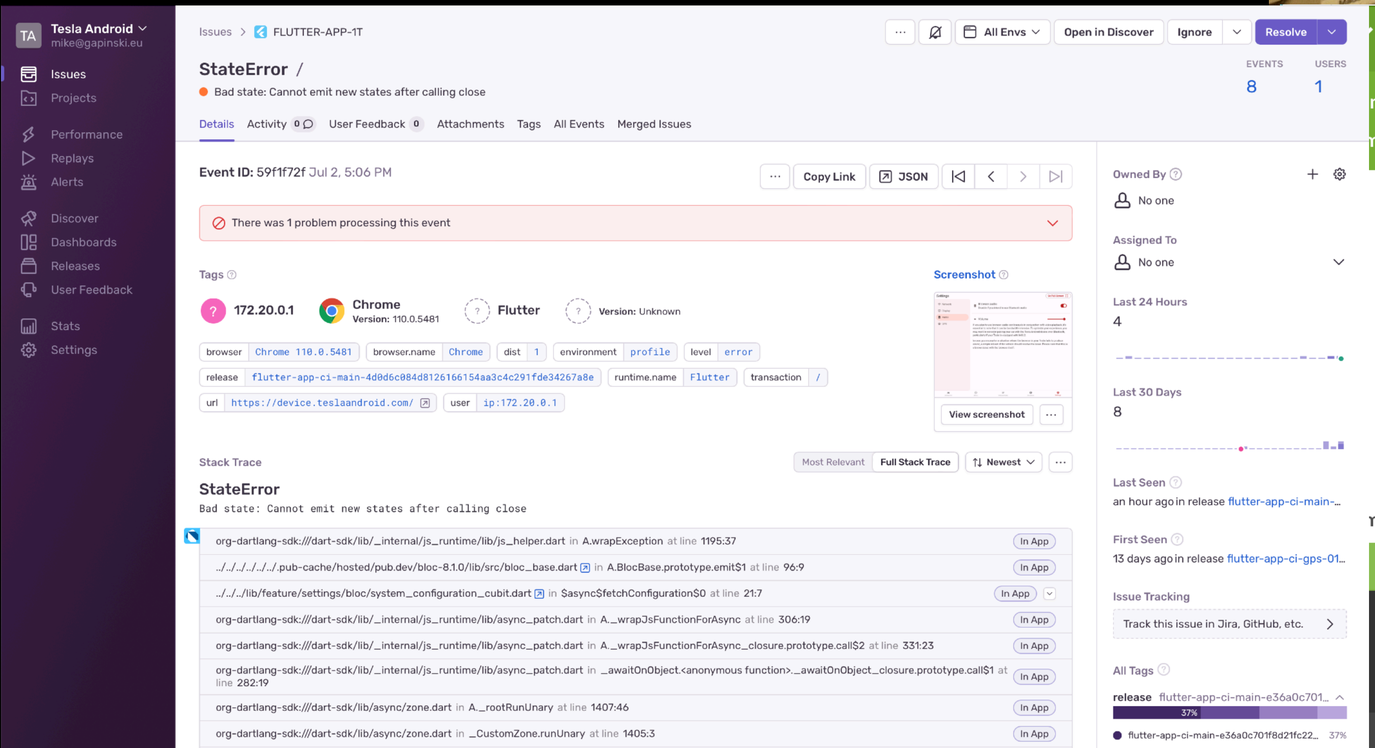
Development process

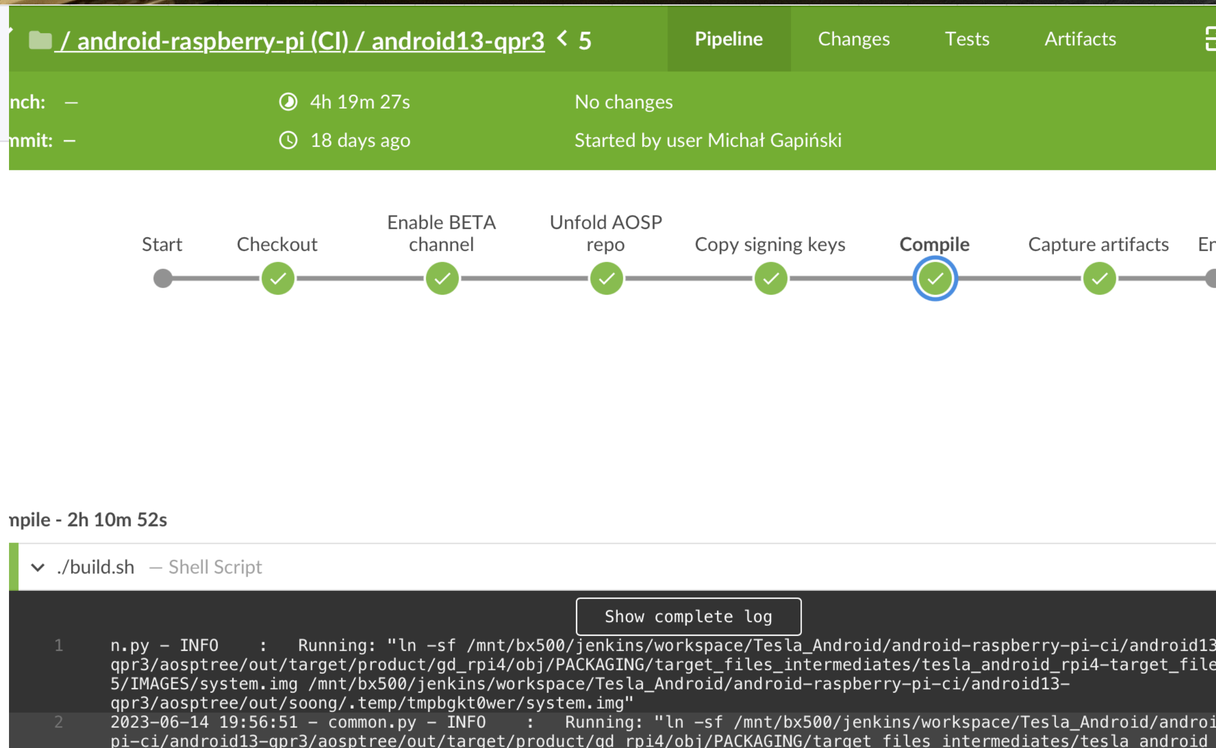
Jenkins is used for CI:

- 3 Linux nodes (Intel) for AOSP
- macOS node(ARM) for Kernel and everything that builds with meson
- Automated release flow
- Each push triggers a full build, OverlayFS is used to provide workspaces for incremental build

All testing and profiling is done on real hardware.

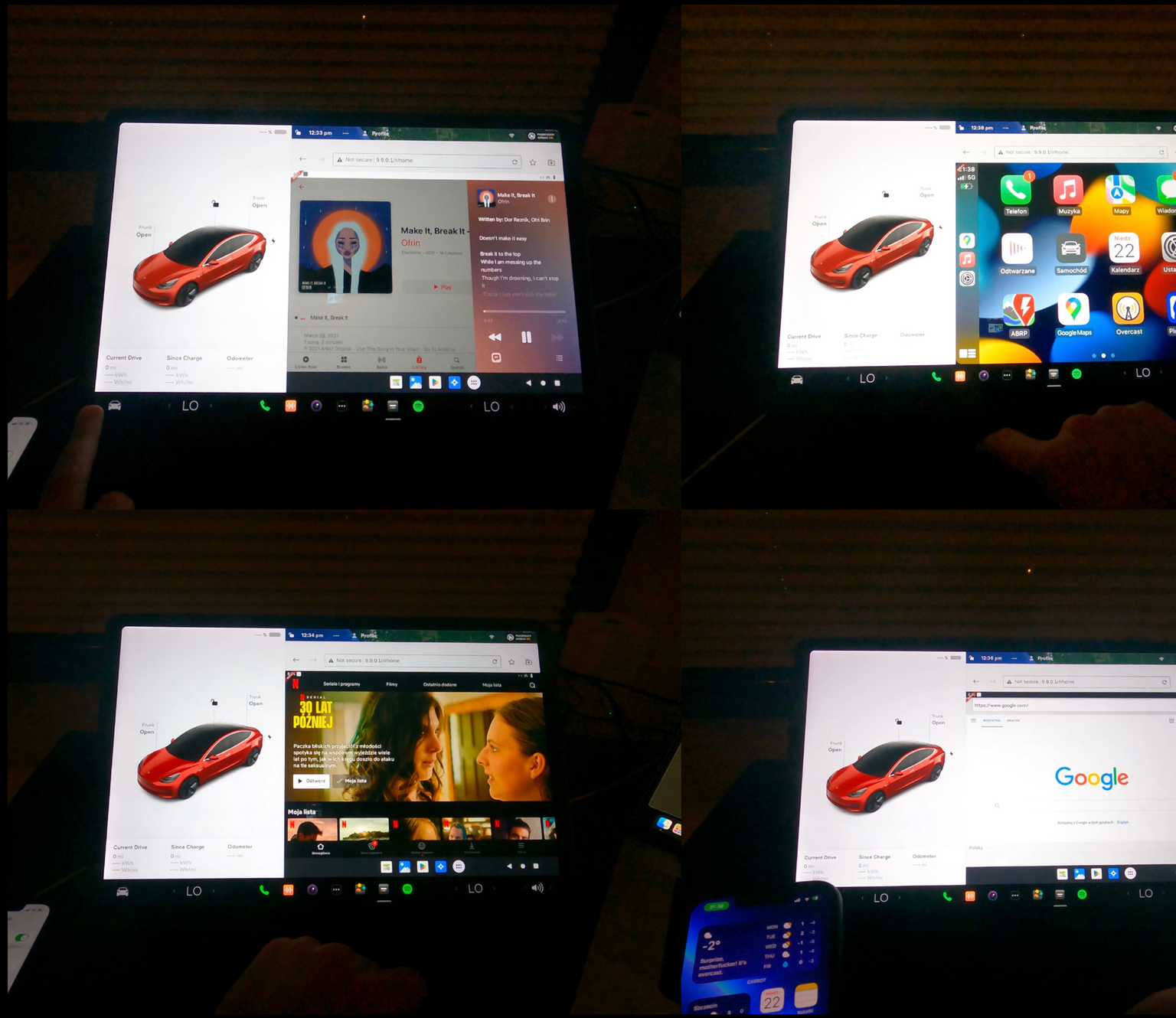
Sentry is used to collect logs/events.



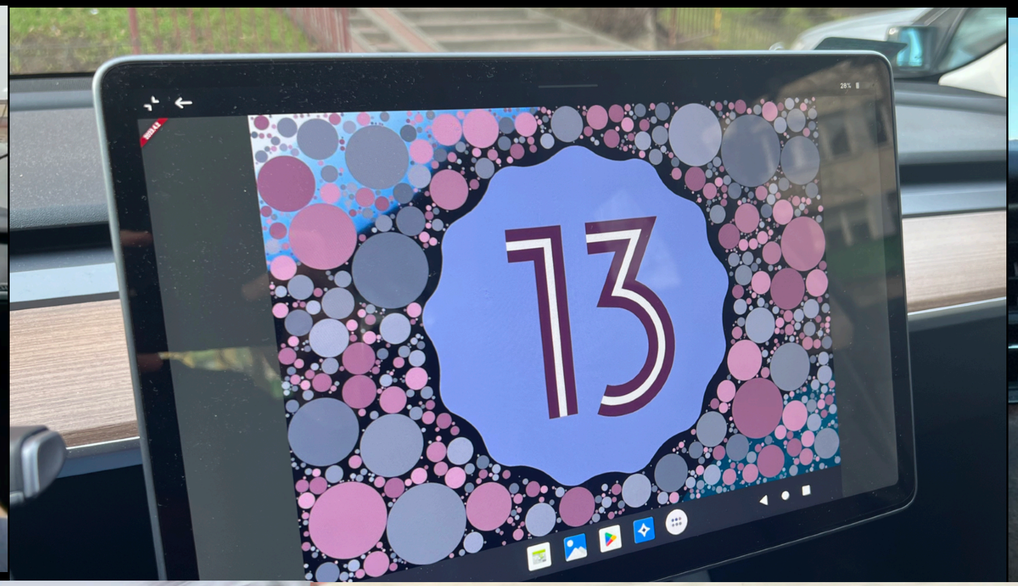


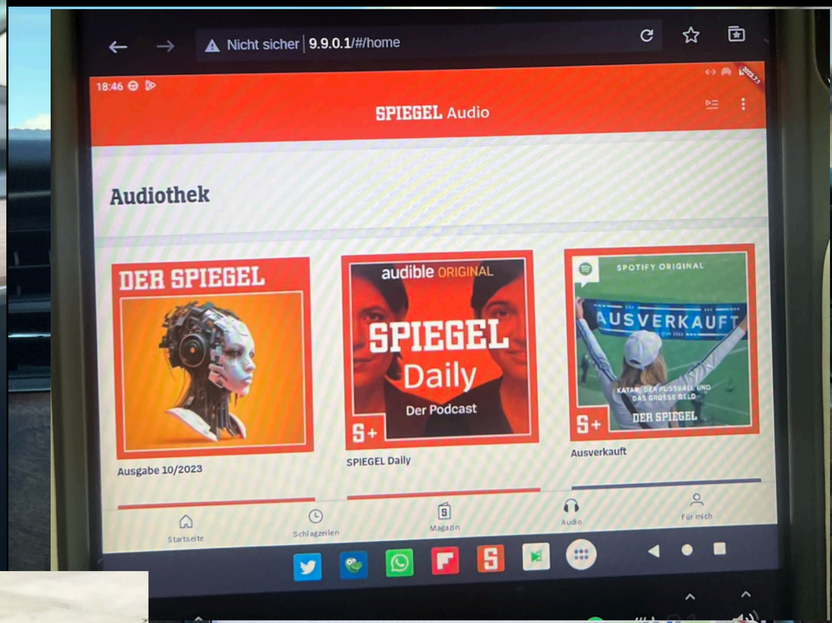
Results

- **250k+** unique website hits (teslaandroid.com)
- **5000+** DAU of users from every continent
- Featured by major outlet all over the world
- All project related costs(dev kits, CI machines, tools etc) were covered with donations
- **I've learned a lot**, this project never gets boring



Thank you for having me!





github.com/mikegapinski

github.com/tesla-android

twitter.com/mikegapinski

twitter.com/teslaandroid