DELTA – Střední škola informatiky a ekonomie, s.r.o.
Ke Kamenci 151, Pardubice

Mobilní aplikace pro konfiguraci a vizualizaci dat z NFC zařízení

Příjmení, jméno: Damián Michálek
Třída: 4.A
Studijní obor: 18-20-M/01 Informační technologie
Školní rok: 2025/2026

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení:	Damián Michálek
Pro školní rok:	2025/2026
Třída:	4.A
Obor:	18-20-M/01 Informační technologie
Téma práce:	Mobilní aplikace pro konfiguraci a vizualizaci dat z NFC zařízení
Vedoucí práce:	Lukáš Bier, MSc.

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cíl projektu:

Cílem projektu je vytvořit multiplatformní mobilní aplikaci, která bude kompilovatelná pro iOS i Android. Aplikace bude využívat technologii NFC integrovanou v mobilním zařízení pro komunikaci s externím zařízením vybaveným NFC pamětí.

Hlavním úkolem bude čtení a vizualizace dat. Po přiložení telefonu k zařízení aplikace načte data z jeho paměti, ověří identitu zařízení pomocí unikátního identifikátoru (UID) v paměti a následně je zobrazí. Naměřené hodnoty budou prezentovány formou grafu, zatímco stavové a konfigurační data (např. UID zařízení, stav baterie, interval měření) budou přehledně zobrazeny v informačním panelu.

Další klíčovou funkcionalitou bude možnost zápisu dat do NFC paměti zařízení. Uživatel bude pomocí aplikace konfigurovat dané externí zařízení, například měnit časový interval sběru dat, spouštět nové měření nebo ukončovat to probíhající.

Během vývoje se předpokládá průběžná komunikace s hardwarovou částí týmu.

Specifikace projektu:

1. **Analýza a návrh**
   - Analýza požadavků
   - Výběr vhodné vývojové platformy a technologií
   - Analýza a porozumění datové struktury NFC paměti
   - Hrubý návrh softwarového řešení a plánovaných softwarových komponent

2. **Vývoj aplikace**
    - NFC komunikace
        i. Implementace čtení dat z NFC paměti po přiblížení telefonu
        ii. Implementace zápisu konfiguračních dat a ovládacích příkazů (start/stop měření)
        iii. Validace dat pomocí jedinečného identifikátoru zařízení
    - Vizualizace dat
    - Správa konfigurace
    - Možnost rozšíření o export vizualizovaných dat (bonusový cíl)
3. **Testování a finalizace**
    - Funkční testování na cílovém hardware
    - Optimalizace výkonu a stability aplikace
    - Finalizace uživatelského rozhraní (UI/UX)

Požadované výstupy:

- Funkční mobilní aplikace
- Projektová dokumentace
- Neveřejný git repozitář (zdrojový kód podléhá obchodnímu tajemství). Přístup bude za účelem hodnocení umožněn vedoucímu práce a členům zkušební komise po předchozí domluvě.
- Prezentace o projektu a demonstrační video funkčnosti aplikace

Hodnocení:

- Kvalita, spolehlivost a funkčnost mobilní aplikace
- Kvalita uživatelského rozhraní (UI) a uživatelské přívětivosti (UX)
- Kreativita a efektivnost zvoleného technického řešení
- Úroveň prezentace a kvalita projektové dokumentace

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

- **Září**: Hrubá analýza požadavků, výběr vývojové platformy, návrh architektury aplikace a základní nástřel UI/UX
- **Říjen**: Nastavení vývojového prostředí, implementace základní komunikace s NFC (proof-of-concept pro čtení a zápis dat)
- **Listopad**: Vývoj UI pro zobrazení stavových informací (úvodní záložka) a konfiguračního panelu (záložka Konfigurace)
- **Prosinec**: Implementace vizualizace dat v grafech (záložka Data) a logiky spouštění/ukončování měření
- **Leden**: Úvodní testování a ladění aplikace na testovacím HW
- **Únor**: Komplexní testování, optimalizace výkonu a finalizace uživatelského rozhraní, začátek tvorby technické dokumentace
- **Březen**: Tvorba technické a uživatelské dokumentace, příprava finální prezentace a demonstračního videa

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

V Pardubicích dne _____ _____

Damián Michálek

Poděkování

Rád bych poděkoval svému vedoucímu práce, Lukáši Bierovi, MSc., za odborné vedení, flexibilitu a podporu během celého procesu vývoje projektu. Jeho enthusiasmus pro mě byl neocenitelný a motivující, což výrazně přispělo k úspěšnému dokončení této práce.

Anotace / Abstract

Česky

Práce se zabývá návrhem a vývojem multiplatformní mobilní aplikace pro komunikaci s embedded zařízením prostřednictvím technologie NFC. Cílové zařízení je vybaveno paměťovým čipem ST25DV64KC s duálním rozhraním, který umožňuje bezkontaktní čtení a zápis dat i bez vlastního napájení. Aplikace, vyvinutá v rámci frameworku React Native, umožňuje čtení naměřených hodnot z NFC paměti zařízení po přiložení mobilního telefonu, jejich vizualizaci formou interaktivních grafů a konfiguraci zařízení zápisem dat do NFC paměti. Implementace pokrývá nízkoúrovňovou komunikaci přes protokoly ISO 15693 (NfcV) a T4T (IsoDep) včetně ošetření přechodových chyb a validace identity zařízení pomocí unikátních magic bytes. Aplikace podporuje dvě verze datového formátu firmwaru díky registrovému vzoru umožňujícímu přidání dalších verzí bez zásahu do stávajícího kódu. Naměřená data jsou vizualizována ve čtyřech typech grafů s podporou filtrování podle časového okna, barevného podbarvení dle pásem indexu kvality ovzduší a automatického přerušení křivky při chybě senzoru. Výsledkem je funkční mobilní aplikace otestovaná na reálném hardwaru, splňující všechny primární cíle definované v zadání projektu.

English

This thesis presents the design and development of a cross-platform mobile application for communication with an embedded device via NFC technology. The target device is equipped with an ST25DV64KC dual-interface memory chip, enabling contactless data read and write operations without the need for the device's own power supply. The application, developed using the React Native framework, allows reading measured values from the device's NFC memory upon bringing a mobile phone into proximity, visualizing them as interactive charts, and configuring the device by writing data back to the NFC memory. The implementation covers low-level communication over ISO 15693 (NfcV) and T4T (IsoDep) protocols, including error handling and device identity validation via unique magic bytes. The application supports two firmware data format versions through a registry pattern that allows adding future versions without modifying existing code. Measured data are visualized in four chart types with support for time window filtering, color-coded air quality index bands, and automatic line breaks at sensor error values. The result is a functional mobile application, tested on real hardware, fulfilling all primary objectives defined in the project specification.

Klíčová slova / Keywords

Česky

1. *NFC* – Near Field Communication je bezdrátová komunikační technologie pro přenos dat na krátkou vzdálenost, často používaná v mobilních zařízeních pro různé aplikace, včetně plateb a přenosu dat
2. *embedded zařízení* – speciální počítačový systém navržený pro konkrétní úkoly, často s omezenými zdroji a integrovaný do většího systému
3. *multiplatformní vývoj* – proces vytváření softwaru, který je kompatibilní s více operačními systémy nebo platformami, například iOS a Android
4. *uživatelské rozhraní* – soubor vizuálních a interaktivních prvků, které umožňují uživateli ovládat aplikaci
5. *vizualizace dat* – proces převodu dat do grafické formy, která usnadňuje jejich pochopení a analýzu
6. *mobilní aplikace* – aplikace určená pro mobilní zařízení, zejména pro platformy iOS a Android
7. *iOS/Android* – nejrozšířenější operační systémy pro mobilní zařízení, vyvinuté společnostmi Apple a Google
8. *komunikační protokol* – soubor pravidel a standardů určujících způsob přenosu a formátování dat

English

1. *NFC* – Near Field Communication is a wireless communication technology for short-range data transfer, commonly used in mobile devices for various applications, including payments and data exchange
2. *embedded device* – a specialized computer system designed for specific tasks, often with limited resources and integrated into a larger system
3. *cross-platform development* – the process of creating software that is compatible with multiple operating systems or platforms, such as iOS and Android
4. *user interface* – a collection of visual and interactive elements that allow users to control an application
5. *data visualization* – the process of converting data into a graphical form that facilitates understanding and analysis
6. *mobile application* – an application designed for mobile devices, particularly for iOS and Android platforms
7. *iOS/Android* – the most widely used operating systems for mobile devices, developed by Apple and Google
8. *communication protocol* – a set of rules and standards that define how data is transmitted and formatted

Obsah

1 Úvod

Tato maturitní práce se zabývá návrhem a vývojem multiplatformní mobilní aplikace pro komunikaci a ovládání embedded zařízení pomocí NFC. Cílem této kapitoly je představit kontext a motivaci pro vznik tohoto projektu, přiblížit řešený problém, definovat hlavní cíle práce a poskytnout přehled současného stavu trhu a existujících řešení a zdůvodnění potřeby vytvořit vlastní řešení na míru.

1.1 Vymezení problému

V rámci projektu je řešen vývoj dedikovaného softwarového řešení na míru pro cílové hardwarové zařízení, které je navrhováno od úplného základu (tzn. „*greenfield* projekt“). Pro zajištění kompatibility a funkčnosti je nezbytná úzká a průběžná spolupráce s hardwarovou částí týmu, zejména pro sjednocení datové struktury a definování komunikačního protokolu.

1.2 Cíle práce

Cílem projektu je vytvořit mobilní aplikaci pro čtení a vizualizaci dat z NFC zařízení, která bude kompilovatelná na iOS i Android. Aplikace bude schopna komunikovat s NFC pamětí externího zařízení, načítat naměřená data a zobrazovat je uživateli formou grafů a informačních panelů. Dále bude umožňovat konfiguraci zařízení prostřednictvím zápisu dat do jeho NFC paměti.

Projekt je třeba zasadit do širšího kontextu. V době zahájení vývoje bylo cílové zařízení zamýšleno primárně pro osobní použití v produktové řadě *HomePass*, zaměřené na měření kvality okolního prostředí (teplota, vlhkost, CO_2, AQI). V průběhu vývoje se však ukázalo, že první reálná poptávka přichází z průmyslu, konkrétně s požadavkem na měření provozních hodnot jako podklad pro posuzování reklamací. To vedlo k upřednostnění druhé plánované produktové řady *DevicePass*, zaměřené na monitorování mechanického namáhání zařízení, kde hlavní roli hraje právě měření G-senzorem.

Tato mobilní aplikace proto představuje živý projekt, který se vyvíjí v přímé vazbě na reálné průmyslové požadavky. Jeho rozvoj odevzdáním této maturitní práce nekončí.

1.3 Přehled současného stavu

Pro komunikaci s embedded zařízeními se v současné době nejčastěji využívají kabelová připojení (USB, sériový port) nebo bezdrátové technologie (Bluetooth, Wi-Fi). Přestože jsou tyto metody široce využívány, nesou s sebou určité nevýhody, jako je potřeba párování a neustálého napájení komunikačního modulu, závislost na síti nebo fyzické připojení.

Využití technologie NFC pro komunikaci přináší zásadní výhody, zejména pro zařízení s omezenými zdroji. Umožňuje totiž bezkontaktní přenos dat na krátkou vzdálenost pouhým přiložením mobilního telefonu, a to dokonce i v situaci, kdy je baterie cílového zařízení zcela vybitá. To je možné díky použití paměťových čipů s duálním rozhraním (tzv. *Dynamic NFC tag*). Z pohledu mobilní aplikace se takové zařízení chová jako standardní pasivní NFC tag, kde pro komunikaci i samotný zápis do vnitřní EEPROM paměti nepotřebuje vlastní napájení, protože potřebnou energii získává pomocí indukce z elektromagnetického pole generovaného čtecím zařízením (mobilním telefonem) [1].

Na trhu existují různé aplikace pro čtení NFC tagů (např. *NFC Tools* [2] nebo *ST25 NFC tap* [3]), tyto nástroje však pouze přečtou surová binární data z NFC paměti, která většinou zobrazí jako hexadecimální řetězec. Pro specifické použití s vyvíjeným zařízením, které má vlastní datovou strukturu a vyžaduje nejen čtení, ale i zpětný zápis na specifické adresy v paměti pro konfiguraci, není možné tyto univerzální nástroje použít. Navíc neobsahují žádné z požadovaných funkcí pro zpracování (přepočet naměřených hodnot, detekce stavu zařízení) a vizualizaci dat (grafy, informační panely) přizpůsobenou pro dané cílové zařízení. Z tohoto důvodu vznikla potřeba vyvinout vlastní mobilní aplikaci, která zajistí nejen správné dekódování (parsování) binárních datových paketů, ale také poskytne uživatelsky přívětivé rozhraní pro vizualizaci těchto dat a následnou konfiguraci zařízení.

2 Teoretická část

V této kapitole se zaměříme na teoretické základy fungování technologie NFC, její principy a specifikace a také na zdůvodnění výběru vývojové platformy pro tento projekt.

2.1 Technologie NFC

NFC (*Near Field Communication*) je bezdrátová komunikační technologie pracující na celosvětově dostupné nelicencované frekvenci 13.56 MHz. Umožňuje výměnu dat na velmi krátkou vzdálenost, typicky do 2 cm a ke spojení dochází téměř okamžitě po přiložení zařízení bez nutnosti předchozího párování.

Z pohledu hardwarové architektury se NFC zařízení rozdělují na aktivní a pasivní. Aktivní zařízení (např. mobilní telefon) má svůj vlastní zdroj energie a vytváří elektromagnetické pole. Pasivní zařízení (např. NFC tag) nemá vlastní napájení a pro komunikaci využívá energii z pole vytvářeného aktivním zařízením pomocí indukce. To umožňuje pasivním NFC tagům fungovat i bez baterie [4].
Existují tři hlavní režimy komunikace:

- **Reader/Writer mode (Čtení a zápis)** – aktivní zařízení (čtečka) čte nebo zapisuje data do pasivního zařízení (tagu)
- **Peer-to-Peer mode (P2P)** – dvě aktivní zařízení komunikují mezi sebou, umožňuje obousměrný přenos dat
- **Card Emulation mode (Emulace karty)** – aktivní zařízení se chová jako pasivní NFC tag, umožňuje například bezkontaktní platby

2.2 Volba vývojové platformy

Pro implementaci byl zvolen framework React Native. Hlavním důvodem pro tento výběr byla podpora multiplatformního vývoje ze společné kódové základny (tzv. *codebase*). Tento přístup výrazně zjednodušuje vývoj a údržbu aplikace, protože umožňuje sdílet většinu kódu mezi iOS a Android verzí namísto nutnosti vytvářet a udržovat dvě samostatné nativní aplikace.

Dalším důvodem výběru byla existence robustních knihoven pro práci s NFC v React Native, které by jinak musely být implementovány od nuly pro každou platformu, protože nativní API pro NFC se mezi iOS [5] a Android [6] výrazně liší.

3 Návrh a implementace aplikace

Tato kapitola popisuje návrh a implementaci mobilní aplikace, nejprve je představena datová struktura NFC paměti cílového zařízení, formát datových paketů, kódování měřených hodnot a rozdíly mezi verzemi firmwaru. Následně popisuje navrženou architekturu aplikace a její komponenty. Kapitola dále pokrývá implementaci NFC komunikace, vizualizaci naměřených dat a správu konfigurace zařízení.

3.1 Datová struktura NFC paměti

Komunikace mezi mobilní aplikací a zařízením probíhá skrz NFC paměť. Zařízení do ní průběžně zapisuje naměřená data, mobilní aplikace je čte a v opačném směru zapisuje konfiguraci a ovládací příkazy. Tato podkapitola popisuje strukturu a formát dat uložených v NFC paměti.

3.1.1 Typy měřených hodnot a jejich kódování

Zařízení měří několik hodnot z reálného světa, s tím, že každá je v NFC paměti uložena jako jednobajtová hodnota H v rozsahu 0–254. Pro úsporu paměti je každá hodnota přepočítávána s pevným rozlišením a případně posunuta, aby prokryla rozsah měřených hodnot (např. teplota). Přepočet z uložené hodnoty H na reálnou hodnotu provádí mobilní aplikace při parsování dat.

Chybový indikátor

Hodnota $H = 255$ (`0xFF`) je u všech veličin rezervována jako indikátor chyby senzoru a nesmí být převedena jako platný naměřený údaj.

Teplota

Teplota je ukládána s rozlišením 0,5 °C. Při kódování se hodnota posouvá o 22 °C, aby bylo možné reprezentovat i záporné teploty:

$$H = (t + 22) \cdot 2 \qquad \Longleftrightarrow \qquad t = \frac{H}{2} - 22 \quad [^\circ\mathrm{C}]$$

Platný rozsah: $-22{,}0\,^\circ\mathrm{C}$ $(H = 0)$ až $105{,}0\,^\circ\mathrm{C}$ $(H = 254)$.

Relativní vlhkost

Relativní vlhkost je ukládána s rozlišením 0,5 % :

$$H = RH \cdot 2 \qquad \Longleftrightarrow \qquad RH = \frac{H}{2} \quad [\%]$$

Platný rozsah: 0 % $(H = 0)$ až 100 % $(H = 200)$.

Napětí baterie

Napětí baterie je ukládáno s rozlišením 0,025 V:

$$H = U \cdot 40 \qquad \Longleftrightarrow \qquad U = \frac{H}{40} \quad [\mathrm{V}]$$

Platný rozsah: 0 V $(H = 0)$ až 6,35 V $(H = 254)$.

Koncentrace CO_2

Koncentrace oxidu uhličitého je ukládána s rozlišením 16 ppm:

$$H = \frac{CO_2}{16} \qquad \Longleftrightarrow \qquad CO_2 = H \cdot 16 \quad [\mathrm{ppm}]$$

Platný rozsah: 0 ppm $(H = 0)$ až 4064 ppm $(H = 254)$.

Index kvality ovzduší (AQI)

Index kvality ovzduší je ukládán přímo bez přepočtu, s rozlišením 1:

$$AQI = H$$

Platný rozsah: 0 $(H = 0)$ až 20 $(H = 20)$.

Koncentrace NO_x

Koncentrace oxidů dusíku je ukládána s rozlišením 2 ppm:

$$H = \frac{NO_x}{2} \qquad \Longleftrightarrow \qquad NO_x = H \cdot 2 \quad [\mathrm{ppm}]$$

Platný rozsah: 0 ppm $(H = 0)$ až 508 ppm $(H = 254)$.

Akcelerometr (G senzor)

Akcelerometr měří maximální zrychlení společně pro všechny osy a ukládá je jako jednobajtovou hodnotu H s rozlišením 0,1 g:

$$H = g \cdot 10 \qquad \Longleftrightarrow \qquad g = \frac{H}{10} \quad [\mathrm{g}]$$

Platný rozsah: 0 g $(H = 0)$ až 25,4 g $(H = 254)$.

Stávající datový formát v1.01 ukládá prostou hodnotu maximálního vektorového zrychlení v rámci jednoho záznamu. V rámci probíhajícího vývoje produktové řady *DevicePass* je plánováno rozšíření záznamu G-senzoru o podrobnější charakteristiky mechanického namáhání: maximální amplitudu (*Peak*), efektivní hodnotu zrychlení (*RMS*) a dominantní frekvenci vibrace. Tyto hodnoty budou sloužit k ověření souladu se zkušebními normami EN 60068-2-6 [7] (sinusové vibrace) a EN 60068-2-64 [8] (širokopásmové náhodné vibrace) a umožní posoudit, zda zařízení nebylo vystaveno vibracím překračujícím přípustné limity, což je klíčové pro předcházení neoprávněným reklamacím.

Kompletní přehled všech měřených veličin shrnuje tabulka 1.

Tabulka 1: Přehled měřených veličin, jejich kódování a platných rozsahů

Veličina	**Rozlišení**	**Minimum**	**Maximum**	**Přepočet**
Teplota	0,5 °C	−22,0 °C	105,0 °C	$t = H/2 - 22$
Rel. vlhkost	0,5 %	0 %	100 %	$RH = H/2$
Napětí baterie	0,025 V	0 V	6,35 V	$U = H/40$
CO_2	16 ppm	0 ppm	4064 ppm	$CO_2 = H \cdot 16$
AQI	1	0	20	$AQI = H$
NO_x	2 ppm	0 ppm	508 ppm	$NO_x = H \cdot 2$
Zrychlení	0,1 g	0 g	25,4 g	$g = H/10$

3.1.2 Formát datových paketů

Paměť NFC čipu má celkovou kapacitu 8192 bajtů paměti. Tato paměť je rozdělena do dvou logických oblastí, konfigurační hlavička a blok naměřených dat. Konfigurační hlavička vždy začíná na adrese 0 a její délka závisí na verzi firmwaru zařízení. Blok naměřených dat navazuje za koncem hlavičky a zabírá zbývající část paměti až do posledního platného záznamu.

Tabulka 2: Rozložení paměti NFC tagu

Oblast	**Začátek**	**Konec**	**Velikost**
Konfigurace (v1.00)	0	37	38 B
Konfigurace (v1.01)	0	59	60 B
Naměřená data (v1.00)	38	proměnný	$recordCount \times activeSensors.length$ B
Naměřená data (v1.01)	60	proměnný	$recordCount \times activeSensors.length$ B

Délka bloku naměřených dat

Každý záznam v bloku naměřených dat odpovídá jednomu okamžiku měření a obsahuje právě tolik bajtů, kolik je aktivních senzorů. Celková délka tohoto bloku je vypočítána takto:

$$\text{délka} = \textit{recordCount} \times \textit{activeSensors.length}$$

kde *recordCount* je počet platných záznamů uložených v konfigurační hlavičce a *activeSensors.length* je počet aktivních senzorů zakódovaných v bitové masce aktivních senzorů.

Struktura bloku naměřených dat

Hodnoty jednotlivých senzorů jsou v paměti uloženy v prokládané (*interleaved*) struktuře. Záznamy nejsou seskupeny po senzorech, ale po časových okamžicích měření, kde každý záznam obsahuje hodnoty všech aktivních senzorů v daném pořadí, následované hodnotami téhož pořadí pro další časový okamžik.

Schematicky:

$$[\, s_1 r_1,\ s_2 r_1,\ \ldots,\ s_n r_1,\ s_1 r_2,\ s_2 r_2,\ \ldots,\ s_n r_2,\ \ldots]$$

kde $s_i r_j$ označuje hodnotu i-tého senzoru v j-tém záznamu. Při parsování aplikace tuto strukturu rozbalí (*de-interleaving*) do samostatných polí pro každý senzor. Index konkrétní hodnoty v lineárním poli bajtů je dán vztahem:

$$\text{index} = j \times \textit{activeSensors.length} + i$$

kde j je index záznamu (od 0) a i je index senzoru v poli aktivních senzorů (od 0).

Rozdíly mezi verzemi v1.00 a v1.01

Verze v1.01 má oproti v1.00 rozšířenou konfigurační hlavičku o podporu G senzoru (akcelerometru). Tato změna si vyžádala přesunutí některých polí v paměťové mapě. Tabulka 3 shrnuje klíčové rozdíly mezi oběma verzemi.

Tabulka 3: Porovnání verzí v1.00 a v1.01

Parametr	**v1.00**	**v1.01**
Velikost konfigurační hlavičky	38 B	60 B
Začátek bloku naměřených dat	38. index	60. index
Adresa bitové masky aktivních senzorů	26	28
Adresa device control byte	27	29
Podpora G senzoru (akcelerometr)	ne	ano (adresy 26–27)

Verze firmwaru je zakódována na adrese 2 v konfigurační hlavičce jako jeden bajt, jehož horní nibble obsahuje hlavní číslo verze a dolní nibble vedlejší číslo verze:

$$version = (major \ll 4) \mid minor$$

Například bajt `0x10` odpovídá verzi v1.00 a bajt `0x11` verzi v1.01. Na základě tohoto bajtu aplikace při čtení automaticky zvolí odpovídající parser a při zápisu odpovídající writer, jak je popsáno v podkapitole 3.4.5.

3.2 Architektura systému

Aplikace je navržena jako vícevrstvá architektura (*layered architecture*), kde každá vrstva má jasně vymezenou odpovědnost a komunikuje pouze se sousedními vrstvami. Toto oddělení odpovědnosti (*separation of concerns*) usnadňuje testování jednotlivých komponent, umožňuje transparentní záměnu implementace na nižších vrstvách (například přechod mezi protokoly NfcV a IsoDep) bez zásahu do vyšších vrstev a zjednodušuje budoucí rozšíření aplikace o podporu nových verzí firmwaru.

Tabulka 4: Přehled vrstev aplikace

Vrstva	**Soubory**	**Odpovědnost**
Protokolová	`nfc.service.ts`	Nízkoúrovňové NFC příkazy (ISO 15693, T4T APDU), správa session
Manažerská	`nfc.manager.ts`	Orchestrace session, dvoufázové čtení, cross-version validace
Datová	`tag-memory.parser.ts`, `tag-memory.writer.ts`	Serializace a deserializace dat, dispatching dle verze firmwaru
Verzovaná	`v1_00/v1_01.{parser,writer}.ts`	Konkrétní implementace parsování a zápisu pro každou verzi
Úložná	`storage.ts`	Persistentní lokální úložiště, správa *snapshotů*
Prezentační	`screens/`, `hooks/`, `contexts/`	Uživatelské rozhraní, stavová logika, React Context

Protokolová vrstva (`nfc.service.ts`) je od manažerské (`nfc.manager.ts`) záměrně oddělena. Servisní vrstva poskytuje jednotné rozhraní `readBytes` a `writeBytes`, které interně přepíná mezi nízkoúrovňovými implementacemi pro NfcV a IsoDep. Manažerská vrstva tak pracuje vždy se stejným rozhraním bez ohledu na to, který protokol je v danou chvíli aktivní.

3.2.1 Komponenty aplikace

Následující sekce shrnuje jednotlivé komponenty aplikace, jejich roli a hlavní exportované funkce či rozhraní.

`nfc.service.ts`

Nízkoúrovňová protokolová vrstva. Implementuje ISO 15693 příkazy (*Read/Write Single Block*, *Read Multiple Blocks*) a T4T APDU příkazy (`SELECT`, `READ BINARY`, `UPDATE BINARY`). Přepíná mezi NfcV a IsoDep podle aktivního protokolu.

`nfc.manager.ts`

Manažerská vrstva. Poskytuje session wrapper `runNfcSession`, implementuje dvoufázové čtení konfigurace (probe + full read), cross-version ochranu při zápisu a exportuje funkce (`readSensorData`, `writeConfigAndStartMeasurement`, `stopMeasurement`).

`tag-memory.parser.ts`

Centrální dispatcher parsování. Čte version byte z adresy 2, vybere odpovídající parser z registru `VERSION_PARSERS` a deleguje parsování. Exportuje rozhraní `VersionParser` implementované verzovanými parsery.

`tag-memory.writer.ts`

Centrální dispatcher zápisu. Analogicky k parseru vybere writer dle version byte z registru `VERSION_WRITERS`.
Exportuje rozhraní `VersionWriter` a funkci `buildConfigWritePayload`.

`storage.ts`

Vrstva perzistentního úložiště nad knihovnou MMKV. Spravuje *snapshoty* dat (`saveTagSnapshot`, `loadTagSnapshot`), konfiguraci (`saveConfig`, `loadConfig`) a *display names* zařízení.
Klíče jsou strukturovány s prefixy `config_`, `sensordata_` a `displayname_`.

`NfcProvider.tsx`

React Context sledující stav NFC adaptéru v reálném čase. Definuje výčet `NfcStatus` (`CHECKING`, `ENABLED`, `DISABLED`, `NOT_SUPPORTED`) a na Androidu naslouchá systémovým událostem změny stavu NFC.

`useLatestTagData.ts`

Hook prezentační vrstvy. Načítá z úložiště konfiguraci a senzorová data posledního přečteného tagu, automaticky se obnovuje při přechodu na obrazovku pomocí *hooku* `useFocusEffect`.

`useGraphData.ts`

Hook obrazovky Data. Načítá všechny uložené snímky, slučuje duplicity stejného měření dle klíče (*UID* + *measuringStartTime*) a vrací seřazený seznam pro výběr měření.

`NfcState.tsx`

Univerzální stavová komponenta NFC. Zobrazuje příslušné UI dle kombinace stavu NFC adaptéru a dostupnosti dat (čekání na tag, načítání, chyba, úspěch). Při aktivní obrazovce automaticky spouští čtení tagu a při přepnutí záložky session korektně zruší.

3.2.2 Interakce mezi komponentami

Pro ilustraci spolupráce jednotlivých vrstev jsou níže popsány tři hlavní toky operací v aplikaci.

Tok čtení dat (obrazovka Home)

1. Komponenta `NfcState` detekuje, že je NFC aktivní, a automaticky spustí `handleAutoRead`.
2. Je volána funkce `nfcManager.readSensorData`, která otevře NFC session přes `runNfcSession`.
3. Manažer provede dvoufázové čtení: nejprve načte 3 bajty (magic bytes + version byte), poté celý konfigurační blok odpovídající délky.
4. Na základě version byte dispatcher `tag-memory.parser` vybere správný verzovaný parser (`v1_00.parser` nebo `v1_01.parser`) a zavolá `parseConfigData`.
5. Manažer vypočítá délku senzorových dat a načte je jako jeden spojitý blok.
6. Funkce `parseSensorData` „rozloží“ (*de-interleaves*) data do samostatných polí pro každý senzor.
7. Výsledek je uložen do MMKV přes `storage.saveTagSnapshot` a hook `useLatestTagData` zajistí překreslení UI.

Tok zápisu konfigurace (obrazovka Config)

1. Uživatel stiskne tlačítko „Nové měření“, čímž se zavolá funkce `nfcManager.writeConfigAndStartMeasurement`.
2. Manažer nejprve přečte aktuální version byte z tagu a ověří, že odpovídá verzi poslední načtené konfigurace (cross-version ochrana).
3. Dispatcher `tag-memory.writer` vybere odpovídající verzovaný writer a sestaví bajtové pole pro zápis.
4. Servisní vrstva zapíše konfigurační bajty na adresy 8–26 (v1.00) nebo 8–28 (v1.01).
5. Následně je zapsán device control byte hodnotou `0x02` na příslušnou adresu, čímž zařízení zahájí nové měření a samo smaže stará data.

Životní cyklus NFC session

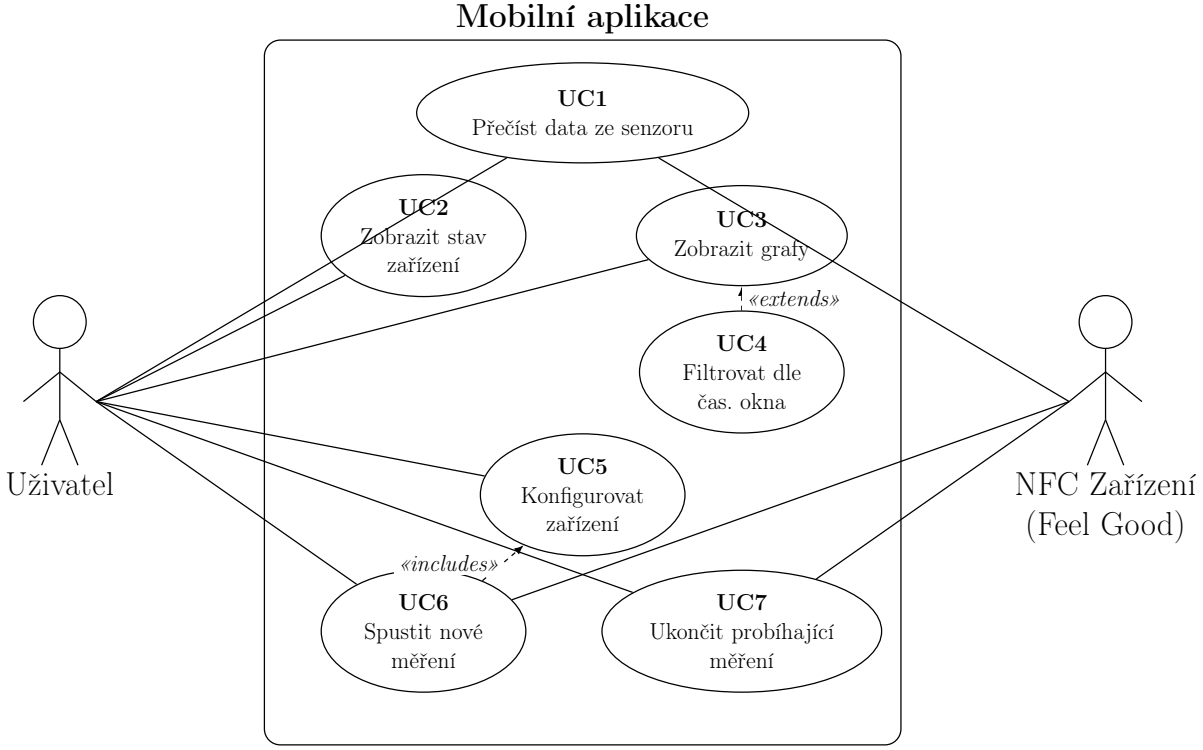
1. `requestSession` se pokusí použít NfcV technologii; při neúspěchu zkusí IsoDep jako záložní protokol.
2. `initializeTagConnection` – pouze pro IsoDep – provede výběr NDEF aplikace a souboru pomocí APDU příkazů `SELECT`.
3. Jsou provedeny požadované operace čtení nebo zápisu.
4. `releaseSession` vždy uvolní session v bloku `finally`, i při výjimce.

Pokud uživatel přepne na jinou záložku aplikace v průběhu čekání na přiložení tagu, komponenta `NfcState` detekuje ztrátu fokusu obrazovky a zavolá funkci `NfcManager.cancelTechnologyRequest`. Zrušení session je odlišeno od skutečné chyby pomocí příznaku `isCancelledRef`, takže uživatel neuvidí chybové hlášení při záměrné navigaci pryč.

3.3 UML diagramy

Tato podkapitola obsahuje Use Case diagram zachycující interakce uživatele se systémem a dva diagramy tříd dokumentující klíčové architektonické vzory aplikace – vzor Strategy pro parsování a zápis verzí firmwaru a závislosti centrálního správce NFC operací.

3.3.1 Use Case diagram



Obrázek 1: Use Case diagram mobilní aplikace

Komentáře k Use Case diagramu

Předpoklady: NFC adaptér je dostupný a zapnutý, telefon je přiložen k tagu.

Diagram zachycuje dva aktéry.

Uživatel (primární aktér, vlevo) iniciuje všechny případy užití – přikládá telefon k tagu, prohlíží dashboard i grafy a konfiguruje zařízení.

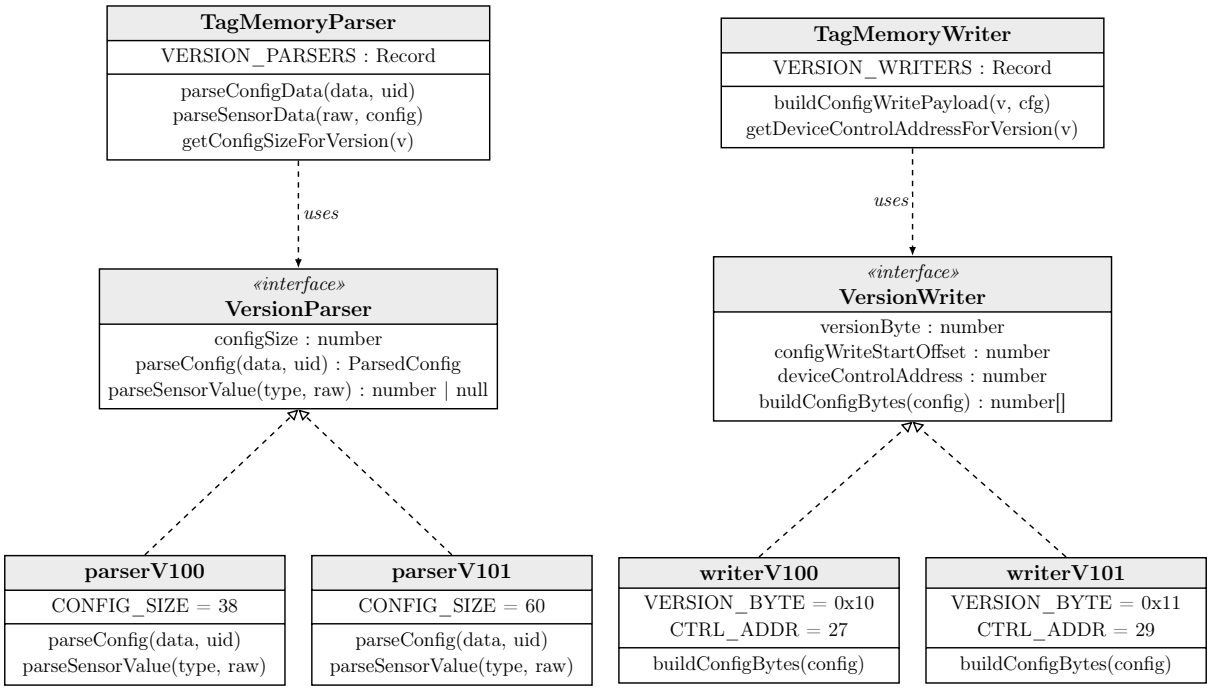
NFC Zařízení Feel Good je sekundárním aktérem: při čtení dat (UC1) poskytuje naměřená data z NFC paměti, při spuštění (UC6) a ukončení měření (UC7) přijímá řídicí bajt zapsaný aplikací.

Relace *«extends»* vyjadřuje, že filtrování dle časového okna (UC4) je volitelné rozšíření zobrazení grafů (UC3) – grafy lze zobrazit i bez filtrování. Relace *«includes»* vyjadřuje, že spuštění nového měření (UC6) vždy zahrnuje zápis konfigurace na tag (UC5); v implementaci jsou tyto dvě operace sloučeny v jedné metodě `writeConfigAndStartMeasurement`.

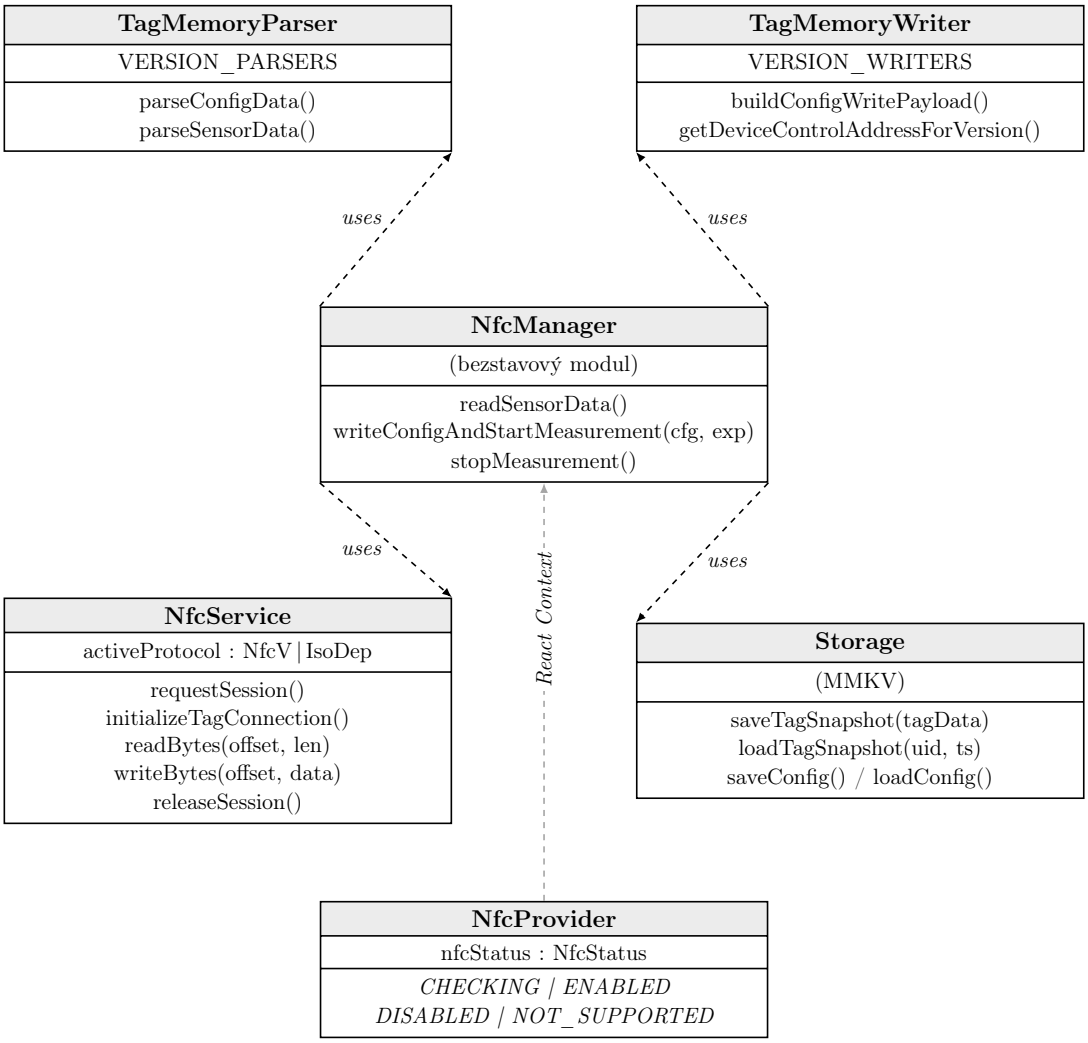
Alternativní toky nejsou v diagramu zachyceny explicitně. Při nedostupném nebo vypnutém NFC zobrazuje komponenta `NfcProvider` příslušný stav (`NOT_SUPPORTED`, `DISABLED`) a akce UC1, UC6 a UC7 nelze iniciovat. Při neplatném tagu (magic bytes mismatch nebo nepodporovaná verze firmwaru) selže validace v `readAndValidateConfig` a uživateli se zobrazí chybový modal; po 3 sekundách se stav vrátí do `idle` a lze pokus opakovat (podkapitola 3.4.3, 3.4.4).

3.3.2 Diagram tříd

Třídní diagram je rozdělen do dvou částí. Obrázek 2 zachycuje vzor Strategy použitý pro podporu více verzí firmwaru. Obrázek 3 zobrazuje závislosti třídy `NfcManager`, která orchestruje veškeré NFC operace.



Obrázek 2: Diagram tříd – vzor Strategy pro parsování a zápis podle verze firmwaru



Obrázek 3: Diagram tříd – závislosti třídy `NfcManager`

Vzor Strategy – parsování a zápis dle verze firmwaru

Diagram 2 zobrazuje dvě paralelní hierarchie. Rozhraní `VersionParser` definuje kontrakt pro parsování s metodami `parseConfig` a `parseSensorValue`. Konkrétní implementace `parserV100` a `parserV101` se liší velikostí konfigurační hlavičky (38 vs. 60 B) a v případě v1.01 navíc přidávají podporu G-senzoru. Dispatcher `TagMemoryParser` uchovává implementace v registru `VERSION_PARSERS` klíčovaném verzovacím bajtem; při každém volání `parseConfigData` vybere odpovídající implementaci automaticky. Rozhraní `VersionWriter` a dispatcher `TagMemoryWriter` tvoří podobnou strukturu pro serializaci konfigurace. Díky registrovému vzoru přidání podpory nové verze firmwaru – například v1.02 – vyžaduje pouze vytvoření souborů `v1_02.parser.ts` a `v1_02.writer.ts` a registraci klíče `0x12` do příslušných slovníků a žádná ze stávajících vrstev se nemění.

Závislosti třídy `NfcManager`

`NfcManager` orchestruje veškeré NFC operace a závisí na čtyřech komponentách zobrazených v diagramu 3: `NfcService` poskytuje nízkoúrovňový přístup k tagu přes NfcV nebo IsoDep, `TagMemoryParser` a `TagMemoryWriter` zajišťují deserializaci a serializaci dat s dispatchingem dle verze firmwaru a `Storage` obstarává perzistentní uložení snapshotů do MMKV. Třída `NfcProvider` stojí mimo přímé závislosti `NfcManageru` a zpřístupňuje stav NFC adaptéru zbytku aplikace prostřednictvím React Context; výčet `NfcStatus` nabývá hodnot `CHECKING`, `ENABLED`, `DISABLED` a `NOT_SUPPORTED`.

3.4 Implementace NFC komunikace

Následující podkapitoly popisují implementaci čtení a zápisu dat na úrovni protokolové a manažerské vrstvy. Pro přehled o zařazení jednotlivých komponent do architektury viz tabulka 4.

3.4.1 Čtení dat z NFC paměti

Aplikace při čtení dat z paměti cílového zařízení používá dvoufázový přístup, díky kterému se zbytečně nečtou tagy s jinou než očekávanou strukturou dat. V první fázi dojde k načtení prvních tří bajtů v paměti. Tyto bajty obsahují dvě hodnoty, a to unikátní identifikátor struktury (*magic bytes*), který slouží jako základní ověření správného typu zařízení, a dále indikátor použité hardwarové verze (*version byte*). Samotnému dekódování a procesu ověření těchto hodnot se podrobněji věnuje podkapitola 3.4.3.

Ve druhé fázi aplikace dekóduje *version byte* a podle něj určí délku a mapu konfiguračního bloku, který poté celý načte do lokální paměti. Díky parametrům získaným z konfigurace lze vypočítat délku datového pole s naměřenými hodnotami. Celková délka senzorových (naměřených) dat v bajtech je dána součinem uloženého počtu platných záznamů (`config.recordCount`) a počtu aktivních senzorů (`config.activeSensors.length`). Následné čtení tohoto spojitého bloku naměřených dat začíná od vypočítaného indexu `sensorDataStartOffset`.

Vyčítání bloků paměti z integrovaného obvodu je implementováno odlišně v závislosti na aktuálně zvoleném protokolu komunikace. Při použití nízkoúrovňového protokolu NfcV zajišťuje funkce `readBytesNfcV` překlad bajtového offsetu na indexy konkrétních bloků v paměti tagu. Kvůli omezením samotného komunikačního standardu a hardwarovým limitům paměťových bufferů NFC čipu probíhá čtení po částech (*chunked multi-block read*). V rámci jedné transakce lze takto přečíst maximálně 32 bloků najednou, kdy jeden blok představuje 4 bajty paměti. Pokud aplikace komunikuje přes záložní protokol IsoDep, využívá k tomu funkce `readBytesT4t` formát standardního APDU příkazu `READ BINARY`. Přestože se zde paměť adresuje odlišně (přímo po jednotlivých bajtech), uplatňuje se obdobný způsob dělení packetů s limitem nejvýše 240 bajtů na jeden požadavek.

Vyčítání navazujícího bloku naměřených dat je vždy řízeno vrstvou komunikačního manažera (`NfcManager`). Pro tyto účely je implementovaná ochranná logika rozdělování požadavků (*manager-level chunking*). Řídící funkce hlídá, aby žádný zprostředkovaný požadavek předaný servisní vrstvě (`nfcService.readBytes`) nepřekročil limit 240 bajtů. Pokud je požadovaný objem naměřených dat menší, stáhne se celý blok najednou. Pokud však aplikace stahuje rozsáhlejší historii měření (například plnou paměť tagu), manažer požadavek rozdělí do sekvence chunků.

3.4.2 Zápis dat do NFC paměti

Zápis dat do NFC paměti probíhá ve dvou odlišných scénářích: zápis kompletní konfigurace při spuštění nového měření a zápis jediného řídícího bajtu při ukončení probíhajícího měření. Oba scénáře sdílejí stejný základ – session wrapper `runNfcSession` a protokolovou vrstvu `nfcService.writeBytes` – ale liší se rozsahem zapisovaných dat a následnou reakcí zařízení.

Sekvence zápisu při novém měření

Před samotným zápisem konfigurace manažerská vrstva nejprve přečte aktuální version byte z adresy 2 tagu (funkce `readVersionByte`). Pokud bylo zařízení předtím načteno, porovná přečtený version byte s hodnotou odvozenou z naposledy načtené konfigurace (*cross-version ochrana*):

$$expectedVersionByte = (config.hwVersion.major \ll 4) \mid config.hwVersion.minor$$

Pokud se hodnoty liší, operace je přerušena s chybou, protože různé verze firmwaru mají odlišné rozmístění adres v paměti a zápis na nesprávné adresy by data v tagu poškodil. Tato ochrana zajišťuje, že aplikace nikdy nezapíše konfiguraci pro v1.00 na tag s firmwarem v1.01 a naopak.

Po úspěšném ověření verzí dispatcher `tag-memory.writer` vybere odpovídající verzovaný writer z registru `VERSION_WRITERS` a sestaví bajtové pole pro zápis. Serializace konfigurace probíhá v metodě `buildConfigBytes` příslušného writeru a pokrývá adresy 8–26 (v1.00) nebo 8–28 (v1.01), tedy 19, respektive 21 bajtů. Tabulka 5 shrnuje obsah zapisovaného bloku pro obě verze.

Tabulka 5: Obsah zapisovaného konfiguračního bloku (adresy 8–28)

v1.00	**v1.01**	**Obsah**	**Kódování**
8	8	Den začátku měření	přímá hodnota (1–31)
9	9	Měsíc začátku měření	přímá hodnota (1–12)
10	10	Rok začátku měření	$rok - 2000$
11	11	Hodina začátku měření	přímá hodnota (0–23)
12	12	Minuta začátku měření	přímá hodnota (0–59)
13	13	Interval měření	viz tabulka 6
14	14	Max. práh teploty	$H = (t + 22) \cdot 2$
15	15	Min. práh teploty	$H = (t + 22) \cdot 2$
16	16	Alarmy teploty	bitová maska
17	17	Max. práh vlhkosti	$H = RH \cdot 2$
18	18	Min. práh vlhkosti	$H = RH \cdot 2$
19	19	Alarmy vlhkosti	bitová maska
20–21	20–21	Limit CO_2	16bit big-endian, $H = CO_2/16$
22	22	Alarmy CO_2	bitová maska
23–24	23–24	Limit NO_x	16bit big-endian, $H = NO_x/2$
25	25	Alarmy NO_x	bitová maska
–	26	Max. práh akcelerometru	$H = g \cdot 10$ (pouze v1.01)
–	27	Alarmy akcelerometru	bitová maska (pouze v1.01)
26	28	Bitmaska aktivních senzorů	bitová maska senzorů

Alarmy jsou kódovány jako bitová maska v jednom bajtu: bit 0 odpovídá LED alarmu (`0x01`), bit 1 zvukovému alarmu (`0x02`). Interval měření se kóduje dle tabulky 6.

Tabulka 6: Kódování intervalu měření

Interval	**Hodnota** H	**Poznámka**
90 s	`0x5A`	speciální případ
n minut	n	obecný případ, $H = sekund/60$
24 hodin	`0x18`	pouze v1.01

Po zapsání konfiguračního bloku manažer zapíše *device control byte* na adresu 27 (v1.00) nebo 29 (v1.01) s hodnotou `0x02`. Tato hodnota signalizuje zařízení, aby zahájilo nové měření a zároveň vymazalo stará naměřená data z paměti. Vymazání dat a veškerou následnou správu paměti provádí samo zařízení, aplikace se o ni nestará.

Sekvence ukončení měření

Při ukončení probíhajícího měření aplikace zapíše pouze jediný bajt: device control byte s hodnotou `0x01` na příslušnou adresu dle verze firmwaru. Žádná konfigurační data se při tomto úkonu nezapisují a dosud naměřená data v paměti tagu zůstávají nedotčena.

Implementace na úrovni protokolu

Způsob fyzického zápisu bajtů do paměti se liší v závislosti na aktivním komunikačním protokolu. Při použití NfcV implementuje funkce `writeBytesNfcV` zápis po čtyřbajtových blocích (ISO 15693 Write Single Block). Jelikož NFC paměť je adresována po blocích a ne po bajtech, je nutné ošetřit tzv. okrajové bloky – situace, kdy zapisovaná data nezačínají nebo nekončí na hranici bloku. V těchto případech aplikace nejprve přečte existující obsah daného bloku (*read-modify-write*), sloučí ho s novými daty a teprve poté celý blok zapíše. Bloky, které jsou zapisovány celé, se přepisují přímo bez předchozího čtení.

Při záložním protokolu IsoDep je zápis realizován funkcí `t4tUpdateBinary` standardním APDU příkazem `UPDATE BINARY` (INS `0xD6`). Paměť je zde adresována přímo po bajtech, takže odpadají komplikace s okrajovými bloky a zápis probíhá bez nutnosti předchozího čtení.

3.4.3 Validace dat

Validace dat probíhá ve dvou fázích – před parsováním konfigurace a před každým zápisem na tag – a zajišťuje, že aplikace zpracovává pouze data ze zařízení se správnou strukturou paměti.

Ověření identity zařízení (magic bytes)

První dvě adresy NFC paměti obsahují unikátní identifikátor zařízení (*magic bytes*) s pevnou hodnotou `0xC0 0x13`. Funkce `readAndValidateConfig` načte první tři bajty jako úvodní sondu (*probe*) a porovná bajty na adresách 0 a 1 s očekávanými hodnotami. Pokud se liší, čtení je okamžitě přerušeno s chybou a žádná další operace s tagem neprobíhá. Tím je zajištěno, že aplikace nezpracuje data z libovolného jiného NFC tagu, který se ocitne v dosahu.

Ověření verze firmwaru

Bajt na adrese 2 kóduje verzi firmwaru ve formátu horní nibble = hlavní verze, dolní nibble = vedlejší verze:

$$versionByte = (major \ll 4) \mid minor$$

Na základě tohoto bajtu dispatcher `tag-memory.parser` vyhledá odpovídající parser v registru `VERSION_PARSERS`. Pokud verze není registrována (tzn. firmware zařízení je no-

vější, než aplikace zná), parsování selže s explicitní chybovou zprávou uvádějící přečtenou verzi a seznam podporovaných verzí.

Cross-version ochrana při zápisu

Před zápisem konfigurace manažerská vrstva vždy nejprve přečte aktuální *version byte* přímo z tagu a porovná ho s verzí odvozenou z naposledy načtené konfigurace:

$$expectedVersionByte = (config.hwVersion.major \ll 4) \mid config.hwVersion.minor$$

Pokud se hodnoty liší, operace zápisu je přerušena. Tento mechanismus chrání tag před poškozením v situaci, kdy by uživatel přiložil jiné zařízení s odlišnou verzí firmwaru, než pro které byla konfigurace připravena, protože různé verze mají odlišné rozmístění adres v paměti a zápis na nesprávné adresy by data v tagu poškodil.

Chybový indikátor senzorových hodnot

Při parsování naměřených dat je hodnota $H = 255$ (`0xFF`) u všech veličin rezervována jako indikátor chyby senzoru. Funkce `parseSensorValue` v každém verzovaném parseru vrací pro tuto hodnotu `null` namísto přepočítané fyzikální hodnoty. Aplikace tuto hodnotu interpretuje jako chybějící měření a při vizualizaci daný bod v grafu vynechá, čímž nedojde ke zobrazení zavádějících dat.

3.4.4 Ošetření chyb a výjimek

Ošetření chyb je v aplikaci řešeno na více úrovních, od nízkoúrovňových chyb NFC protokolu až po stavové *modaly* v prezentační vrstvě.

Chyby na úrovni protokolu

Na úrovni protokolové vrstvy (`nfc.service.ts`) jsou rozlišovány dva typy selhání v závislosti na aktivním protokolu. U NfcV příkazů je první bajt každé odpovědi příznakovým registrem (*flags byte*) dle standardu ISO 15693 – je-li nastaven bit 0, odpověď indikuje chybu a případný druhý bajt nese kód chyby. Funkce `nfcvReadSingleBlock`, `nfcvReadMultipleBlocks` a `nfcvWriteSingleBlock` v takovém případě vyhodí výjimku s popisem adresy bloku a hexadecimálním kódem chyby.

U IsoDep příkazů ověřuje funkce `assertT4tSuccess` poslední dva bajty každé APDU odpovědi (*status word* SW1-SW2). Pokud se status liší od očekávané hodnoty `90 00`, je vyhozena výjimka s informací o konkrétním příkazu, který selhal, a přijatém status slovu.

Přerušení komunikace a záměrné zrušení session

Klíčovým případem, který je třeba odlišit od skutečné chyby, je záměrné zrušení NFC session při navigaci uživatele na jinou záložku. Komponenta `NfcState` sleduje fokus obrazovky pomocí hooku `useIsFocused`.

Při ztrátě fokusu nastaví příznak `isCancelledRef.current = true` a okamžitě zavolá `NfcManager.cancelTechnologyRequest`. Tento příznak je kontrolován v bloku `catch` handleru čtení – pokud je nastaven, chyba je tiše ignorována a stav se resetuje do `idle` bez zobrazení chybového UI. Díky tomu uživatel nevidí chybové hlášení při přepnutí záložky v průběhu čekání na tag.

Zobrazování stavu čtení a automatický reset

Průběh NFC operace v prezentační vrstvě je modelován stavovým automatem s hodnotami `idle` → `waiting` → `working` → `success` / `error`. Stavy `success` a `error` jsou přechodné a po uplynutí 2, respektive 3 sekund se stav automaticky vrátí do `idle`, čímž je umožněno zahájení dalšího čtení bez nutnosti zásahu uživatele. Toto chování zabraňuje uvíznutí UI v chybovém stavu po přechodném selhání komunikace.

Neplatná nebo neúplná data

Funkce `readData` v manažerské vrstvě po dokončení chunked čtení ověří, zda počet přijatých bajtů odpovídá požadovanému rozsahu. Pokud dojde k neúplnému přenosu (tag se vzdálil příliš brzy nebo komunikace byla přerušena), je vyhozena výjimka s přesným počtem přijatých a očekávaných bajtů. Tato výjimka je zachycena handlerem v komponentě `NfcState`, která zobrazí chybový stav a po uplynutí časového limitu umožní opakovat akci.

Uživatelská zpětná vazba

Veškeré chybové stavy jsou uživateli komunikovány prostřednictvím komponenty `NfcState`, která zobrazuje modální overlay s ikonou chyby a textem chybové zprávy. Zprávy jsou formulovány srozumitelně (např. „Nepodařilo se načíst data z NFC tagu“) a doplněny výzvou k opakování pokusu. Technické detaily chyb (kódy protokolu, offsety bloků) jsou logovány pouze do vývojářské konzole a uživateli nejsou zobrazovány.

3.4.5 Kompatibilita verzí firmware

Podpora více verzí firmware je zajištěna registrovým vzorem (*registry pattern*) popsaným v podkapitole 3.2.1 – přidání nové verze vyžaduje pouze vytvoření nových souborů `v1_0x.parser.ts` a `v1_0x.writer.ts` a jejich registraci do slovníků `VERSION_PARSERS` a `VERSION_WRITERS`, bez zásahu do stávajícího kódu.

Rozdíly v paměťové mapě mezi verzemi v1.00 a v1.01 jsou shrnuty v tabulce 3. Mechanismus detekce verze při čtení je popsán v podkapitole 3.4.3, ochrana integrity dat při zápisu pak v podkapitole 3.4.2 věnované zápisu konfigurace.

3.5 Vizualizace dat

Vizualizace naměřených dat je implementována na záložce Data. Záložka umožňuje prohlížení historických snímků ze všech přečtených zařízení, přepínání mezi typy grafů, filtrování dat podle časového okna a zobrazení souhrnných statistik.

3.5.1 Grafická komponenta

Grafy jsou vykresleny pomocí knihovny `react-native-svg` [9]. Tento přístup byl zvolen místo hotových grafových knihoven z důvodu plné kontroly nad vykreslením, konkrétně pro implementaci AQI barevných pásů podbarvení, přerušení čáry při chybě senzoru a vlastní ovládání posunu po *timeline* dotykem.

Aplikace rozlišuje čtyři stránky grafů, přičemž každá se zobrazuje pouze tehdy, jsou-li příslušná data dostupná v načteném *snapshotu*:

- **Teplota + Vlhkost** – dvě čáry grafu na společné ose; teplota červeně, vlhkost modře
- **CO_2 + AQI** – čárový graf oxidu uhličitého s barevným podbarvením pozadí grafu dle aktuálního pásma AQI (viz tabulka 7)
- **NO_x** – koncentrace oxidů dusíku jako samostatný graf
- **Zrychlení** – maximální naměřené zrychlení společné pro všechny osy; záznamy s hodnotou $g = 0$ jsou v grafu zvýrazněny červeným pruhem na pozadí jako indikátor možného volného pádu (*free fall*)

Hodnoty $H = 255$ parsované jako `null` způsobují přerušení čáry (*gap*) v grafu – metoda `buildPath` v komponentě `GraphView` zahájí nový segment příkazem MoveTo při každém výskytu `null`. Při nízkém počtu bodů (do 60) jsou navíc vykresleny tečky v datových bodech a symbol „×“ na místě chybějícího měření.

Graf podporuje horizontální posuv (*pan*) prostřednictvím `PanResponder`. Přejetí prstem doleva posouvá zobrazení dopředu v čase, doprava dozadu. Klepnutím na datový bod se zobrazí plovoucí tooltip s přesnou hodnotou a časovou značkou. Tooltip je automaticky skryt při zahájení posuvu.

Tabulka 7: Barevná pásma AQI (0–20)

Rozsah AQI	Hodnocení	Barva
0–4	Výborná	zelená
5–8	Dobrá	světle zelená
9–12	Přijatelná	žlutá
13–16	Špatná	oranžová
17–20	Velmi špatná	červená

3.5.2 Transformace dat pro zobrazení

Naměřená data uložená v paměti neobsahují explicitní časové značky jednotlivých záznamů, ty se dopočítávají až při vizualizaci. Funkce `generateTimestamps` generuje pole objektů `Date` dle vztahu:

$$t_i = t_{\text{start}} + i \times \Delta t_{\text{interval}}$$

kde t_{start} je čas začátku měření z konfigurace (`measuringStartTime`), $\Delta t_{\text{interval}}$ je interval měření v sekundách a i je index záznamu od 0.

Zobrazené časové okno je určeno volbou uživatele (1 hod, 6 hod, 24 hod, 7 dní, 30 dní, nebo vše) a aktuálním posunem `scrollOffsetMs`. Funkce `filterByTimeWindow` vrátí indexy `startIdx` a `endIdx`, jimiž se ořezávají všechna datová pole před předáním do `GraphView`. Maximální dosažitelný posun je omezen hodnotou:

$$maxScrollOffset = t_{\text{poslední}} - t_{\text{první}} - \Delta t_{\text{okno}}$$

Načtené snímky jsou před zobrazením deduplikovány funkcí `mergeSnapshots`. Ta seskupuje záznamy podle kompozitního klíče ($UID + t_{\text{start}}$) a ze skupiny ponechá pouze *snapshot* s nejnovějším `readTimestamp`, protože obsahuje aktuálnější data.

3.5.3 Legenda a indikátory

Pod grafem jsou zobrazeny tyto vizuální prvky:

- **Legenda řad** – barevné čáry s popisem veličiny a jednotky, zobrazuje se pouze při více než jedné řadě
- **Legenda limitů** – přerušované čáry odpovídající limitním hodnotám z konfigurace, zobrazení lze přepínačem *Limity* vypnout
- **Indikátor chyby senzoru** – zobrazí se, pokud viditelný úsek obsahuje alespoň jedno `null` (tj. hodnotu $H = 255$)

- **AQI legenda** – barevná stupnice s rozsahy a popisy pásem, zobrazuje se pouze na stránce CO_2 + AQI

Pod grafem je dále umístěna tabulka souhrnných statistik (`SummaryTable`), která pro každou viditelnou řadu zobrazuje minimum, maximum a průměr přes aktuálně zobrazený úsek. Hodnoty `null` (chyby senzoru) jsou z výpočtu vyloučeny a počet platných bodů se bere jako základ průměru.

3.6 Správa konfigurace

Záložka Konfigurace slouží k nastavení parametrů měření a k ovládání jeho průběhu. Uživatel zde vybírá aktivní senzory, interval měření, alarmové limity a spouští nebo ukončuje měření. Zadaná konfigurace se zapisuje přímo do NFC paměti zařízení způsobem popsaným v podkapitole 3.4.2.

3.6.1 Konfigurační parametry

Bitmaska aktivních senzorů je uložena v jednom bajtu (adresa 26 pro v1.00, adresa 28 pro v1.01), přičemž každý bit odpovídá jednomu senzoru. Aplikace zobrazí pouze senzory označené jako dostupné v poli `availableSensors` přečtené konfigurace, senzory nepodporované hardwarem jsou v UI nedostupné. Interval měření se kóduje jako jeden bajt dle tabulky 6 a uživatel jej vybírá z předdefinované sady hodnot. Pro každou aktivní veličinu lze nastavit prahové hodnoty a způsob upozornění (LED, zvuk nebo obojí) kódovaný jako bitová maska v jednom bajtu. Řídící bajt *device control* je write-only pole, hodnota `0x02` zahájí nové měření a smaže stará data, hodnota `0x01` probíhající měření ukončí bez zásahu do naměřených dat.

3.6.2 Perzistentní úložiště konfigurace

Konfigurační hodnoty jsou průběžně ukládány do MMKV [10] pod strukturovanými klíči s prefixem `config.*`, odděleně od klíčů pro *snapshoty* dat (`sensordata_`) a zobrazované názvy zařízení (`displayname_`). Při otevření záložky se hodnoty načítají s prioritou, nejprve z MMKV, a pokud tam žádné nejsou, předvyplní se z naposledy přečtené konfigurace tagu.

3.6.3 Ovládání měření

Záložka obsahuje jediné akční tlačítko, jehož funkce se mění podle stavu zařízení. Pokud měření neprobíhá, tlačítko *Nové měření* sestaví konfigurační bajty, doplní aktuální čas jako čas zahájení a zapíše vše na tag spolu s device control bytem `0x02`. Pokud zařízení měří (`measuringStatus == "ongoing"`), tlačítko *Ukončit měření* zapíše pouze device

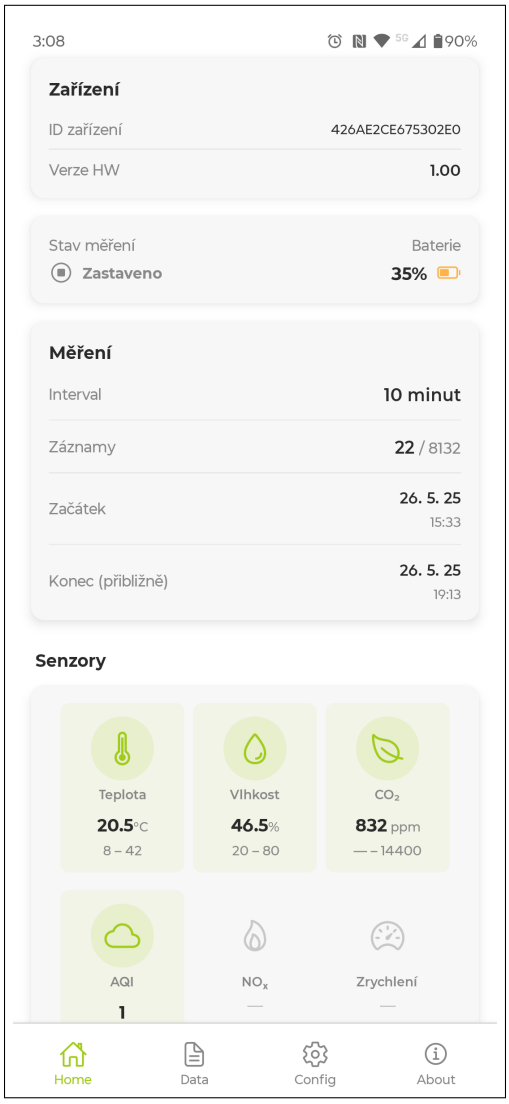
control byte hodnotou `0x01`. Průběh zápisu je indikován stavovým modalem se stavy `waiting`, `working`, `success` a `error`, analogickým k modalu čtení dat.

3.7 Uživatelské rozhraní

3.7.1 Navigace a obrazovky

Aplikace využívá knihovnu Expo Router s navigací založenou na záložkách (*tab-based navigation*). Každá záložka představuje samostatnou obrazovku s jasně vymezenou funkcí.

Záložka **Home** slouží jako hlavní dashboard. Po přiložení telefonu k tagu proběhne čtení automaticky bez nutnosti stisknout tlačítko. Obrazovka zobrazuje stav zařízení, napětí baterie a přehledný grid ikon s posledními naměřenými hodnotami.

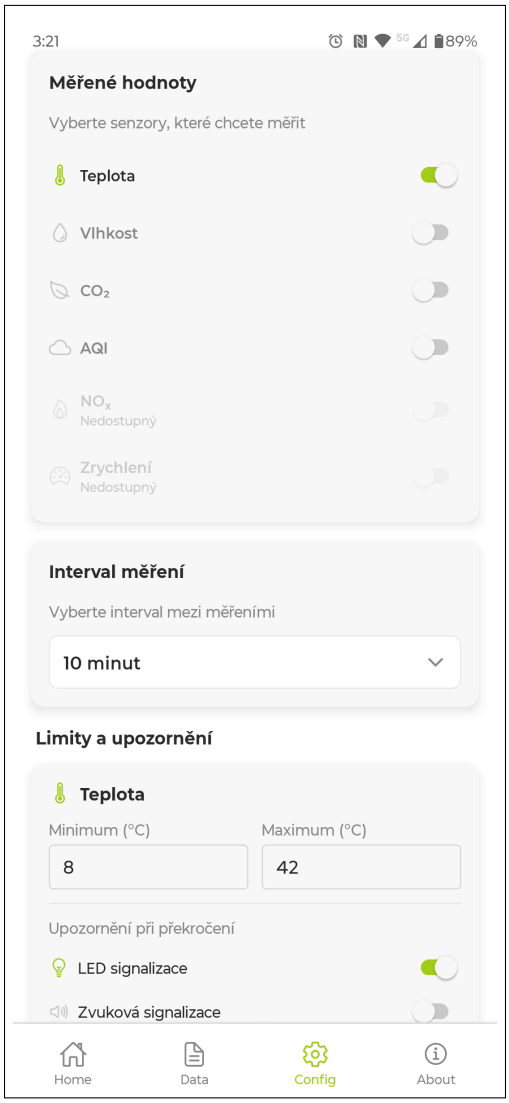


Obrázek 4: Záložka Home – dashboard s posledními naměřenými hodnotami

Záložka **Data** umožňuje prohlížení naměřených hodnot ve formě grafů. Zobrazují se pouze senzory dostupné v načteném snímku. Uživatel může volit časové okno a procházet historii měření. Pod grafem je zobrazena legenda a tabulka souhrnných statistik, jak je popsáno v podkapitole věnované vizualizaci dat.

Obrázek 5: Záložka Data – graf naměřených hodnot s legendou a statistikami

Záložka **Konfigurace** je popsána v předchozí podkapitole. Obsahuje výběr senzorů, rozbalovací menu pro interval měření, nastavení alarmů pro každou veličinu a ovládací tlačítko pro spuštění nebo ukončení měření.

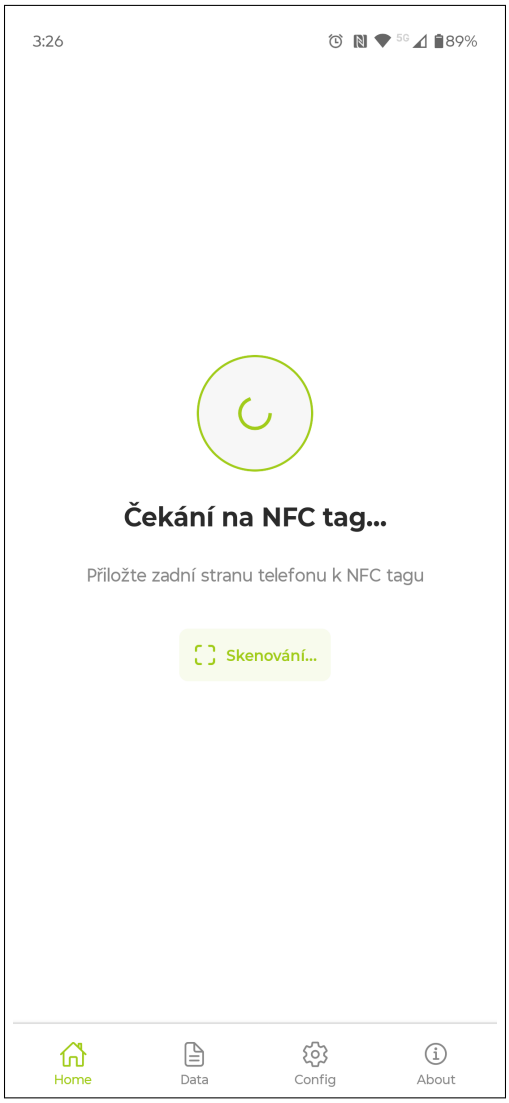


Obrázek 6: Záložka Konfigurace – nastavení senzorů, intervalu a alarmů

3.7.2 Stavy aplikace a zpětná vazba

Stav NFC adaptéru je sledován v reálném čase komponentou `NfcProvider`. Ta rozlišuje čtyři stavy: `CHECKING` (probíhá zjišťování), `ENABLED` (NFC je aktivní), `DISABLED` (NFC je vypnuto, zobrazí se tlačítko pro otevření systémového nastavení) a `NOT_SUPPORTED` (zařízení NFC nepodporuje). Na Androidu jsou změny stavu NFC sledovány pomocí systémového posluchače událostí.

Průběh NFC operace je modelován stavovým automatem s hodnotami `idle` → `waiting` → `working` → `success` / `error`, přičemž stavy `success` a `error` jsou přechodné a po uplynutí krátkého časového limitu se stav automaticky vrátí do `idle`. Centrálním správcem tohoto stavového UI je komponenta `NfcState`, která během aktivní operace zobrazuje modální overlay s příslušnou ikonou a textem.



Obrázek 7: Stavový modal – čekání na přiložení NFC tagu

Při navigaci uživatele na jinou záložku v průběhu čekání na tag komponenta `NfcState` detekuje ztrátu fokusu pomocí hooku `useFocusEffect`, nastaví příznak `isCancelledRef` a okamžitě zruší NFC session. Záměrné zrušení je tímto příznakem odlišeno od skutečné chyby, takže uživatel neuvidí chybové hlášení při přepnutí záložky.

4 Výsledky

Tato kapitola shrnuje výsledky vývoje mobilní aplikace a ověřuje dosažení cílů definovaných v zadání projektu. Popisuje splněné funkcionality, průběh testování na reálném hardwaru i na syntetických binárních datech a chování aplikace v mezních a chybových stavech.

4.1 Přehled splněných cílů

Tabulka 8 porovnává požadavky ze specifikace projektu s dosaženým stavem implementace.

Tabulka 8: Porovnání specifikace projektu s dosaženým stavem

Požadovaná funkcionalita	**Stav**	**Poznámka**
Čtení dat z NFC paměti po přiblížení telefonu	✓	NfcV + IsoDep záloha
Validace identity zařízení (magic bytes)	✓	sekce 3.4.3
Vizualizace naměřených hodnot v grafech	✓	záložka Data
Informační panel (stavové a konfigurační údaje)	✓	záložka Home
Zápis konfiguračních dat do NFC paměti	✓	sekce 3.4.2
Spuštění nového měření (ovládací příkaz)	✓	záložka Konfigurace
Ukončení probíhajícího měření (ovládací příkaz)	✓	záložka Konfigurace
Podpora firmware v1.00	✓	parser + writer
Podpora firmware v1.01 (rozšíření o G-senzor)	✓	parser + writer téma dále rozvíjeno v rámci produktové řady *DevicePass* (viz sekce 5.4)
Výběr nástrojů pro multiplatformní vývoj	✓	podpora iOS + Android
Export vizualizovaných dat	×	bonusový cíl

Všechny primární funkcionality ze specifikace projektu byly implementovány a ověřeny na cílovém hardwaru. Bonusový cíl exportu vizualizovaných dat nebyl z časových důvodů realizován. Jeho případné doplnění je diskutováno v podkapitole věnované omezením a možným vylepšením.

4.2 Funkční aplikace

Výsledkem projektu je mobilní aplikace sestavená v React Native. Aplikace se skládá ze tří záložek – Home, Data a Konfigurace – jejichž rozhraní je ukázáno na obrázcích 4, 5 a 6.

4.3 Testování

Tato část projednává způsoby ověřování funkčnosti celé aplikace.

4.3.1 Testování na reálném hardwaru

Aplikace byla nejprve testována s deskou ANT7-T-M24SR64 [11] a následně se po přepsání čtecí logiky vývoj přesunul na cílové hardwarové zařízení vybavené NFC čipem ST25DV64KC [1]. Testování funkčnosti aplikace bylo provedeno na mobilních zařízeních uvedených v tabulce 9.

Tabulka 9: Testovaná mobilní zařízení

Zařízení	**OS**	**Verze OS**	**Protokol**
Motorola G84 5G	Android	15	NfcA / IsoDep (podporuje NfcV)
Xiaomi Redmi Note 9 Pro	Android	12	NfcA / IsoDep (podporuje NfcV)
Samsung Galaxy S22	Android	15	NfcA / IsoDep (podporuje NfcV)
Honor 400 Lite	Android	15	NfcA / IsoDep (podporuje NfcV)
Realme C21	Android	11	bez NFC podpory

Tabulka 10 shrnuje provedené testovací scénáře a jejich výsledky.

Tabulka 10: Testovací scénáře – reálný hardware

Scénář	Výsledek	Poznámka
Čtení tagu s firmware v1.00	✓	všechna pole parsována správně
Čtení tagu s firmware v1.01	✓	vč. dat z G-senzoru
Čtení tagu bez očekávané struktury	✓	aplikace nepokračovala dál ve čtení
Zápis konfigurace a spuštění nového měření	✓	zařízení zahájilo měření
Ukončení probíhajícího měření	✓	data v paměti zachována
Čtení po dokončení sady záznamů	✓	správný počet záznamů
Vizualizace dat v grafech	✓	vč. AQI pásem a chyb senzorů

Dalším testem bylo ověření správnosti de-interleavingu dat. V případě chyby by každý senzor zobrazoval hodnoty jiného senzoru posunuté o konstantní offset indexu – například by graf teploty zobrazoval hodnoty vlhkosti – a samotné přepočty by přitom algoritmicky proběhly bez chyby, takže by se problém neprojevil žádnou výjimkou.

4.3.2 Edge cases a chybové stavy

V průběhu testování byly ověřeny všechny definované chybové toky aplikace. Níže jsou popsány testované mezní stavy a chování aplikace.

Neplatné magic bytes (cizí NFC tag)

Čtení je přerušeno ve fázi probe čtení a vyhozena výjimka s popisem nesouladu hodnot na adresách 0 a 1. Uživateli se zobrazí chybový modal; po 3 sekundách se stav automaticky resetuje do `idle`.

Cross-version nesoulad při zápisu (tag v1.00 vs. konfigurace v1.01)

Zápis je odmítnut ještě před sestavením bajtového pole – manažerská vrstva detekuje neshodu *version byte* přečteného přímo z tagu s hodnotou odvozenou z naposledy načtené konfigurace. Uživateli se zobrazí chybový modal; data tagu nejsou dotčena.

Neúplné čtení (tag vzdálen příliš brzy)

Je vyhozena výjimka s přesným počtem přijatých vs. očekávaných bajtů, zachycená v handleru komponenty `NfcState`. Uživateli se zobrazí chybový modal; po 3 sekundách lze operaci opakovat.

Navigace pryč v průběhu čekání na tag

Nastaven příznak `isCancelledRef`, NFC session je korektně zrušena a výsledná výjimka je tiše ignorována v bloku `catch`. Uživatel neuvidí žádné chybové hlášení – záměrné zrušení je tímto příznakem odlišeno od skutečného selhání komunikace a stav se tiše resetuje do `idle`.

NFC adaptér vypnut v průběhu operace

Komponenta `NfcProvider` detekuje systémovou změnu stavu adaptéru a přepne výčet `NfcStatus` do hodnoty `DISABLED`. Uživateli se zobrazí výzva k zapnutí NFC s odkazem do systémového nastavení.

Nepodporovaná verze firmware

Dispatcher `tag-memory.parser` nenalezne klíč přečteného *version byte* v registru `VERSION_PARSERS` a vyhodí výjimku uvádějící přečtenou verzi a seznam podporovaných verzí. Uživateli se zobrazí informativní chybová zpráva.

Zvláštní pozornost si zasloužil případ přerušení čtení při navigaci na jinou záložku. V rané fázi vývoje způsobovalo toto chování zobrazení chybového hlášení vždy, když uživatel přepnul záložku v době, kdy komponenta čekala na přiložení tagu. Problém byl vyřešen zavedením příznaku `isCancelledRef`, díky němuž handler v bloku `catch` rozliší záměrné zrušení od skutečného selhání komunikace a tiše resetuje stav do `idle` bez zobrazení chybového UI.

5 Diskuse

Tato kapitola hodnotí průběh projektu, zdůvodňuje technická rozhodnutí učiněná během vývoje, porovnává zvolené přístupy s alternativami a shrnuje omezení implementace včetně návrhu možných budoucích vylepšení.

5.1 Zhodnocení průběhu projektu

Vývoj projektu probíhal ve spolupráci s hardwarovou částí týmu, se kterou bylo nutné sjednotit použitý komunikační protokol a strukturu NFC paměti. Díky dohodnuté struktuře a konceptu magic bytes spolu s version byte bylo možné implementovat robustní validaci dat na straně aplikace.

Vývoj se ve značné části vývoje opíral o vývojovou desku ANT7-T-M24SR64 [11], která umožnila ověřit základní logiku zpracování dat dříve, než byl k dispozici cílový hardware. Přechod na čip ST25DV64KC [1] si vyžádal přepsání části čtecí logiky, protože se NFC adresový prostor a chování multi-block operací lišily od vývojové desky. Díky zvolené vícevrstvé architektuře se však tato změna dotkla výhradně protokolové vrstvy (`nfc.service.ts`) a vyšší vrstvy zůstaly beze změny.

Všechny primární funkcionality ze specifikace projektu byly úspěšně implementovány a ověřeny na reálném hardware, jak dokumentuje tabulka 8. Jediným nesplněným bodem zůstal bonusový cíl exportu vizualizovaných dat, jehož realizaci zabránil časový rozvrh projektu.

5.2 Zdůvodnění zvoleného řešení

Volba frameworku React Native

React Native byl zvolen jako vývojový framework z důvodu možnosti sdílet jednu kódovou základnu pro obě cílové platformy, iOS a Android, bez nutnosti udržovat dvě oddělené nativní implementace. Klíčovým faktorem byl rovněž existující ekosystém knihoven, zejména `react-native-nfc-manager` [12], která poskytuje jednotné rozhraní nad nativními NFC API obou platforem. Nativní přístup k NFC se na iOS a Androidu výrazně liší – iOS využívá CoreNFC [5] a Android vlastní `android.nfc` [6] API – a implementovat tuto abstrakci vlastními silami by znamenalo duplicitní práci v Swiftu a Kotlinu s minimálním přínosem pro daný rozsah projektu.

Vlastní SVG rendering místo grafové knihovny

Pro vykreslování grafů byla zvolena nízkoúrovňová knihovna `react-native-svg` [9] místo hotových grafových řešení, jako jsou Victory Native [13] nebo Recharts [14]. Důvodem byla potřeba plné kontroly nad vykreslenou geometrií pro tři konkrétní požadavky: barevné podbarvení pozadí grafu dle pásem AQI (tabulka 7), přerušení čáry (*gap*) při

hodnotě `null` odpovídající chybě senzoru a vlastní gesta posuvu po časové ose implementovaná přes `PanResponder`. Ověřené hotové knihovny nepodporovaly kombinaci těchto tří funkcí současně a jejich přizpůsobení by bylo náročnější než přímá implementace nad SVG primitivy. V budoucnu bude tato otázka přehodnocena, zda by se na trhu nenašlo vhodné alternativní řešení.

MMKV místo AsyncStorage

Jako perzistentní úložiště bylo zvoleno MMKV [10] namísto standardního AsyncStorage. Rozhodujícím faktorem byl synchronní přístup k datům. AsyncStorage je ze své podstaty asynchronní a jeho použití v *React hooks* by vyžadovalo komplexní správu vedlejších efektů s rizikem tzv. *„race conditions“* při opakovaném rychlém čtení tagu. MMKV navíc disponuje výrazně vyšším výkonem díky přímé vazbě na nativní implementaci v C++, bez přechodu přes JS bridge, takže pro synchronní načítání dat při inicializaci obrazovek je tato volba přirozená.

Vzor Strategy a registrový vzor pro správu verzí firmware

Návrh parserů a writerů jako implementací rozhraní registrovaných do slovníku dle *version byte* (podkapitola 3.2.1) se ukázal jako správná volba v momentě, kdy hardwarový tým vydal druhou verzi datového formátu v1.01. Přidání podpory vyžadovalo výhradně vytvoření dvou nových souborů (`v1_01.parser.ts` a `v1_01.writer.ts`) a registraci klíče `0x11` do příslušných slovníků. Stávající parsery, writery ani manažerská vrstva se nijak nezměnily a zpětná kompatibilita s v1.00 zůstala zachována. V případě budoucí verze v1.02 platí stejný postup.

5.3 Porovnání s alternativními přístupy

Flutter vs. React Native

Uvažovanou alternativou k React Native byl framework Flutter. Obě platformy nabízejí srovnatelnou úroveň podpory multiplatformního vývoje a NFC komunikace. Pro React Native hovoří výrazně větší komunita vývojářů a zralejší ekosystém knihoven v době psaní dokumentace, zejména knihovna `react-native-nfc-manager`, která poskytuje přístup k nízkoúrovňovým NFC funkcím potřebným pro práci s paměťovými tagy. Přechod na Flutter by si vyžádal vlastní implementaci abstrakce nad platformně specifickými NFC API nebo utilizaci méně udržované knihovny, přičemž výsledná aplikace by z pohledu uživatele nebyla funkčně odlišná. Dalším rozhodujícím faktorem byla předchozí znalost vývoje v React.

Bluetooth vs. NFC pro komunikaci s embedded zařízením

Alternativou k NFC by bylo využití Bluetooth Low Energy (BLE). BLE umožňuje vyšší přenosové rychlosti a komunikaci na větší vzdálenost, avšak vyžaduje nepřetržité napájení komunikačního modulu v zařízení a proces párování. NFC přináší dvě klíčové výhody specifické pro toto nasazení: komunikace probíhá pouhým přiložením telefonu bez nutnosti párování, a díky použití čipu ST25DV64KC s duálním rozhraním je čtení dat možné i v situaci, kdy je hlavní baterie zařízení zcela vybitá – potřebnou energii pro komunikaci s NFC pamětí získává čip indukcí z elektromagnetického pole mobilního telefonu [1]. Pro datalogger s přerušovaným sběrem dat a občasnou kontrolou naměřených hodnot je NFC vhodnější volbou než BLE.

5.4 Omezení a možná vylepšení

Rozšířená analýza dat G-senzoru (produktová řada DevicePass)

Aktuálně implementovaná podpora G-senzoru v rámci firmwaru v1.01 ukládá do NFC paměti prostou hodnotu maximálního vektorového zrychlení jako jednobajtový záznam. V průběhu vývoje projektu přišel první reálný průmyslový požadavek: měřit provozní hodnoty zařízení za účelem posouzení reklamací způsobených nadměrným mechanickým namáháním. To vedlo k upřednostnění produktové řady *DevicePass* a k zásadnímu rozšíření role G-senzoru v celém systému, jehož konkrétní podoba na úrovni datových struktur a firmwaru se v době odevzdání této práce teprve finalizuje. Mobilní aplikace bude muset reagovat vydáním nové verze parseru a writeru a rozšířením vizualizační vrstvy – díky zvolené registrové architektuře (sekce 3.4.5) však tento rozvoj nevyžaduje zásah do stávajícího kódu.

Export vizualizovaných dat

Export naměřených dat do formátu CSV nebo PDF byl definován jako bonusový cíl v zadání projektu a z časových důvodů nebyl realizován. Implementace by technicky nevyžadovala zásah do stávající architektury – data jsou již uložena jako strukturované *snapshoty* v MMKV a jejich serializace do CSV by spočívala v iteraci přes pole hodnot s dopočítanými časovými značkami dle vztahu popsaného v podkapitole věnované transformaci dat. Případné rozšíření lze doplnit jako samostatnou funkci v záložce Data bez dopadu na ostatní části aplikace.

Push notifikace při překročení alarmových limitů

Při prezentaci tohoto projektu byla mezi lidmi opakovaně vznášena otázka, zda systém bude podporovat zasílání oznámení při překročení nastavených limitů. Tato funkce však není realizovatelná v rámci stávajícího hardwarového a komunikačního řešení. Push notifikace by vyžadovaly, aby aplikace průběžně získávala nová data ze senzorů na pozadí, což je principiálně neslučitelné s NFC komunikací, která funguje výhradně reaktivně, kdy

data jsou dostupná pouze v momentu fyzického přiložení telefonu k tagu. Implementace takových notifikací by vyžadovala zásadní zásah do základního principu cílového zařízení na úrovni hardware, což výrazně přesahuje rámec tohoto projektu i zadání.

Testování na iOS

Aplikace nebyla prozatím otestována na platformě iOS. Důvodem je absence přístupu k zařízení Apple v průběhu vývoje a skutečnost, že sestavení aplikace pro iOS vyžaduje macOS a vývojářský účet v Apple Developer Program. Vývoj přesto probíhal s ohledem na kompatibilitu s oběma platformami, kdy použité knihovny i NFC API jsou multiplatformní. Ověření funkčnosti na iOS zůstává otevřeným bodem pro budoucí testování.

6 Závěr

Cílem této maturitní práce bylo navrhnout a vyvinout multiplatformní mobilní aplikaci pro komunikaci s embedded zařízením prostřednictvím NFC. Výsledkem je funkční aplikace sestavená v React Native, která pokrývá všechny primární funkcionality definované v zadání projektu. Aplikace umožňuje bezkontaktní čtení naměřených dat z NFC paměti zařízení po přiložení telefonu, jejich vizualizaci formou interaktivních grafů, konfiguraci zařízení prostřednictvím zápisu dat do NFC paměti a ovládání průběhu měření. Plná podpora je implementována pro obě aktuálně vydané verze firmwaru, v1.00 a v1.01, přičemž aplikace umožňuje přidání podpory dalších verzí bez zásahu do stávajícího kódu.

Z hlediska osobního přínosu přinesl projekt řadu nových znalostí a zkušeností. Práce s NFC protokoly ISO 15693 a T4T na nízké úrovni přinesla pochopení fyzických omezení bezdrátové komunikace – zejména nutnosti dělení přenosů do chunků, ošetření okrajových bloků při zápisu a rozdílů v adresování paměti mezi protokoly. Klíčovým architektonickým poznatkem bylo ověření, že investice do dobře navržené vícevrstvé architektury s jasně vymezenými odpovědnostmi se vrátí při každé iteraci protokolu, jako tomu bylo při přidání podpory v1.01.

Co se týče dalšího rozvoje aplikace, za nejbližší časově dostupné rozšíření lze považovat implementaci exportu naměřených dat do formátu CSV/PDF, která jako bonusový cíl nebyla z časových důvodů implementována, ale její realizace by neměla vyžadovat zásah do stávající architektury. Další možnou cestou rozvoje je přehodnocení aktuálního řešení grafování v záložce data, odstranění fixních časových rámců zobrazených dat a implementace možnosti přiblížení/oddálení dotykem dvou prstů. Ostatní potencionální vylepšení a jejich technická proveditelnost jsou rozebrána v kapitole 5.

Seznam použité literatury

1. STMICROELECTRONICS. *ST25DV64KC Datasheet* [online]. 2024. [cit. 2026-03-15]. Dostupné z: https://www.st.com/resource/en/datasheet/st25dv64kc.pdf.
2. WAKDEV. *NFC Tools* [online]. [cit. 2026-03-15]. Dostupné z: https://nfc.software/.
3. STMICROELECTRONICS. *ST25 NFC Tap* [online]. [cit. 2026-03-15]. Dostupné z: https://www.st.com/en/embedded-software/stsw-st25001.html.
4. NFC FORUM. *NFC Technology* [online]. [cit. 2026-03-15]. Dostupné z: https://nfc-forum.org/learn/nfc-technology.
5. APPLE INC. *Core NFC | Apple Developer Documentation* [online]. [cit. 2026-03-16]. Dostupné z: https://developer.apple.com/documentation/corenfc.
6. GOOGLE LLC. *NFC basics | Android Developers* [online]. [cit. 2026-03-16]. Dostupné z: https://developer.android.com/develop/connectivity/nfc.
7. *IEC 60068-2-6: Environmental Testing – Part 2-6: Tests – Test Fc: Vibration (Sinusoidal)*. International Electrotechnical Commission, 2007. No. IEC 60068-2-6.
8. *IEC 60068-2-64: Environmental Testing – Part 2-64: Tests – Test Fh: Vibration, Broadband Random and Guidance*. International Electrotechnical Commission, 2008. No. IEC 60068-2-64.
9. SOFTWARE MANSION. *react-native-svg* [online]. [cit. 2026-03-22]. Dostupné z: https://github.com/software-mansion/react-native-svg.
10. ROUSAVY, Marc. *react-native-mmkv* [online]. 2025. Ver. 3.3.0 [cit. 2026-03-09]. Dostupné z: https://github.com/mrousavy/react-native-mmkv.
11. STMICROELECTRONICS. *ANT7-T-M24SR64 Databrief* [online]. 2016. [cit. 2026-03-22]. Dostupné z: https://www.st.com/resource/en/data_brief/ant7-t-m24sr64.pdf.
12. REVTEL TECH. *react-native-nfc-manager* [online]. 2024. [cit. 2026-03-15]. Dostupné z: https://github.com/revtel/react-native-nfc-manager.
13. FORMIDABLELABS. *victory-native* [online]. 2025. [cit. 2026-03-22]. Dostupné z: https://github.com/FormidableLabs/victory-native-xl.
14. RECHARTS. *recharts* [online]. [cit. 2026-03-22]. Dostupné z: https://github.com/recharts/recharts.

Seznam obrázků

Seznam tabulek