STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

HabitTrack – Aplikace pro sledování návyků a zdraví

Miroslav Lubeník

Obor: 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování
Třída: IT4
Školní rok: 2023/2024

Poděkování

poděkování (například vedoucímu práce).

. . . .

Prohlášení

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 4. ledna 2026

.....

podpis autora práce

ABSTRAKT

Tato závěrečná práce se zabývá vytvořením mobilní aplikace pro sledování návyků a zdraví pomocí Flutter frameworku. Aplikace HabitTrack umožňuje uživatelům vytvářet a spravovat vlastní návyky, sledovat jejich denní plnění, zobrazovat statistiky pokroku a personalizovat prostředí aplikace. Aplikace je navržena s důrazem na moderní uživatelské rozhraní, offline fungování a intuitivní ovládání. Implementace využívá lokální SQLite databázi pro ukládání dat, systém notifikací pro připomenutí a centralizovaný design systém pro konzistentní vzhled. Práce obsahuje popis teoretických základů, použité technologie, způsoby řešení a výsledky implementace.

Klíčová slova: Flutter, mobilní aplikace, sledování návyků, SQLite, Dart

ABSTRACT

This final thesis deals with the creation of a mobile application for tracking habits and health using the Flutter framework. The HabitTrack application allows users to create and manage their own habits, track their daily completion, display progress statistics, and personalize the application environment. The application is designed with emphasis on modern user interface, offline functionality, and intuitive operation. The implementation uses a local SQLite database for data storage, a notification system for reminders, and a centralized design system for consistent appearance. The thesis contains description of theoretical foundations, used technologies, solution approaches and implementation results.

Keywords: Flutter, mobile application, habit tracking, SQLite, Dart

Obsah

Seznam obrázků

Úvod

Tato závěrečná práce se zabývá vytvořením mobilní aplikace pro sledování návyků a zdraví pomocí Flutter frameworku. Aplikace HabitTrack poskytuje uživatelům moderní a intuitivní nástroj pro správu jejich denních návyků s důrazem na vizualizaci pokroku a motivaci k dlouhodobé pravidelnosti.

0.1 Náhled do řešené problematiky

Sledování návyků je důležitou součástí osobního rozvoje a zdravého životního stylu. Moderní mobilní aplikace mohou uživatelům významně pomoci při budování a udržování pozitivních návyků prostřednictvím vizualizace pokroku, připomenutí a gamifikace. Flutter framework představuje ideální volbu pro vývoj takové aplikace díky své cross-platform kompatibilitě a moderním UI možnostem.

0.2 Zdůvodnění volby problematiky

Volba této problematiky byla motivována potřebou praktické aplikace pro osobní použití, možností využít moderní technologie (Flutter, Dart), zájmem o vývoj mobilních aplikací s komplexní funkcionalitou a snahou vytvořit aplikaci s moderním designem a uživatelsky přívětivým rozhraním.

0.3 Předem definované cíle práce

Hlavní cíle této závěrečné práce jsou:

- Navrhnout a implementovat mobilní aplikaci pro sledování návyků
- Vytvořit intuitivní uživatelské rozhraní s moderním designem
- Implementovat systém pro ukládání dat s offline podporou
- Zajistit personalizaci aplikace (dark mode, barvy, profil)

- Vytvořit systém statistik a vizualizace pokroku
- Implementovat notifikace pro připomenutí o návycích

0.4 Stručné uvedení obsahu následujících kapitol

Práce je rozdělena do následujících kapitol:

Kapitola 1 – Teoretická a metodická východiska popisuje teoretické základy pro vývoj mobilních aplikací, Flutter framework a související technologie.

Kapitola 2 – Využité technologie představuje konkrétní technologie, nástroje a knihovny použité při vývoji aplikace.

Kapitola 3 – Způsoby řešení a použité postupy popisuje návrh architektury aplikace, databázového schématu a implementační postupy.

Kapitola 4 – Výsledky řešení, výstupy, uživatelský manuál prezentuje finální aplikaci, její funkcionality a uživatelský manuál.

Závěr shrnuje výsledky práce a navrhuje možná vylepšení.

Kapitola 1

Teoretická a metodická východiska

1.1 Flutter Framework

Flutter je open-source UI framework vyvinutý společností Google pro vytváření nativních aplikací pro mobilní, webové a desktopové platformy z jediného kódu. Flutter používá programovací jazyk Dart.

1.1.1 Výhody Flutteru

Volba Flutter frameworku byla motivována následujícími výhodami:

- **Cross-platform vývoj** – jedna codebase pro Android, iOS, Web a Desktop
- **Vysoký výkon** – kompilace do nativního kódu (AOT compilation)
- **Bohatá widget knihovna** – Material Design a Cupertino widgety
- **Hot reload** – rychlý vývoj s okamžitou aktualizací změn
- **Silná komunita** – rozsáhlá dokumentace a podpora
- **Bezplatný a open-source** – žádné licenční poplatky

1.2 Dart programovací jazyk

Dart je objektově orientovaný programovací jazyk vyvinutý společností Google. Je navržen pro vývoj aplikací, které běží na klientovi i serveru.

1.2.1 Klíčové vlastnosti Dartu

- Statická typová kontrola pro lepší bezpečnost kódu

- Asynchronní programování pomocí `async/await`
- Garbage collection pro automatickou správu paměti
- Mixiny pro vícenásobnou dědičnost
- Streamy pro zpracování datových toků

1.3 SQLite databáze

SQLite je lehká relační databázová knihovna implementovaná jako knihovna C. V aplikaci je použita přes balíček `sqflite` pro lokální ukládání dat.

1.3.1 Výhody SQLite

Volba SQLite byla motivována:

- **Offline fungování** – data jsou uložena lokálně na zařízení
- **Rychlý přístup** – přímý přístup k datům bez síťového připojení
- **Snadná implementace** – jednoduché SQL dotazy
- **Malá velikost** – minimální nároky na paměť
- **Bez serveru** – nevyžaduje databázový server

1.4 SharedPreferences

SharedPreferences je mechanismus pro ukládání jednoduchých datových typů (String, int, bool) v key-value formátu. V aplikaci je použit pro ukládání uživatelských nastavení (dark mode, theme color, login status).

1.5 Flutter Local Notifications

Balíček `flutter_local_notifications` umožňuje zobrazovat lokální notifikace na zařízení. V aplikaci je použit pro připomenutí uživatelům o plnění návyků v nastavený čas.

1.6 FL Chart

FL Chart je knihovna pro vytváření grafů a vizualizací v Flutter aplikacích. V aplikaci je použita pro zobrazení statistik a pokroku uživatelů v různých formátech (line charts, bar charts).

1.7 Image Picker

Balíček `image_picker` umožňuje uživatelům vybrat obrázky z galerie nebo pořídit fotografie pomocí kamery. V aplikaci je použit pro výběr profilové fotky uživatele.

1.8 Path Provider

Balíček `path_provider` poskytuje přístup k systémovým cestám pro ukládání souborů. V aplikaci je použit pro ukládání profilových fotografií uživatelů.

1.9 Zdůvodnění výběru technologií

Výběr technologií byl proveden s ohledem na požadavek na cross-platform kompatibilitu, potřebu offline fungování aplikace, snahu o moderní a atraktivní uživatelské rozhraní, dostupnost a kvalitu dokumentace a aktivitu komunity a podporu.

Všechny použité technologie jsou open-source a bezplatné, což umožňuje volné použití a distribuci aplikace.

Kapitola 2

Využité technologie

2.1 Flutter Framework

Flutter je open-source UI framework vyvinutý společností Google pro vytváření nativních aplikací pro mobilní, webové a desktopové platformy z jediného kódu. Flutter používá programovací jazyk Dart a kompiluje aplikace do nativního kódu, což zajišťuje vysoký výkon.

2.1.1 Zdůvodnění výběru

Volba Flutter frameworku byla motivována možností vytvářet aplikace pro více platforem z jediného kódu, vysokým výkonem díky kompilaci do nativního kódu, bohatou widget knihovnou, rychlým vývojem pomocí hot reload a silnou komunitou s rozsáhlou dokumentací.

2.2 Dart programovací jazyk

Dart je objektově orientovaný programovací jazyk vyvinutý společností Google. Je navržen pro vývoj aplikací, které běží na klientovi i serveru. Dart poskytuje statickou typovou kontrolu, asynchronní programování pomocí `async/await`, automatickou správu paměti (garbage collection) a podporu pro mixiny a streamy.

2.3 SQLite databáze

SQLite je lehká relační databázová knihovna implementovaná jako knihovna C. V aplikaci je použita přes balíček `sqflite` pro lokální ukládání dat. SQLite byla zvolena pro svou jednoduchost, rychlost, offline fungování a minimální nároky na paměť.

2.4 SharedPreferences

SharedPreferences je mechanismus pro ukládání jednoduchých datových typů (String, int, bool) v key-value formátu. V aplikaci je použit pro ukládání uživatelských nastavení jako je dark mode, barva motivu a stav přihlášení.

2.5 Flutter Local Notifications

Balíček `flutter_local_notifications` umožňuje zobrazovat lokální notifikace na zařízení. V aplikaci je použit pro připomenutí uživatelům o plnění návyků v nastavený čas. Tento balíček byl zvolen pro svou jednoduchost a dobrou podporu na Android i iOS platformách.

2.6 FL Chart

FL Chart je knihovna pro vytváření grafů a vizualizací v Flutter aplikacích. V aplikaci je použita pro zobrazení statistik a pokroku uživatelů v různých formátech (line charts, bar charts). Knihovna byla zvolena pro svou flexibilitu a možnost vytvářet moderní a atraktivní grafy.

2.7 Image Picker

Balíček `image_picker` umožňuje uživatelům vybrat obrázky z galerie nebo pořídit fotografie pomocí kamery. V aplikaci je použit pro výběr profilové fotky uživatele. Balíček poskytuje jednoduché API a dobrou podporu pro obě platformy.

2.8 Path Provider

Balíček `path_provider` poskytuje přístup k systémovým cestám pro ukládání souborů. V aplikaci je použit pro ukládání profilových fotografií uživatelů na správné místo v systému souborů.

2.9 Zdůvodnění výběru technologií

Výběr technologií byl proveden s ohledem na požadavek na cross-platform kompatibilitu, potřebu offline fungování aplikace, snahu o moderní a atraktivní uživatelské rozhraní,

dostupnost a kvalitu dokumentace a aktivitu komunity a podporu. Všechny použité technologie jsou open-source a bezplatné, což umožňuje volné použití a distribuci aplikace.

Kapitola 3

Způsoby řešení a použité postupy

3.1 Návrh architektury aplikace

Aplikace je strukturována podle principů čisté architektury s oddělením vrstev. Struktura projektu je následující:

3.1.1 Struktura projektu

- `lib/screens/` – obrazovky aplikace (UI vrstva)
- `lib/services/` – business logika a služby (databáze, notifikace)
- `lib/widgets/` – znovupoužitelné widgety
- `lib/styles/` – centralizované styly a design systém

3.1.2 Design Patterns

V aplikaci jsou použity následující návrhové vzory:

- **Singleton** – pro DatabaseHelper a NotificationService (zajišťuje jedinou instanci)
- **State Management** – pomocí StatefulWidget a setState pro lokální stav
- **Repository Pattern** – pro práci s databází (abstrakce datové vrstvy)
- **Factory Pattern** – pro vytváření widgetů a stylů

3.2 Návrh databáze

Databáze obsahuje následující tabulky:

3.2.1 Schéma databáze

- `users` – informace o uživatelích (id, email, nickname, password_hash, profile_photo_path)
- `habits` – definice návyků (id, user_id, name, description, color, icon, daily_target, atd.)
- `habit_logs` – záznamy o plnění návyků (id, habit_id, date, completed)
- `calendar_notes` – poznámky k datům (id, user_id, date, note)
- `achievements` – odemčené achievementy (id, user_id, habit_id, type, unlocked_at)

3.2.2 Migrace databáze

Databáze podporuje migrace pro plynulé aktualizace schématu. Při změně verze databáze se automaticky provedou potřebné změny (přidání sloupců, vytvoření nových tabulek).

3.3 Návrh uživatelského rozhraní

UI bylo navrženo s důrazem na:

- Minimalistický a moderní design
- Konzistentní barevné schéma (gradient Pink → Orange)
- Animace pro lepší uživatelský zážitek
- Dark mode podpora
- Responzivní layout pro různé velikosti obrazovek

3.4 Implementační postupy

3.4.1 Inicializace aplikace

Aplikace začíná v souboru `main.dart`, kde se inicializují klíčové služby (databáze, notifikace) a načítají uživatelská nastavení.

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await DatabaseHelper.instance.database;
  await NotificationService.instance.initialize();
```

```
5
6   final prefs = await SharedPreferences.getInstance();
7   final savedEmail = prefs.getString('user_email');
8   final isDarkMode = prefs.getBool('dark_mode') ?? false;
9   final themeColor = prefs.getString('theme_color') ?? '#009688';
10
11  runApp(HabitTrackApp(
12    isLoggedIn: savedEmail != null,
13    isDarkMode: isDarkMode,
14    themeColor: themeColor,
15  ));
16 }
```

Listing 3.1: Inicializace aplikace v main.dart

3.4.2 Databázová vrstva

Databázová vrstva je implementována pomocí Singleton patternu v třídě `DatabaseHelper`. Všechny databázové operace jsou asynchronní a používají SQL dotazy.

```
1  class DatabaseHelper {
2    static final DatabaseHelper instance = DatabaseHelper._init();
3    static Database? _database;
4
5    DatabaseHelper._init();
6
7    Future<Database> get database async {
8      if (_database != null) return _database!;
9      _database = await _initDB('habittrack.db');
10     return _database!;
11   }
12
13   Future<Database> _initDB(String filePath) async {
14     final dbPath = await getDatabasesPath();
15     final path = join(dbPath, filePath);
16     return await openDatabase(path, version: 6,
17                               onCreate: _createDB,
18                               onUpgrade: _onUpgrade);
19   }
20 }
```

Listing 3.2: Implementace Singleton patternu pro DatabaseHelper

3.4.3 Správa návyků

Hlavní obrazovka zobrazuje seznam návyků uživatele s možností označit je jako splněné. Při dokončení návyku se zobrazí confetti efekt pro vizuální zpětnou vazbu.

```
Future<void> _loadHabits() async {
  final data = await DatabaseHelper.instance.getHabits(userId);
  final todayStr =
      DateTime.now().toIso8601String().split('T').first;

  Map<int, int> todayCount = {};
  Map<int, int> streaks = {};

  for (var habit in data) {
    final habitId = habit['id'] as int;
    final count = await DatabaseHelper.instance
        .getDailyCompletionCount(habitId, todayStr);
    todayCount[habitId] = count;

    final streak = await DatabaseHelper.instance
        .getHabitStreak(habitId);
    streaks[habitId] = streak;
  }

  setState(() {
    habits = data;
    completedTodayCount = todayCount;
    habitStreaks = streaks;
  });
}
```

Listing 3.3: Načítání návyků a jejich statistik

3.4.4 Stylování a design systém

Aplikace používá centralizovaný design systém s oddělenými soubory pro barvy, gradienty, textové styly a dekorace. To umožňuje snadnou údržbu a konzistenci vzhledu.

```
class AppColors {
  // Prim rn  barvy
  static const Color primaryOrange = Color(0xFFFF9800);
  static const Color primaryPink = Color(0xFFE91E63);
  static const Color primaryPurple = Color(0xFF9C27B0);

  // Barvy pro~n vyky
```

```
  static const Color habitPink = Color(0xFFE91E63);
  static const Color habitRed = Color(0xFFF44336);
  static const Color habitOrange = Color(0xFFFF9800);
  // ... dal barvy
}
```

Listing 3.4: Definice barev v design systému

3.4.5 Animace a efekty

Aplikace obsahuje různé animace pro zlepšení uživatelského zážitku:

- AnimatedGradientBackground – animované gradientní pozadí
- ConfettiEffect – confetti při dokončení návyku
- BounceAnimation – bounce efekt pro seznamy
- ScaleOnTap – scale efekt při kliknutí
- SkeletonLoader – skeleton loading screens

3.5 Způsoby testování

Aplikace byla testována na několika úrovních:

3.5.1 Manuální testování

Aplikace byla manuálně testována na:

- Android emulátoru (různé verze Android)
- Fyzickém Android zařízení

3.5.2 Testované funkcionality

- Autentizace (registrace, přihlášení)
- Správa návyků (vytváření, úprava, mazání)
- Označení návyku jako splněného
- Zobrazení statistik a grafů

- Notifikace a připomenutí
- Dark mode přepínání
- Personalizace (barvy, profilová fotka)

3.6 Řešení problémů

Během vývoje byly řešeny následující problémy:

3.6.1 Přetečení layoutu na kartách návyků

Problém: Ikonky a text se překrývaly na kartách návyků při menších obrazovkách.

Řešení: Zvýšena výška karet a upraveno rozložení prvků s lepším paddingem.

3.6.2 Bílá obrazovka před splash screenem

Problém: Na Android zařízeních se zobrazovala bílá obrazovka před načtením Flutter splash screenu.

Řešení: Upraveny native Android launch screen soubory (`launch_background.xml`) pro zobrazení gradientního pozadí.

3.6.3 Type mismatch při dark mode

Problém: Některé barvy byly definovány jako funkce vyžadující BuildContext, ale používaly se jako statické hodnoty.

Řešení: Rozděleny barvy na statické (pro gradient pozadí) a context-aware (pro karty a text).

Kapitola 4

Výsledky řešení, výstupy, uživatelský manuál

4.1 Splněné cíle práce

Všechny stanovené cíle práce byly úspěšně splněny:

- Navržena a implementována mobilní aplikace pro sledování návyků
- Vytvořeno intuitivní uživatelské rozhraní s moderním designem
- Implementován systém pro ukládání dat s offline podporou
- Zajištěna personalizace aplikace (dark mode, barvy, profil)
- Vytvořen systém statistik a vizualizace pokroku
- Implementovány notifikace pro připomenutí o návycích

4.2 Hlavní funkcionality aplikace

4.2.1 Autentizace

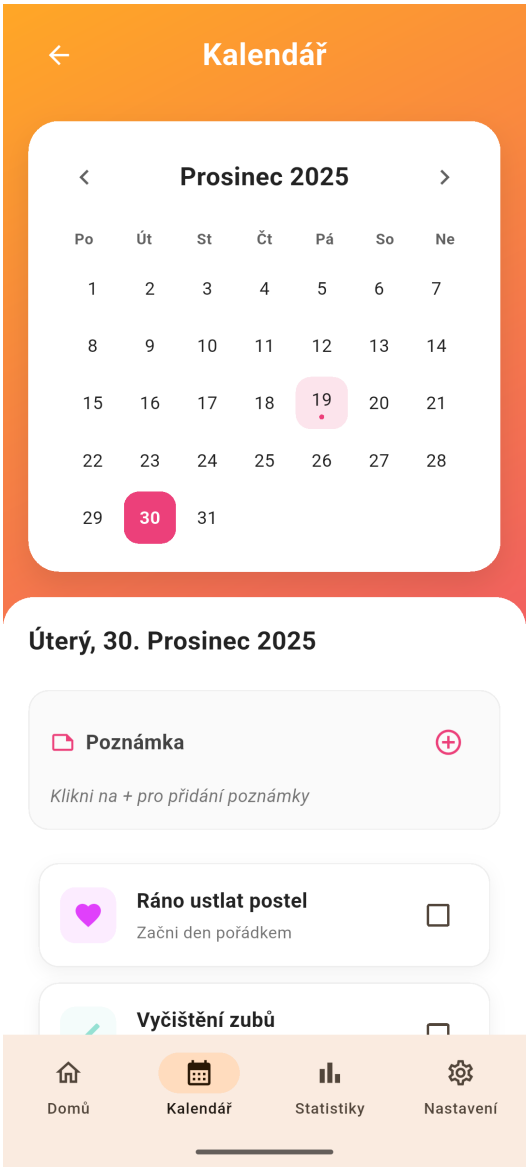
Aplikace obsahuje systém autentizace s registrací, přihlášením a ukládáním přihlašovacích údajů.

4.2.2 Správa návyků

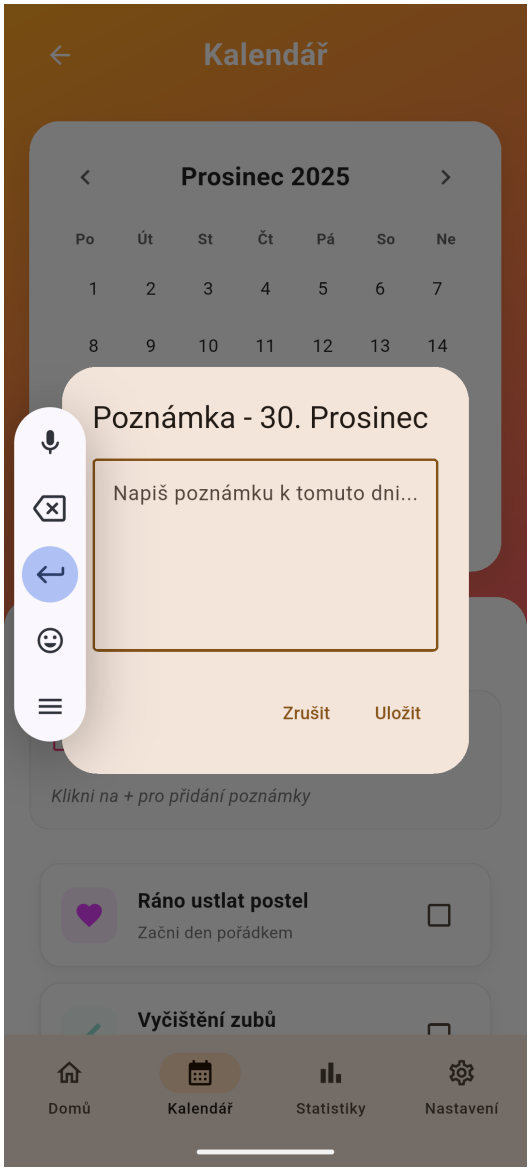
Uživatel může vytvářet, upravovat a mazat návyky s vlastními parametry (název, popis, barva, ikona, denní cíl), označovat návyky jako splněné pro daný den a zobrazovat progress a streak.

4.2.3 Kalendář

Kalendář zobrazuje historii plnění návyků pro jednotlivé dny, umožňuje přidat poznámky k datům a zobrazuje vizuální indikace splněných/nesplněných návyků.



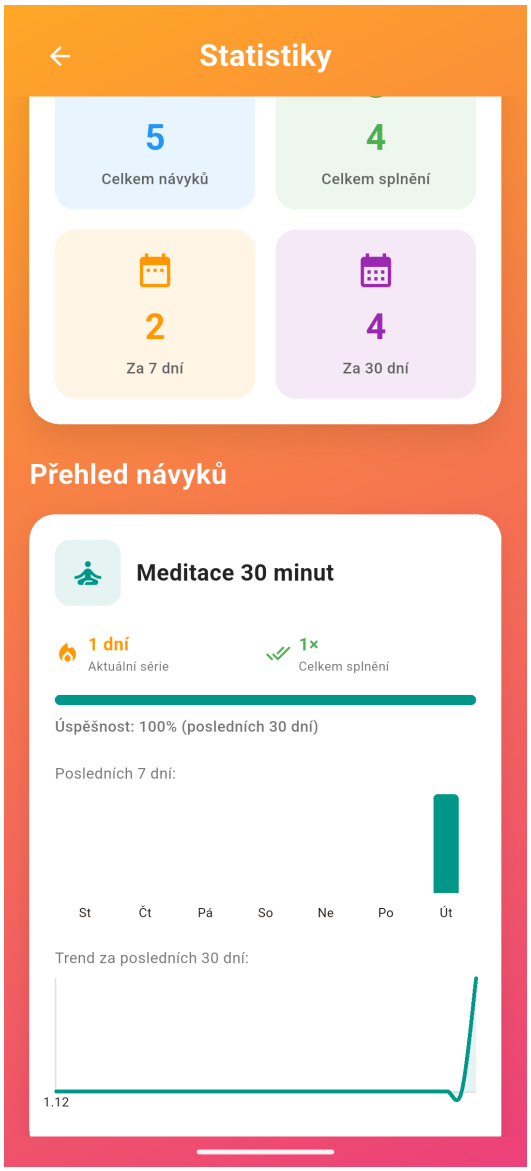
Kalendář s historií plnění návyků



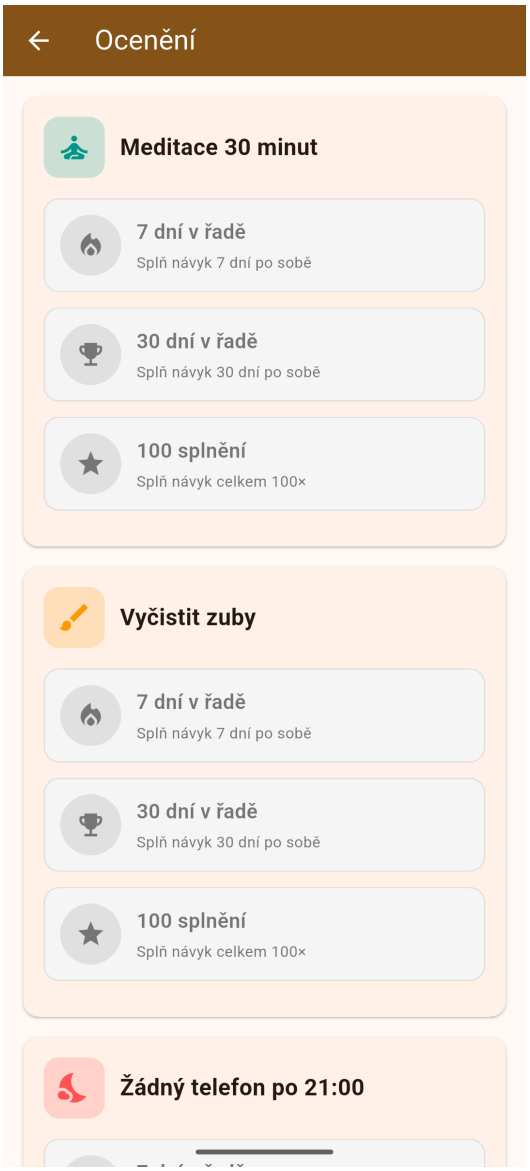
Přidání poznámky do kalendáře

4.2.4 Statistiky a Achievements

Statistiky obsahují týdenní a měsíční přehledy, grafy pokroku a počty splnění návyků. Systém achievementů motivuje uživatele (7, 30, 100 dní v řadě).



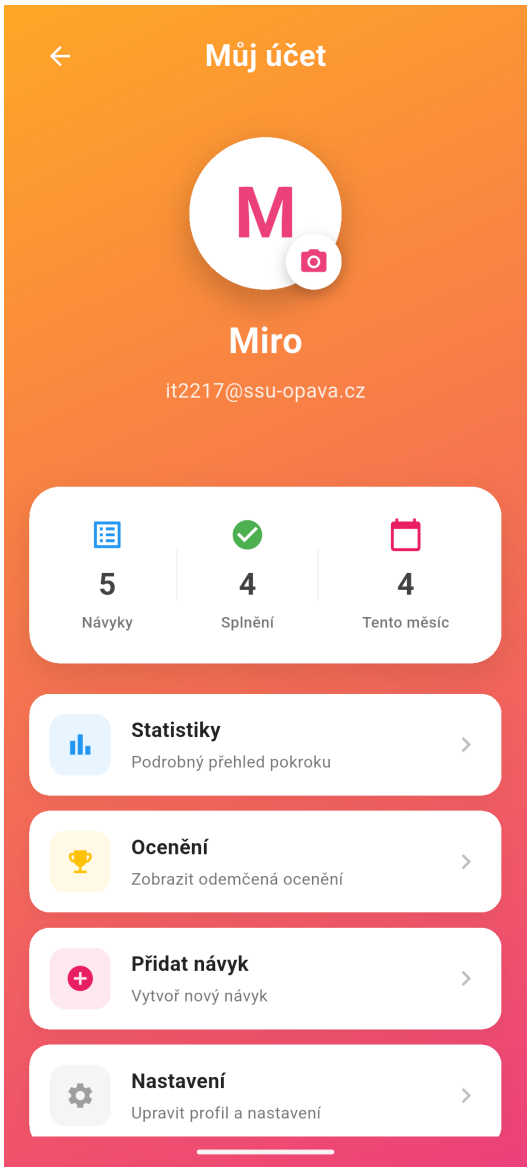
Statistiky a grafy pokroku



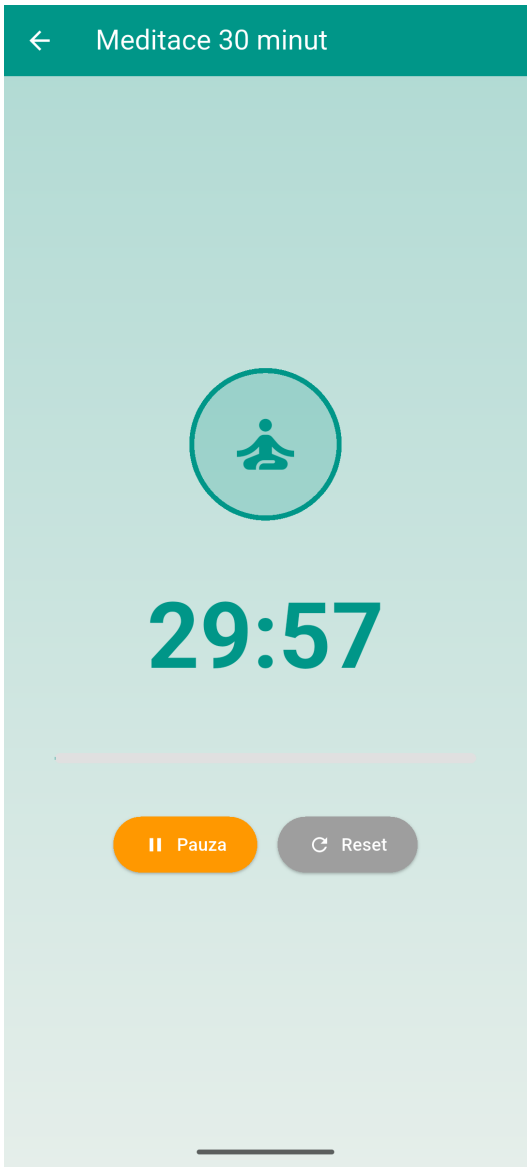
Systém achievementů

4.2.5 Personalizace a Notifikace

Uživatel může přepínat mezi světlým a tmavým režimem, měnit barvu motivu a nastavit profil. Systém notifikací umožňuje naplánovat připomenutí pro návyky.



Profil uživatele a nastavení

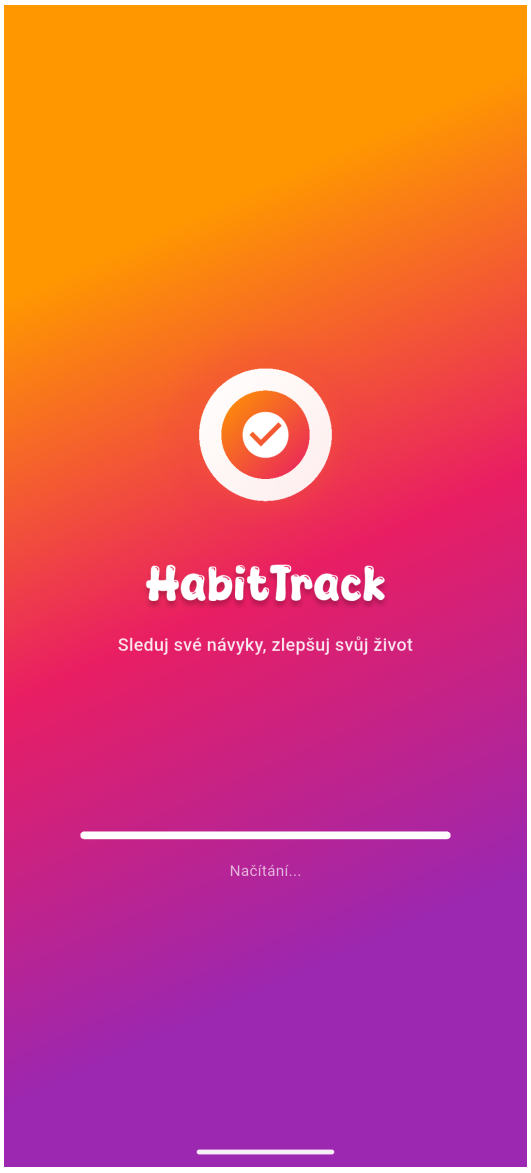


Časovač pro návyky s časovým limitem

4.3 Uživatelský manuál

4.3.1 První spuštění a přihlášení

Po spuštění aplikace se zobrazí splash screen s logem aplikace. Následně se uživatel dostane na obrazovku přihlášení nebo registrace. Po úspěšném přihlášení se dostane na hlavní obrazovku.



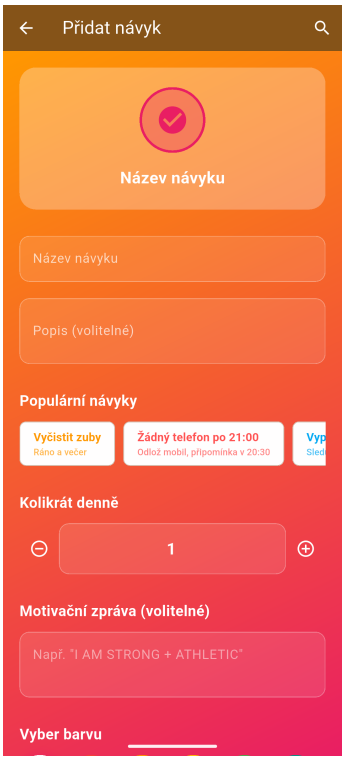
Loading screen s logem aplikace

Hlavní obrazovka s návyky

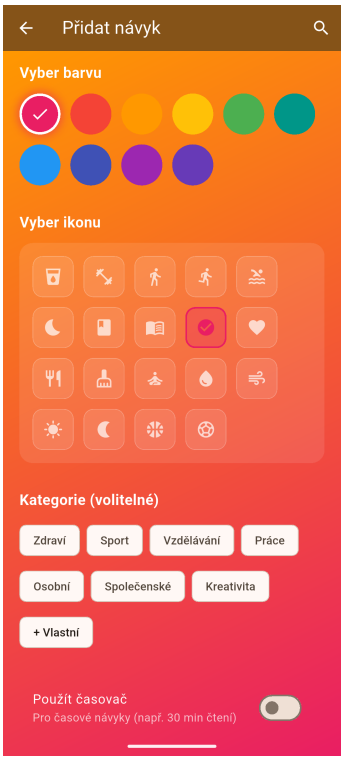
4.3.2 Vytvoření návyku

Proces přidání nového návyku probíhá ve třech krocích:

1. Na hlavní obrazovce klikněte na tlačítko "+"(vpravo nahoře)
2. Vyplňte formulář (název, popis, barva, ikona, denní cíl)
3. Volitelně nastavte připomenutí
4. Klikněte na "Vytvořit návyk"



Krok 1: základní informace



Krok 2: výběr ikony

Krok 3: dokončení

4.3.3 Základní funkce

Na hlavní obrazovce lze kliknutím na kartu návyku označit návyk jako splněný. V dolní navigaci lze přepínat mezi obrazovkami pro zobrazení statistik a nastavení profilu.

4.4 Technické specifikace

Aplikace vyžaduje Android 5.0 (API level 21) nebo vyšší a minimálně 50 MB volného místa. Podporuje offline fungování, dark mode, notifikace, profilové fotky a statistiky s grafy.

Závěr

Tato závěrečná práce úspěšně splnila všechny stanovené cíle. Byla vytvořena plně funkční mobilní aplikace HabitTrack pro sledování návyků a zdraví pomocí Flutter frameworku.

4.5 Shrnutí výsledků

Aplikace obsahuje všechny plánované funkcionality: systém autentizace, správu návyků, sledování denního plnění, kalendář s historií, statistiky, systém achievementů, personalizaci, notifikace a offline fungování s lokálním ukládáním dat.

4.6 Hodnocení splnění cíle práce

Všechny stanovené cíle práce byly úspěšně splněny. Aplikace poskytuje uživatelům moderní a intuitivní nástroj pro správu jejich denních návyků s důrazem na vizualizaci pokroku.

4.7 Možnost uplatnění řešení v praxi

Aplikace může být použita jako osobní nástroj pro sledování návyků, pro distribuci v Google Play Store a Apple App Store, jako základ pro další rozšíření nebo pro vzdělávací účely.

4.8 Nastínění možného dalšího rozvoje

Pro budoucí vývoj by bylo možné implementovat cloud synchronizaci, sociální funkce, export dat, více jazyků, widgety, health integraci a AI doporučení.

4.9 Závěr

Aplikace HabitTrack představuje funkční a moderní řešení pro sledování návyků. Díky použití Flutter frameworku je aplikace cross-platform a může být snadno rozšířena o další funkcionality. Implementace čisté architektury a centralizovaného design systému usnadňuje údržbu a budoucí rozvoj aplikace.

Seznam použitých informačních zdrojů

Literatura

Seznam příloh

- Příloha č. 1: Titulní list
- Příloha č. 2: Čestné prohlášení
- Příloha č. 3: Poděkování
- Příloha č. 4: Významné části kódu
- Screenshoty aplikace jsou integrované do kapitoly 4 (Výsledky řešení, výstupy, uživatelský manuál)

Příloha A

Významné části kódu

A.1 Významné části kódu

A.1.1 DatabaseHelper – Singleton Pattern

```
class DatabaseHelper {
  static final DatabaseHelper instance = DatabaseHelper._init();
  static Database? _database;

  DatabaseHelper._init();

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDB('habittrack.db');
    return _database!;
  }
}
```

Listing A.1: Implementace Singleton patternu pro DatabaseHelper

A.1.2 Animovaný gradient pozadí

```
class AnimatedGradientBackground extends StatefulWidget {
  const AnimatedGradientBackground({super.key});

  @override
  State<AnimatedGradientBackground> createState()
      => _AnimatedGradientBackgroundState();
}

class _AnimatedGradientBackgroundState
```

```
    extends State<AnimatedGradientBackground>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(
      duration: const Duration(seconds: 3),
      vsync: this,
    )..repeat();
  }

  @override
  Widget build(BuildContext context) {
    return AnimatedBuilder(
      animation: _controller,
      builder: (context, child) {
        return Container(
          decoration: BoxDecoration(
            gradient: LinearGradient(
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
              colors: [
                AppColors.primaryOrange,
                AppColors.primaryPink,
              ],
              stops: [
                0.0 + _controller.value * 0.1,
                1.0 - _controller.value * 0.1,
              ],
            ),
          ),
        );
      },
    );
  }
}
```

Listing A.2: Implementace animovaného gradientního pozadí

A.1.3 Confetti efekt

```
class ConfettiEffect extends StatefulWidget {
```

```
  final Color color;
  final VoidCallback? onComplete;

  const ConfettiEffect({
    super.key,
    required this.color,
    this.onComplete,
  });

  @override
  State<ConfettiEffect> createState() => _ConfettiEffectState();
}

class _ConfettiEffectState extends State<ConfettiEffect> {
  final List<Particle> _particles = [];
  final Random _random = Random();

  @override
  void initState() {
    super.initState();
    _generateParticles();
  }

  void _generateParticles() {
    for (int i = 0; i < 50; i++) {
      _particles.add(Particle(
        x: _random.nextDouble(),
        y: _random.nextDouble(),
        vx: (_random.nextDouble() - 0.5) * 0.02,
        vy: _random.nextDouble() * 0.03 + 0.01,
        size: _random.nextDouble() * 5 + 3,
      ));
    }
  }
}
```

Listing A.3: Implementace confetti efektu při dokončení návyku