VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Aplikovaná informatika

VYUŽÍTÍ JAZYKA KOTLIN PŘI TVORBĚ MOBILNÍCH APLIKACÍ

Bakalářská práce

Autor práce: Tomáš Březina

Vedoucí práce: Ing. Marek Musil

Jihlava 2026

Vysoká škola polytechnická Jihlava

Tolstého 16, 586 01 Jihlava

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	**Tomáš Březina**
Studijní program:	Aplikovaná informatika
Garant studijního programu:	Ing. Lenka Kuklišová Pavelková, Ph.D.
Název práce:	**Využití jazyka Kotlin při tvorbě mobilních aplikací**
Vedoucí práce:	Ing. Marek Musil
Cíl práce:	Cílem práce je představení jazyka Kotlin používaného pro vývoj mobilních aplikací na platformě Android. Součástí práce bude posouzení aktuálního stavu, diskuze k problematice a shrnutí dostupné literatury. Bude vytvořena mobilní aplikace pro zkoušení slovíček. Tato aplikace bude obsahovat rozšiřující funkce ve srovnání s již existujícími aplikacemi. Vytvořená aplikace bude ukázkou použitelnou ve výuce zaměřené na tvorbu mobilních aplikací.

Abstrakt

Bakalářská práce se zabývá návrhem a implementací mobilní aplikace pro operační systém Android, která slouží ke zkoušení anglické slovní zásoby. Teoretická část práce se věnuje charakteristice mobilní platformy Android a modernímu programovacímu jazyku Kotlin, který byl zvolen jako primární nástroj pro vývoj. Dále jsou rozebrány doporučené architektonické postupy, konkrétně vzor Model-View-ViewModel a možnosti lokálního ukládání dat pomocí knihovny Room Persistence Library. Práce dále popisuje kompletní proces vývoje aplikace - analýzu trhu a specifikaci funkčních požadavků, návrh uživatelského rozhraní a nakonec samotnou implementaci. Výsledkem je funkční aplikace, která uživateli umožňuje učení slovíček podle tematických kategoriích. Aplikace nabízí režim zkoušení formou výběru z možností se zpětnou vazbou a sledováním úspěšnosti uživatele. Hlavním přínosem práce je vytvoření intuitivního nástroje, který klade důraz na lokální správu dat a plynulé uživatelské prostředí.

Klíčová slova

Android; Jetpack Compose; Kotlin; Mobilní aplikace; MVVM; Room

Abstract

Bachelor’s thesis focuses on the design and implementation of a mobile application for the Android operating system, designed to test English vocabulary. The theoretical section of the thesis examines the characteristics of the Android mobile platform and the modern programming language Kotlin, which was chosen as the primary development tool. It also examines recommended architectural practices, specifically the Model-View-ViewModel pattern, and options for local data storage using the Room Persistence Library.

Furthermore, it describes the complete application development process, from market analysis and functional requirements specification, through user interface design, to the actual implementation. The result is a functional application that allows users to learn vocabulary organized by thematic categories. The application offers a quiz mode featuring multiple-choice questions with feedback and tracking of the user’s success. The main contribution of this work is the creation of an intuitive tool that emphasizes local data management and a seamless user experience.

Keywords

Android; Jetpack Compose; Kotlin; Mobile application; MVVM; Room

Prohlašuji, že předložená bakalářská práce je původní a zpracoval/a jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil/a autorská práva (ve smyslu zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, v platném znění, dále též „**AZ**“).

Byl/a jsem seznámen/a s tím, že na mou bakalářskou práci se plně vztahuje **AZ**, zejména § 60 (školní dílo).

Podle § 47b zákona o vysokých školách souhlasím se zveřejněním své práce podle Směrnice pro vedení, vypracování a zveřejňování závěrečných prací na VŠPJ, a to bez ohledu na výsledek obhajoby.

Beru na vědomí, že VŠPJ má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom/a toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem VŠPJ, která má právo ode mě požadovat přiměřený příspěvek na úhradu nákladů, vynaložených vysokou školou na vytvoření díla (až do jejich skutečné výše), z výdělku dosaženého v souvislosti s užitím díla či poskytnutím licence.

V Jihlavě dne 14. dubna 2026

…….

Podpis studenta/ky

Poděkování

Děkuji vedoucímu práce Ing. Markovi Musilovi za odborné vedení, ochotu a připomínky k práci. Dále chci poděkovat svému kamarádovi Bc. Alexandru Škopkovi, za pomoc s otestováním funkčnosti aplikace na svém mobilním zařízení.

Obsah

Seznam obrázků

Seznam zkratek

AOSP	Android Open Source (projekt zdrojového kódu systému android)
AOT	Ahead-of-Time (předběžná kompilace kódu před spuštěním aplikace)
API	Application Programming Interface (rozhraní pro programování aplikací)
ART	Android Runtime (běhové prostředí Androidu pro vykonávání aplikací)
DAO	Data Acces Object (objekt zajišťující přístup k databázovým operacím)
DEX	Dalvik Executable (formát bytekódu používaný v systému Android)
GC	Garbage Collection (mechanismus automatického uvolňování paměti)
HAL	Hardware Abstraction Layer (vrstva abstrakce hardwaru)
JIT	Just-in-Time (kompilace kóduza běhu aplikace)
KTX	Kotlin Extensions (rozčíření Android knihoven pro jazyk Kotlin)
MVVM	Model-View-ViewModel (architektonický návrhový vzor)
SQL	Structured Query Language (strukturovaný dotazovací jazyk)
SQLite	Structured Query Language Lite (odlehčená verze jazyka SQL)
UI	User Interface (uživatelské rozhraní)

Úvod

Mobilní aplikace se staly běžnou součástí každodenního života a významně ovlivňují způsob, jakým lidé získávají nové znalosti. V oblasti vzdělávání se stále více prosazují aplikace zaměřené na samostudium, které často využívají prvky gamifikace[1] a mikrovzdělávání, čímž podporují pravidelnost učení a motivaci uživatelů. Operační systém Android je v současnosti nejrozšířenějším na světě a nachází své uplatnění nejen na chytrých telefonech, ale také na tabletech a dalších chytrých zařízeních. Vývoj aplikací pro uvedenou platformu vyžaduje volbu vhodných technologií, které umožňují vytvářet moderní, přehledná a efektivní řešení.

Cílem bakalářské práce je návrh a implementace mobilní aplikace pro platformu Android s použitím programovacího jazyka Kotlin. Aplikace je zaměřena na zkoušení slovíček z českého jazyka do anglického. Klade důraz na jednoduchost, přehlednost a možnost používání bez nutnosti připojení k internetu. Hlavní myšlenkou je vytvořit nástroj, který umožní rychlé a efektivní procvičování pomocí krátkých testovacích úloh a zároveň poskytne uživateli přehled o jeho pokroku. Výsledná aplikace je určena především pro uživatele, kteří mají obtíže se zapamatováním slovní zásoby. Volba tématu vychází z vlastní zkušenosti, kdy při delší absenci používání anglického jazyka dochází ke zmíněným obtížím, a ze snahy vytvořit jednoduchý a efektivní nástroj pro zkoušení slovíček.

Práce je rozdělena do čtyř hlavních částí. První část se věnuje teoretickým východiskům a literární rešerši, kde jsou představeny klíčové technologie a principy využité při vývoji aplikace. Druhá část obsahuje analýzu stávajících výukových aplikací, specifikaci požadavků, návrh architektury, uživatelského rozhraní a databázového schématu. Třetí část se zaměřuje na samotnou implementaci aplikace. Poslední část je věnována testování aplikace a zhodnocení její funkčnosti.

[1] Gamifikace využívá herní principy a mechanismy v neherním kontextu ke zvýšení zapojení a motivace uživatelů.

1 Teoretická východiska a literární rešerše

První kapitola slouží jako teoretický základ pro návrh a realizaci celého projektu. Zároveň představuje literární rešerši zaměřenou na současný stav vývoje aplikací pro platformu Android. Cílem kapitoly je představit klíčové technologie a přístupy používané při vývoji mobilních aplikací a zasadit je do kontextu moderního vývoje.

Na základě dostupných zdrojů a oficiální dokumentace lze říci, že vývoj Android aplikací se v posledních letech výrazně posunul, zejména v oblasti používaných nástrojů a architektur. Společnost Google v současnosti prosazuje přístup založený na využití jazyka Kotlin, architektury Model-View-ViewModel (MVVM) a knihoven Android Jetpack, které tvoří základ moderního vývoje aplikací.

Kapitola nejprve představí platformu Android a její architekturu. Následně se zaměří na programovací jazyk Kotlin, který je dnes považován za standard pro vývoj Android aplikací. Dále bude popsána doporučená architektura aplikací, zejména model MVVM, a taky dostupné databázové řešení se zaměřením na knihovnu Room.

Získané poznatky slouží jako podklad pro návrh vlastní aplikace na procvičování slovní zásoby a zároveň vytvářejí prostor pro následnou diskuzi a shrnutí dostupné literatury.

1.1 Mobilní platforma Android

Mobilní platforma Android představuje operační systém založený na modifikovaném jádře Linuxu, který je vyvíjen a spravován společností Google (Platform Architecture, 2024). Od svého uvedení v roce 2008 se Android stal jedním z nejrozšířenějších operačních systémů pro mobilní zařízení, jako jsou chytré telefony či tablety. Důležitým faktorem jeho rozvoje je projekt Android Open Source Project (AOSP), který umožňuje výrobcům přizpůsobovat systém vlastním potřebám a implementovat ho na různá zařízení (AOSP overview, 2026).

Z pohledu vývoje aplikací je Android založen na vícevrstvé architektuře, která odděluje jednotlivé části systému. Na nejnižší úrovni se nachází linuxové jádro, které zajišťuje komunikaci s hardwarem a správu systémových prostředků. Nad ním se nachází prostředí Android Runtime (ART), které slouží ke spouštění aplikací napsaných v jazycích Java a Kotlin a ovlivňuje jejich výkon (Platform Architecture, 2024). ART využívá kombinaci kompilace Ahead-Of-Time (AOT) a Just-In-Time (JIT), čímž dochází k optimalizaci výkonu a rychlejšímu spuštění aplikací (Android Runtime and Dalvik, 2024; Android 7.0 for Developers, 2024).

Aplikace v prostředí Android jsou tvořeny čtyřmi základními komponentami:

- Aktivity (zajišťují uživatelské rozhraní)
- Služby (úlohy běžící na pozadí)
- Broadcast Receivers (pro reakci na systémové události)
- Content Providers (pro správu a sdílení dat)

Výše uvedené komponenty definují základní stavební bloky aplikace a umožňují její interakci se systémem i ostatními aplikacemi (Application fundamentals, 2025). V současnosti je vývoj aplikací silně ovlivněn knihovnami Android Jetpack, které poskytují sadu nástrojů pro řešení běžných problémů, jako je správa životního cyklu nebo práce s daty. Knihovny jsou

optimalizovány pro jazyk Kotlin a představují doporučený standard moderního vývoje (Getting started with Android Jetpack, 2026).

1.1.1 Architektura systému a Hardware Abstraction Layer (HAL)

Architektura systému Android je navržena jako vícevrstvý zásobník (software stack), který zajišťuje oddělení aplikací od hardwaru a umožňuje jejich bezpečný běh v izolovaném prostředí. Daný přístup, označovaný jako sandboxing, zajišťuje, že každá aplikace běží jako samostatný proces s vlastním uživatelským identifikátorem, čímž je minimalizováno riziko neoprávněného přístupu k datům jiných aplikací (Application Sandbox, 2026; Android security features, 2026).

Celý systém lze rozdělit do několika logických vrstev (Platform Architecture, 2024):

- **Jádro Linuxu (Linux Kernel)**: Základní vrstva systému, která tvoří jádro platformy Android a zajišťuje komunikaci mezi hardwarem zařízení a zbytkem softwarového zásobníku. Linuxové jádro poskytuje nízkoúrovňové služby, jako je správa operační paměti, řízení procesů a práci s vlákny. Zároveň obsahuje ovladače pro jednotlivé hardwarové komponenty zařízení, například displej, fotoaparát nebo bezdrátovou komunikaci. Díky využití jádra Linuxu získává Android stabilní a bezpečný základ pro běh aplikací (Platform Architecture, 2024; Kernel overview, 2026).
- **Vrstva abstrakce hardwaru (HAL)**: Vrstva HAL představuje rozhraní mezi hardwarem zařízení a vyššími vrstvami systému. Poskytuje standardizované API, které umožňuje systému využívat hardwarové komponenty bez nutnosti znát jejich konkrétní implementaci. HAL je tvořena moduly, které implementují rozhraní pro jednotlivé typy zařízení, například fotoaparát nebo audio. Díky této abstrakci mohou výrobci zařízení implementovat podporu pro svůj hardware bez zásahu do vyšších vrstev systému. Pro vývojáře aplikací to znamená, že přístup k hardwaru probíhá jednotným způsobem prostřednictvím systémového frameworku, nezávisle na konkrétním zařízení (Platform Architecture, 2024; HAL overview, 2026).
- **Android Runtime (ART) a nativní knihovny**: Vrstva zajišťující běh aplikací v systému Android a má zásadní vliv na jejich výkon. Zdrojový kód aplikace napsaný v jazyce Kotlin nebo Java je kompilován do bajtkódu (byte code), který je následně převeden do formátu DEX (Dalvik Executable), optimalizovaného pro mobilní zařízení s omezenými prostředky. Prostředí Android Runtime (ART) zajišťuje spuštění a běh aplikace tím, že převádí instrukce ve formátu DEX do nativního kódu a následně je vykonává na procesoru zařízení, čímž dochází k optimalizaci výkonu a rychlejšímu spuštění aplikací. Součástí jsou také nativní knihovny psané v jazycích C a C++, které poskytují základní funkcionalitu systému, například práci s databází prostřednictvím knihovny SQLite (Platform Architecture, 2024; Android Runtime and Dalvik, 2024).
- **Java API Framework**: Vrstva poskytující vývojářům soubor nástrojů a systémových služeb pro tvorbu aplikací. Obsahuje aplikační framework, který zahrnuje komponenty jako Activity Manager, Window Manager nebo systém správy notifikací. Prostřednictvím těchto služeb mohou aplikace komunikovat se systémem a využívat jeho funkce. Pro usnadnění vývoje v jazyce Kotlin poskytuje Google rozšíření Android KTX, která rozšiřují existující API o typické prvky jazyka Kotlin, například extension functions a zjednodušují tak práci s platformou Android (Platform Architecture, 2024; Android KTX, 2026).

- **Systémové a uživatelské aplikace**: Vrstva zahrnující aplikace běžící nad aplikačním frameworkem, a to jak aplikace dodávané se systémem, tak aplikace instalované uživatelem. Android zajišťuje bezpečnost tak, že každé aplikaci přiřazuje unikátní identifikátor a spouští ji ve vlastním procesu v rámci aplikačního sandboxu. Daný přístup omezuje přímý přístup k datům ostatních aplikací a přispívá k celkové bezpečnosti systému (Platform Architecture, 2024; Application sandbox, 2026).

Na základě dostupné dokumentace lze říci, že zvolená architektura umožňuje efektivní integraci moderních jazyků do stávajícího ekosystému. Kotlin je plně interoperabilní s jazykem Java, což umožňuje využití existujících knihoven a nástrojů. Společnost Google v roce 2019 označila Kotlin za preferovaný jazyk pro vývoj Android aplikací (Calling Java from Kotlin, 2026).

1.1.2 Android Runtime (ART) a správa procesů

Klíčovým prvkem pro běh a výkon každé Android aplikace je prostředí Android Runtime (ART). Jedná se o běhové prostředí, které zajišťuje vykonávání aplikací a tvoří vrstvu mezi zkompilovaným kódem aplikace a hardwarem zařízení. Pro aplikaci vyvíjenou v jazyce Kotlin je ART zásadní, protože ovlivňuje plynulost uživatelského rozhraní a efektivitu využití systémových prostředků (Android Runtime and Dalvik, 2024).

Když je aplikace v Kotlinu sestavena, vzniká výstup ve formátu DEX (Dalvik Executable), který obsahuje instrukce optimalizované pro běh na platformě Android. Instrukce jsou následně zpracovávány prostředím ART (Dalvik bytecode, 2024). Na rozdíl od tradiční kompilace přímo do strojového kódu využívá Android hybridní model, který kombinuje dva přístupy ke zpracování kódu:

1. **AOT (Ahead-of-Time)**: Způsob kompilace, který je využíván prostředím ART při instalaci aplikace, kdy dochází k překladu DEX kódu do nativního strojového kódu. Tento přístup přispívá ke zlepšení výkonu aplikace (Android Runtime and Dalvik, 2024).
2. JIT (Just-in-Time): Během běhu aplikace ART dynamicky optimalizuje části kódu pomocí JIT kompilátoru, který doplňuje AOT kompilaci a přispívá ke zlepšení výkonu aplikace (Android 7.0 for Developers, 2024; JIT compiler, 2026).

Další důležitou funkcí ART je automatická správa paměti pomocí mechanismu Garbage Collection. Mechanismus zajišťuje uvolňování paměti objektů, které již nejsou aplikací využívány. V praxi to znamená, že objekty vytvořené aplikací, které již nejsou potřeba, jsou z paměti automaticky odstraněny (Android Runtime and Dalvik, 2024).

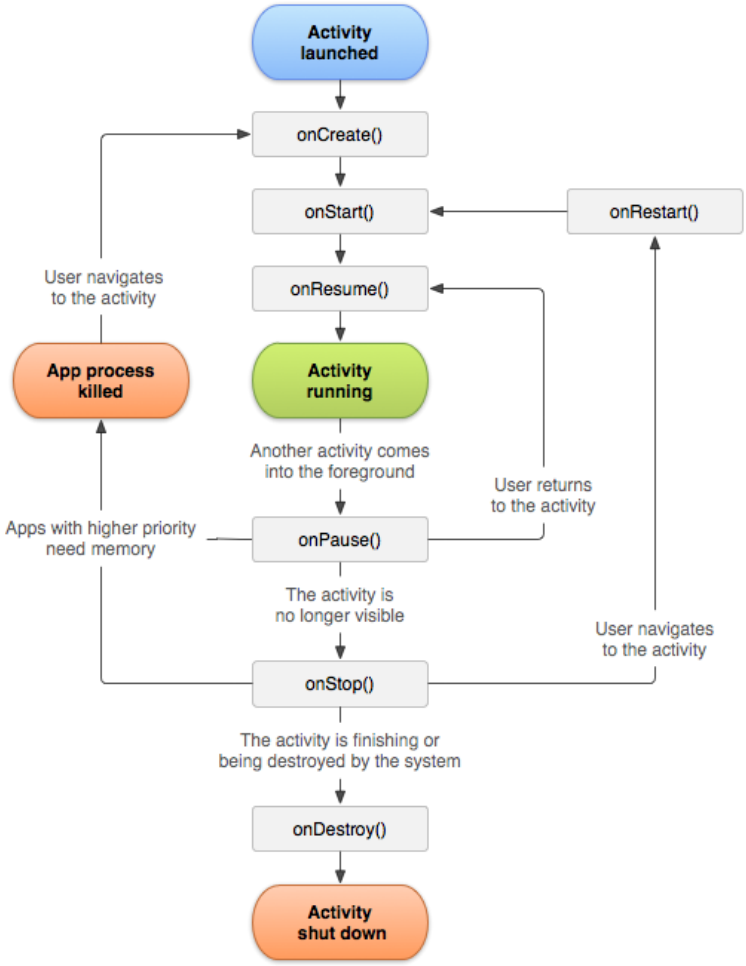
Mobilní zařízení mají ve srovnání s počítači omezené systémové prostředky, zejména operační paměť, a proto je efektivní správa paměti důležitá pro plynulý chod aplikací (Processes and threads overview, 2024).

Moderní implementace GC v prostředí ART je navržena s důrazem na efektivní správu paměti a minimalizaci dopadu na výkon aplikace (Android Runtime and Dalvik, 2024).

1.1.3 Životní cyklus aktivit a správa zdrojů

Životní cyklus aktivit představuje jeden z nejdůležitějších konceptů při vývoji Android aplikací. Aktivita prochází během svého života několika stavy od svého vytvoření až po zánik. Jednotlivé

přechody mezi stavy jsou řízeny operačním systémem v závislosti na interakci uživatele a stavu aplikace (Activity Lifecycle, 2025).



Obr. 1: Životní cyklus aktivity

Zdroj: https://developer.android.com/guide/components/activities/activity-lifecycle

Stav aktivity se mění prostřednictvím tzv. zpětných volání (callback metod), které vývojář v jazyce implementuje. Mezi základní metody patří (Activity Lifecycle, 2025):

- **onCreate()**: Volá se při vytvoření aktivity. Zde probíhá nastavení uživatelského rozhraní a inicializace základních datových struktur.
- **onStart() a onResume()**: Aktivita se stává viditelnou a následně interaktivní v popředí. V daném stavu aplikace přijímá vstupy od uživatele.
- **onPause() a onStop()**: Volají se, když uživatel aplikaci opouští (např. přechod na domovskou obrazovku nebo překrytí jinou aplikací). V následující fázi by aplikace měla omezit náročné výpočty a připravit se na možné ukončení.
- **onDestroy()**: Volá se při ukončení aktivity.

Specifickým rysem Androidu je proces, kdy systém může aktivitu zničit a znovu vytvořit při tzv. změnách konfigurace (např. otočení displeje z vertikální do horizontální polohy). Pokud není takový stav správně ošetřen, může dojít ke ztrátě neuložených dat v uživatelském rozhraní nebo nekonzistentnímu chování aplikace (Activity Lifecycle, 2025).

Pro efektivní řešení zmíněných situací se využívají knihovny Android Jetpack, zejména komponenty pro práci s životním cyklem, které umožňují spravovat data nezávisle na životním cyklu uživatelského rozhraní a zajistit jejich zachování při manipulaci s orientací zařízení (Handling Lifecycles with Lifecycle-Aware Components, 2026).

Správné pochopení životního cyklu aktivit je důležitá pro stabilitu aplikace a přímo ovlivňuje kvalitu uživatelského zážitku.

1.2 Programovací jazyk Kotlin

Programovací jazyk Kotlin je staticky typovaný jazyk vyvinutý společností JetBrains, který je navržen pro vývoj moderních aplikací (FAQ, 2026). V současnosti je považován za preferovaný jazyk pro vývoj mobilních aplikací na platformě Android, což potvrzuje i oficiální dokumentace společnosti Google (Android's Kotlin-first approach, 2026).

Kotlin běží na platformě Java Virtual Machine a je plně interoperabilní s jazykem Java, což umožňuje kombinovat oba jazyky v rámci jednoho projektu bez nutnosti zásadních úprav existujícího kódu (Calling Java from Kotlin, 2026). Kotlin přináší vlastnosti, které vedou ke zvýšení produktivity vývojářů, zjednodušení zápisu kódu a omezení výskytu běžných programátorských chyb (Android's Kotlin-first approach, 2026).

Ve srovnání s jazykem Java přináší Kotlin stručnější a přehlednější syntaxi, která vede ke snížení množství opakujícího se kódu a ke zlepšení čitelnosti aplikací. Důležitou vlastností jazyka je také důraz na bezpečnost (Kotlin for Android, 2026).

Mezi klíčové vlastnosti jazyka Kotlin patří bezpečnost práce s nulovými hodnotami *Null Safety*, interoperabilita s jazykem Java a podpora asynchronního programování prostřednictvím korutin *Coroutines*. Následující vlastnosti jsou podrobněji rozebrány v následujících podkapitolách. Na základě uvedených informací lze Kotlin považovat za moderní a efektivní nástroj pro vývoj mobilních aplikací.

1.2.1 Interoperabilita s jazykem Java a moderní syntaxe

Klíčovým faktorem pro rychlé přijetí jazyka Kotlin v ekosystému Android je jeho plná interoperabilita s jazykem Java, která umožňuje vývojářům využívat existující knihovny a nástroje vytvořené pro jazyk Java a kombinovat oba jazyky v rámci jednoho projektu bez nutnosti zásadních úprav kódu (Calling Java from Kotlin, 2026; FAQ, 2026).

Mezi nejdůležitější vlastnosti, které budou využity i při implementaci aplikace pro zkoušení slovíček, patří:

- **Data Classes**: Speciální třídy určené primárně pro uchovávání dat. Kompilátor u nich automaticky generuje standardní metody jako equals(), hashCode() nebo toString(), což snižuje množství kódu (Data classes, 2026).
- **Extension Functions**: Umožňují rozšiřovat existující třídy o novou funkcionalitu bez nutnosti dědičnosti nebo úpravy původního kódu (Extensions, 2025).
- **Stručnost a čitelnost**: Díky typovému odvozování (type inference) a absenci povinných středníků je kód v Kotlinu kratší a přehlednější, což usnadňuje jeho čitelnost a údržbu (Android's Kotlin-first approach, 2026).

Na základě dostupných dat lze konstatovat, že Kotlin přináší výrazné zlepšení produktivity díky redukci opakujícího se kódu a lepší čitelnosti, což z něj tvoří ideální nástroj pro tvorbu robustních a udržovatelných aplikací.

1.2.2 Kotlin Null Safety a typový systém

Jedním z nejvýznamnějších přínosů jazyka Kotlin je jeho integrovaný systém bezpečnosti proti nulovým referencím, známý jako Null Safety. V tradičních programovacích jazycích, jako je Java, patří výjimka NullPointerException mezi časté chyby v programování. Kotlin zmíněný problém řeší tím, že striktně rozlišuje mezi typy, které mohou obsahovat hodnotu null, a typy, které ji obsahovat nesmí (Null safety, 2025).

V praxi to znamená, že pokud je proměnná definována jako String, nemůže obsahovat hodnotu null. Pokud má být proměnná nulovatelná (nullable variable), musí být explicitně označena symbolem otazníku, tedy jako (String?). Díky tomu jsou potenciální chyby odhaleny již při kompilaci, nikoliv až za běhu aplikace (Null safety, 2025).

K efektivní práci s nulovatelnými typy poskytuje Kotlin několik specifických operátorů:

- **Safe Call operátor**: Umožňuje bezpečný přístup k vlastnostem objektu. Pokud je objekt null, operace se nevykoná a výsledkem je hodnota null.
- **Elvis operátor**: Dovoluje definovat výchozí hodnotu, která se použije v případě, že je výraz null.
- **Not-null assertion**: Operátor, který přetypuje nulovatelnou proměnnou na nenulovatelnou. Pokud je hodnota null, dojde k vyvolání vájimky (Null safety, 2025).

Z pohledu vývoje mobilních aplikací je daný přístup zásadní, protože přispívá ke zvýšení stability aplikace a omezuje výskyt chyb způsobených nulovými referencemi (Null safety, 2025; Android's Kotlin-first approach, 2026).

1.2.3 Kotlin Coroutines a asynchroní programování

Pro moderní mobilní aplikace je nezbytné provádět náročné operace, jako je přístup k lokální databázi nebo síťová komunikace, mimo hlavní vlákno. Pokud by operace probíhaly na hlavním vlákně, došlo by k zablokování uživatelského rozhraní, což by uživatel vnímal jako zamrznutí aplikace (Processes and threads overview, 2024). Kotlin pro uvedené účely poskytuje podporu pro asynchronní programování pomocí Coroutines (korutin), které umožňují zapisovat asynchronní kód sekvenčním a přehledným způsobem a zároveň umožňují pozastavení běhu bez blokování vlákna (Coroutines, 2025; Kotlin coroutines on Android, 2026).

Klíčovou výhodou korutin je efektivita. Na rozdíl od klasických vláken jsou korutiny spravovány na úrovni knihovny a mohou být ve velkém množství efektivně plánovány nad menším počtem vláken (Coroutines, 2025). Korutina může svou činnost dočasně pozastavit (např. při čekání na data z databáze), aniž by blokovala běh vlákna, a po dokončení operace v ní pokračovat.

V rámci vývoje aplikace pro zkoušení slovíček jsou korutiny využívány především v kombinaci s architekturou MVVM. Umožňují plynulé načítání dat na pozadí, zatímco uživatelské rozhraní zůstává responzivní. Zároveň usnadňují správu asynchronních úloh a přispívají k přehlednější struktuře kódu (Coroutines, 2025; Android's Kotlin-first approach, 2026).

1.3 Architektura Android aplikací

Moderní vývoj Android aplikací klade důraz na návrh struktury aplikace a oddělení jednotlivých částí systému. V minulosti byl návrh aplikací často spojen s implementací logiky přímo do komponent uživatelského rozhraní, což vedlo k obtížně udržovatelnému kódu. V současnosti jsou proto doporučovány moderní architektonické přístupy, které zajišťují lepší přehlednost, testovatelnost a dlouhodobou udržitelnost aplikací (Guide to app architecture, 2026).

Společnost Google v této oblasti prosazuje princip oddělení odpovědností a využití vícevrstvých architektur, které umožňují oddělit uživatelské rozhraní od aplikační logiky a práce s daty (Recommendations for Android architecture, 2026). Na zmíněných principech jsou založeny současné doporučené přístupy k návrhu Android aplikací, které budou popsány v následujících podkapitolách.

1.3.1 Problémy tradiční architektury Android aplikací

V počátečních fázích vývoje Android aplikací byla aplikační logika často implementována přímo do komponent životního cyklu, zejména tříd Activity a Fragment. Dané komponenty jsou však primárně určeny pro správu uživatelského rozhraní, nikoliv pro zpracování aplikační logiky (Guide to app architecture, 2026).

Důsledkem takového přístupu byl vznik nepřehledného a obtížně udržovatelného kódu, kdy jednotlivé komponenty obsahovaly jak logiku uživatelského rozhraní, tak práci s daty a řízení stavu aplikace. Dalším problémem je silná vazba na životní cyklus komponent. Při změnách konfigurace, například otočení zařízení, může dojít ke zničení a opětovnému vytvoření aktivity, což vede ke ztrátě aplikačního stavu, pokud není správně ošetřen (Guide to app architecture, 2026).

Z uvedených důvodů jsou v současnosti doporučovány architektonické postupy založené na oddělení odpovědností (Separation of Concerns) a využití samostatných vrstev aplikace (Guide to app architecture, 2026).

1.3.2 Princip oddělení odpovědností (Separation of Concerns)

Princip oddělení odpovědností (Separation of Concerns) je v prostředí Android aplikací využíván jako doporučený přístup pro strukturování aplikace do samostatných vrstev s jasně definovanou odpovědností (Recommendations for Android architecture, 2026).

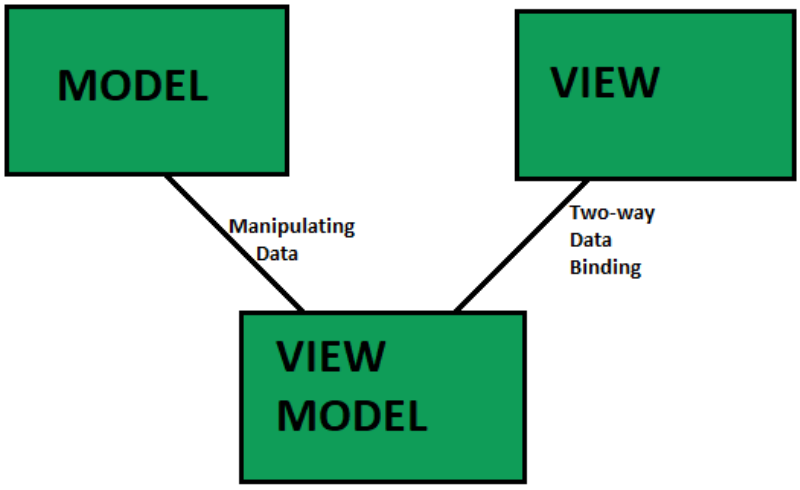
Daný přístup umožňuje oddělit uživatelské rozhraní od aplikační logiky a práce s daty. Uživatelské rozhraní by mělo sloužit pouze k zobrazování dat a zpracování uživatelských interakcí, zatímco aplikační logika a přístup k datům jsou implementovány v samostatných vrstvách (Recommendations for Android architecture, 2026).

Uvedený princip tvoří základ moderních architektonických přístupů, jako je například model MVVM, který je popsán v následující kapitole.

1.3.3 Architektura Model-View-ViewModel (MVVM)

V současnosti je za doporučený přístup k návrhu Android aplikací považována architektura odpovídající vzoru Model–View–ViewModel (MVVM), která odděluje uživatelské rozhraní od aplikační logiky a práce s daty (Guide to app architecture, 2026). Architektura MVVM se skládá ze tří základních vrstev:

1. **Model**: Zahrnuje data a aplikační logiku, včetně přístupu ke zdrojům dat, jako je lokální databáze nebo síť (Guide to app architecture, 2026).
2. **View (Aktivita/Fragment)**: Vrstva zodpovědná za zobrazení uživatelského rozhraní. Neobsahuje žádnou logiku, pouze pozoruje data a reaguje na vstupy uživatele (Guide to app architecture, 2026).
3. **ViewModel**: Slouží jako prostředník mezi vrstvami Model a View. Jeho klíčová vlastnost spočívá v tom, že přežívá změny konfigurace (např. otočení zařízení), čímž zamezuje ztrátě dat (ViewModel overview, 2026).



Obr. 2: Architektura MVVM

Zdroj: https://www.geeksforgeeks.org/websites-apps/introduction-to-model-view-view-model-mvvm/

Pro implementaci MVVM poskytuje platforma Android sadu knihoven Android Jetpack, zejména komponentu ViewModel a nástroje pro práci s životním cyklem (Guide to app architecture, 2026). Dané komponenty umožňují uchovávat stav aplikace nezávisle na uživatelském rozhraní a zajišťují jeho správnou obnovu při změnách konfigurace.

1.3.4 Android Jetpack

Pro implementaci doporučené architektury poskytuje platforma Android sadu knihoven Android Jetpack, která obsahuje nástroje určené pro řešení běžných problémů při vývoji aplikací (Getting started with Jetpack, 2026). Jetpack je navržen tak, aby pomáhal vývojářům vytvářet kvalitní aplikace rychleji a s menším množstvím opakujícího se kódu.

Základní myšlenkou Android Jetpack je poskytnout předpřipravené komponenty, které řeší problémy spojené se správou životního cyklu aplikace, prací s daty nebo kompatibilitou napříč verzemi systému Android (Getting started with Jetpack, 2026).

Součástí Jetpacku jsou mimo jiné knihovny jako ViewModel, Lifecycle nebo Room, které podporují doporučenou architekturu aplikací a princip oddělení zájmů. Dané komponenty umožňují oddělit uživatelské rozhraní od aplikační logiky a práce s daty a přispívají tak k vyšší přehlednosti a udržovatelnosti aplikace (Guide to app architecture, 2026).

Android Jetpack představuje důležitý základ moderního vývoje Android aplikací a je využit i při návrhu aplikace v rámci dané práce.

1.4 Databázová řešení v Androidu

Moderní mobilní aplikace často pracují s daty, která je nutné ukládat trvale přímo v zařízení uživatele. V případě aplikací zaměřených na výuku a procvičování, jako je aplikace pro zkoušení slovní zásoby v rámci navrhované práce, představuje lokální databázové řešení důležitou součást systému. Databáze slouží k uchovávání slovíček, překladů, kategorií a dalších informací, které musí být dostupné i bez připojení k internetu.

1.4.1 Možnosti ukládání dat v Androidu

Platforma Android poskytuje mnoho možností perzistentního ukládání dat, které se liší svou složitostí a vhodností pro konkrétní použití. Mezi základní přístupy patří ukládání dat pomocí úložišť typu klíč–hodnota (např. SharedPreferences nebo DataStore), ukládání do souborového systému a využití relační databáze SQLite (Data and file storage overview, 2026).

Úložiště typu klíč–hodnota jsou vhodná především pro ukládání konfiguračních hodnot nebo uživatelských nastavení, avšak nejsou vhodná pro práci s větším množstvím vzájemně provázaných dat. Ukládání do souborů je využitelné zejména pro nestrukturovaná data, jako jsou texty nebo multimédia, avšak neumožňuje efektivní dotazování ani správu vztahů mezi daty (Data and file storage overview, 2026).

Pro aplikace pracující se strukturovanými daty poskytuje platforma Android relační databázi SQLite, která umožňuje ukládání dat do tabulek a jejich zpracování pomocí SQL dotazů (Data and file storage overview, 2026).

1.4.2 Relační databáze SQLite

Relační databáze SQLite je součástí platformy Android a představuje základní nástroj pro ukládání strukturovaných dat přímo v zařízení uživatele. Jedná se o lehkou databázi, která nevyžaduje samostatný server a umožňuje práci s daty pomocí jazyka SQL (android.database.sqlite, 2025).

SQLite je v prostředí Android zpřístupněno prostřednictvím aplikačního rozhraní, které umožňuje vytváření, správu a práci s databází v aplikacích (android.database.sqlite, 2025).

Přímé použití SQLite však přináší i určité nevýhody. Vývojář musí pracovat s SQL dotazy ve formě textových řetězců a zajišťovat mapování dat mezi databází a objekty aplikační logiky. Zvolený

přístup vede k vyššímu množství opakujícího se kódu a zvyšuje riziko chyb, které se mohou projevit až za běhu aplikace (Save data in a local database using Room, 2026).

Z uvedených důvodů je v moderním vývoji Android aplikací doporučeno využívat vyšší abstrakční vrstvy, jako je knihovna Room, která práci s databází výrazně zjednodušuje.

1.4.3 Room Persistence Library

Pro zjednodušení práce s databází představila společnost Google knihovnu Room Persistence Library, která poskytuje abstrakční vrstvu nad databází SQLite a je součástí Android Jetpack (Save data in a local database using Room, 2026).

Architektura knihovny Room je založena na třech základních komponentách. První z nich je databázová třída, která slouží jako hlavní přístupový bod k databázi a zajišťuje spojení s uloženými daty. Druhou komponentou jsou datové entity, které reprezentují jednotlivé tabulky v databázi aplikace. Třetí komponentou jsou objekty přístupu k datům (DAO), které definují metody pro práci s databází, jako je vkládání, aktualizace, mazání a dotazování dat (Save data in a local database using Room, 2026).

Na základě uvedených vlastností lze Room považovat za vhodné řešení pro práci se strukturovanými daty v moderních Android aplikacích, a proto je využita i v rámci stávající práce.

1.5 Shrnutí dostupné literatury

Na základě provedené literární rešerše lze říci, že dostupné zdroje poskytují dostatečný teoretický základ pro návrh a realizaci mobilní aplikace. Klíčovým zdrojem informací byla oficiální dokumentace platformy Android a nástrojů společnosti Google, která nabízí aktuální informace o doporučených postupech vývoje. Dalším významným zdrojem byla dokumentace programovacího jazyka Kotlin, která poskytuje přehled moderních jazykových prvků a principů využívaných při vývoji aplikací. Kromě oficiálních materiálů jsou dostupné i online výukové kurzy a odborné články, které pomáhají lépe porozumět praktickému použití jednotlivých technologií.

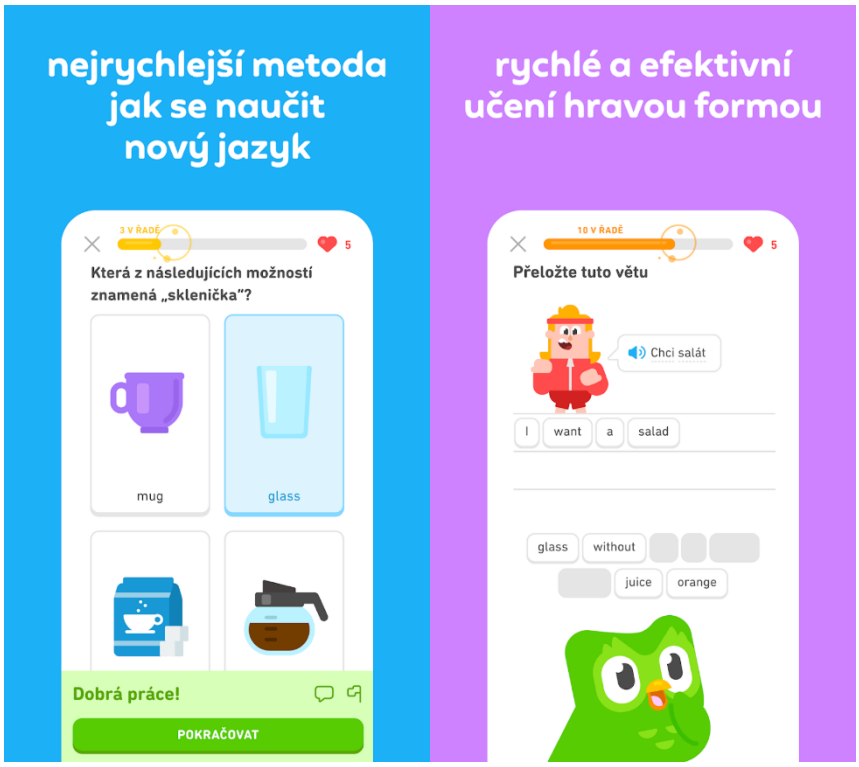
2 Analýza a návrh řešení

Druhá kapitola se věnuje procesu transformace teoretických poznatků do konkrétního návrhu aplikace. Obsahuje analýzu současných výukových aplikací, specifikaci požadavků, návrh uživatelského rozhraní, architektury aplikace a databázového schématu.

2.1 Analýza trhu stávajících výukových aplikací

Při analýze současného trhu s mobilními výukovými aplikacemi lze identifikovat několik významných zástupců, kteří formují způsob, jakým uživatelé studují a procvičují nové znalosti. Moderní aplikace se liší především metodami učení, mírou interaktivity a zaměřením na konkrétní potřeby uživatelů. Mezi velké zástupce patří například aplikace Duolingo, Quizlet, Memrise či Babbel.

Jedním z nejvýraznějších trendů v oblasti digitálního vzdělávání je gamifikace, kterou reprezentuje například aplikace Duolingo. Gamifikace spočívá v implementaci herních prvků, jako jsou body, úrovně, odznaky nebo denní série (streaks), které motivují uživatele k pravidelnému učení. V současnosti je považována za efektivní nástroj pro zvýšení zapojení uživatelů a jejich dlouhodobé aktivity v aplikaci (Gamification examples: Duolingo). Aplikace Duolingo využívá zmíněné principy k vytvoření krátkých interaktivních cvičení, které kombinují procvičování slovní zásoby, poslech i základní gramatiku (Duolingo; Duolingo Review, 2025).



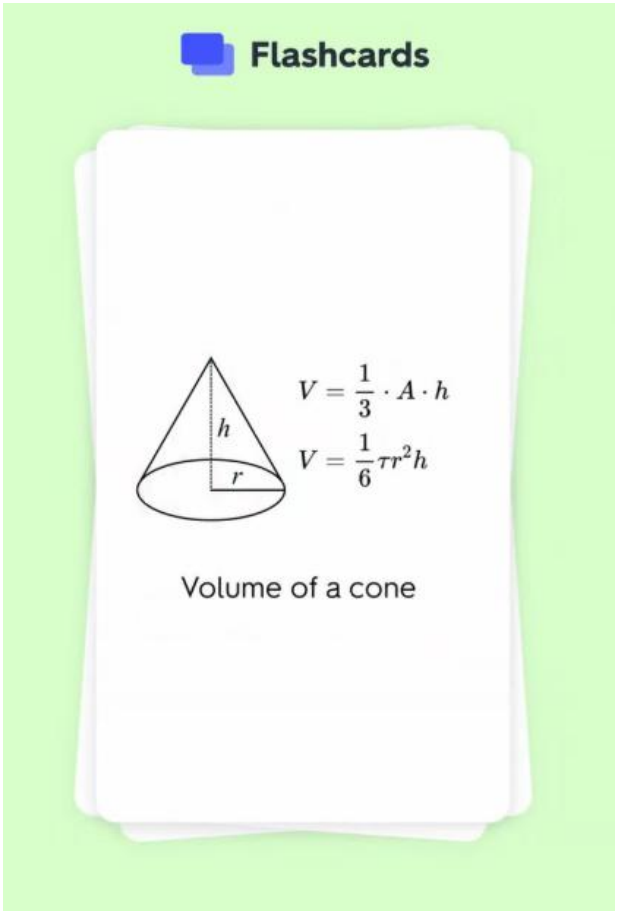
Obr. 3: Ukázka gamifikace Duolingo

Zdroj: https://play.google.com/store/apps/details?hl=cs&id=com.duolingo

Aplikace Memrise kombinuje princip vzdělávání pomocí opakování s multimediálním obsahem, zejména videi rodilých mluvčích. Výuka je zaměřena na praktické použití jazyka a porozumění reálné komunikaci. Uživatel se naučí nejen význam slov, ale i jejich správnou výslovnost a kontext použití. Memrise využívá moderní přístup, který propojuje učení s autentickými situacemi, což může zvyšovat efektivitu osvojování jazyka (Memrise).

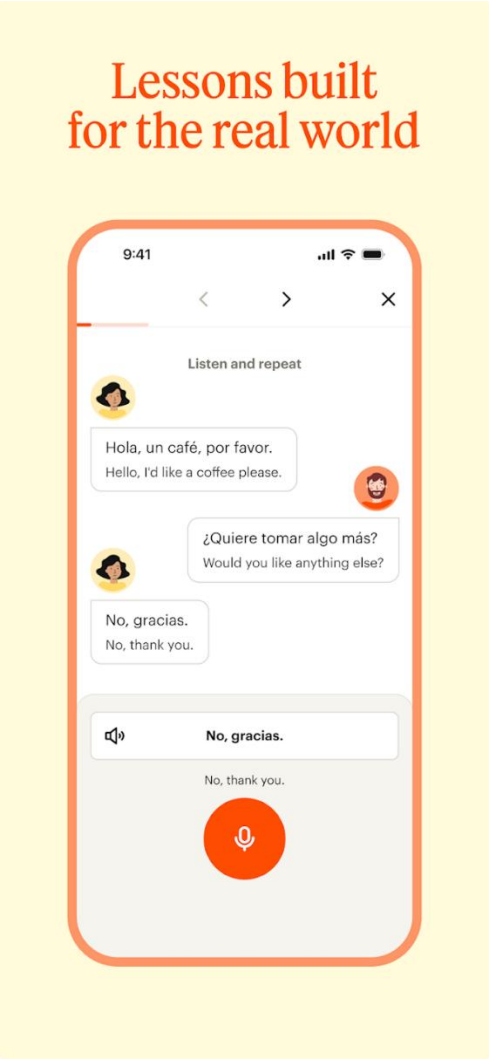
Obr. 4: Ukázka aplikace Memrise
Zdroj: https://www.memrise.com/

Dalším významným způsobem je využití kartičkového systému (flashcards), který je typický pro aplikaci Quizlet. Výukový nástroj umožňuje uživatelům vytvářet vlastní studijní sady nebo využívat existující obsah. Učení probíhá prostřednictvím opakování, testování a různých interaktivních režimů. Daný přístup je vhodný pro zapamatování pojmů a slovní zásoby, ale neposkytuje komplexní vedení výuky ani systematické vysvětlení gramatiky (Quizlet; Quizlet Review, 2024).



Obr. 5: Ukázka kartičkového systému flashcards
Zdroj: https://quizlet.com/features/flashcards

Odlišný styl vzdělávání nabízí aplikace Babbel, která používá strukturované kurzy. Výuka je založena na systematické návaznosti lekcí s důrazem na gramatiku, poslech a konverzaci. Na rozdíl od gamifikace se Babbel zaměřuje na hlubší porozumění jazykovým pravidlům. Uvedený styl se blíží tradiční výuce a je ideální pro uživatele, kteří preferují řízené studium (Babbel; Babbel Review, 2025).



Obr. 6: Ukázka výuky konverzace

Zdroj: https://play.google.com/store/apps/details?id=com.babbel.mobile.android.en&hl=cs

V moderních vzdělávacích aplikacích lze také pozorovat trend mikrovzdělávání (micro-learning), který spočívá v krátkých, časově nenáročných lekcích. Uvedený způsob umožňuje učit se v krátkých časových intervalech během dne a zvyšuje pravděpodobnost pravidelného používání aplikace. Krátké lekce jsou často kombinovány s gamifikací a opakováním, což vytváří efektivní a uživatelsky přívětivý model učení (What is Microlearning?, 2026; What is Microlearning and Why Is It Effective?, 2025).

Na základě provedené analýzy lze říci, že současné aplikace kombinují především gamifikaci, opakování a krátké interaktivní cvičení, které podporují pravidelné učení. Zároveň však existuje prostor pro jednodušší aplikaci zaměřenou na rychlé procvičování slovní zásoby bez nutnosti registrace, složitého nastavování nebo připojení k internetu. Navrhovaná aplikace se proto soustředí na minimalistický přístup, který kombinuje výhody mikrovzdělávání s důrazem na jednoduchost a okamžitou použitelnost.

2.2 Specifikace požadavků

Specifikace požadavků slouží jako přesné zadání pro vývoj aplikace a zároveň jako kritérium pro její následné testování. Požadavky jsou rozděleny na funkční, které popisují konkrétní chování systému a interakce s uživatelem, a nefunkční, které definují technická omezení a vlastnosti aplikace.

2.2.1 Funkční požadavky

Funkční požadavky definují operace, které musí být aplikace schopna provádět v reakci na vstupy uživatele. Pro navrhovanou aplikaci byly stanoveny následující funkce:

- **Organizace obsahu**: Systém umožní hierarchické členění slovní zásoby. Uživatel si nejprve volí tematickou kategorii, která se dále dělí na jednotlivé úrovně obsahující slovíčka.
- **Režim zkoušení**: V rámci lekce bude uživateli zobrazeno české slovíčko a tři varianty anglického překladu. Pouze jedna z uvedených variant je správná.
- **Odložená zpětná vazba**: Na rozdíl od okamžité indikace bude zpětná vazba uživateli poskytnuta až po dokončení celé úrovně. Cílem je udržet plynulost testování a simulovat reálné podmínky zkoušení.
- **Vyhodnocení lekce**: Po ukončení zkoušení aplikace zobrazí přehled všech zodpovězených otázek, přičemž u nesprávné odpovědi bude uveden i správný český překlad.
- **Sledování úspěšnosti**: Systém bude průběžně sledovat výsledky uživatele a zobrazovat statistiky úspěšnosti, což slouží jako motivační prvek.

2.2.2 Nefunkční požadavky

Nefunkční požadavky určují, jak má systém fungovat z hlediska výkonu, použitelnosti a technických omezení. Byly stanoveny následující funkce:

- **Platforma**: Aplikace bude určena pro operační systém Android.
- **Dostupnost dat**: Veškerá data budou uložena lokálně v zařízení (v databázi Room), aplikace nebude vyžadovat připojení k internetu.
- **Responzivní design (Responzivita)**: Uživatelské rozhraní musí reagovat plynule.
- **Intuitivnost**: Rozhraní bude navrženo tak, aby nevyžadovalo nápovědu a uživatel mohl začít se zkoušením ihned po instalaci.

2.3 Návrh uživatelského rozhraní

Uživatelské rozhraní zahrnuje vizuální prvky, jako jsou tlačítka, texty, ikony a rozvržení obrazovky, které umožňují ovládání aplikace a zobrazování informací. Navržené uživatelské rozhraní by mělo být přehledné, konzistentní a snadno ovladatelné, aby uživatel mohl aplikaci používat bez zbytečných komplikací.

Cílem návrhu v rámci dané práce je vytvořit jednoduchý a intuitivní design, který umožní uživateli rychlou orientaci v aplikaci a plynulé používání bez nutnosti nápovědy. Důraz je kladen především na přehledný design jednotlivých obrazovek a minimalizaci rušivých prvků.

2.3.1 První spuštění aplikace

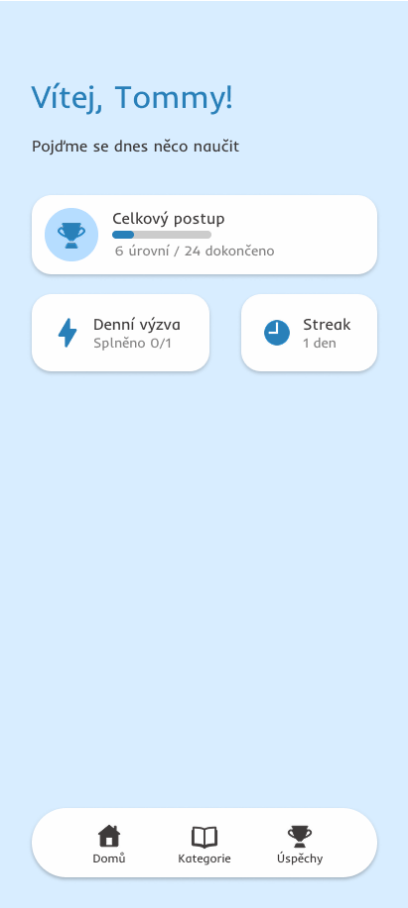
Design obrazovek je navržen s důrazem na jednoduchost a přehlednost. Použité modré odstíny mají v uživateli vyvolat uklidňující dojem a podpořit soustředěnost při používání aplikace. První obrazovka zobrazuje logo aplikace, krátký uvítací text a tlačítko, které uživatele přesměruje na další obrazovku pro výběr přezdívky, která má sloužit k personalizaci aplikace. Uvedené obrazovky se zobrazí pouze při prvním spuštění aplikace po instalaci.

Obr. 7: UI návrh pro uvítání a výběr přezdívky
Zdroj: vlastní práce

2.3.2 Další spuštění aplikace

Design aplikace navazuje na předchozí stránky, zachovává jednotné barevné schéma a konzistenci prostředí. Po zadání přezdívky nebo při dalších spuštěních aplikace je uživatel přesměrován na domovskou obrazovku, jejímž cílem je poskytnout přehled o postupu. Zobrazuje se zde uvítací text s přezdívkou uživatele, ukazatel celkového postupu, informace o splnění denní výzvy a počet streaku, který motivuje uživatele k pravidelnému používání aplikace.

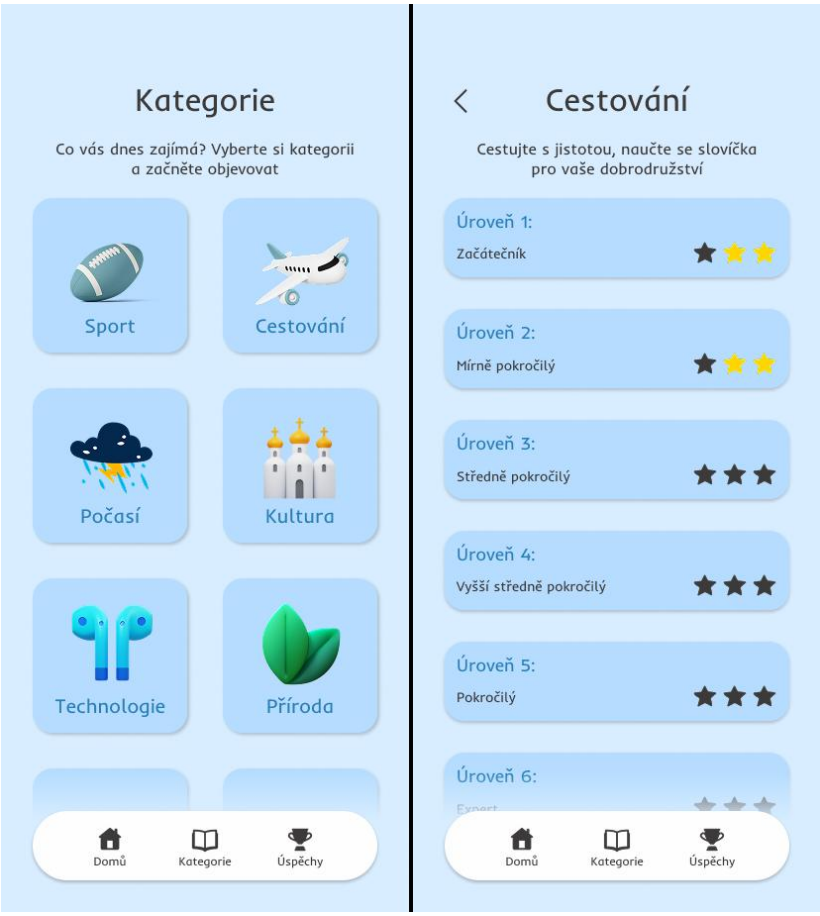
Ve spodní části obrazovky se nachází navigační lišta, která slouží k přechodu mezi jednotlivými obrazovkami a zajišťuje jednoduchou a intuitivní orientaci uživatele napříč celou aplikací.



Obr. 8: UI návrh pro domovskou obrazovku

Zdroj: vlastní práce

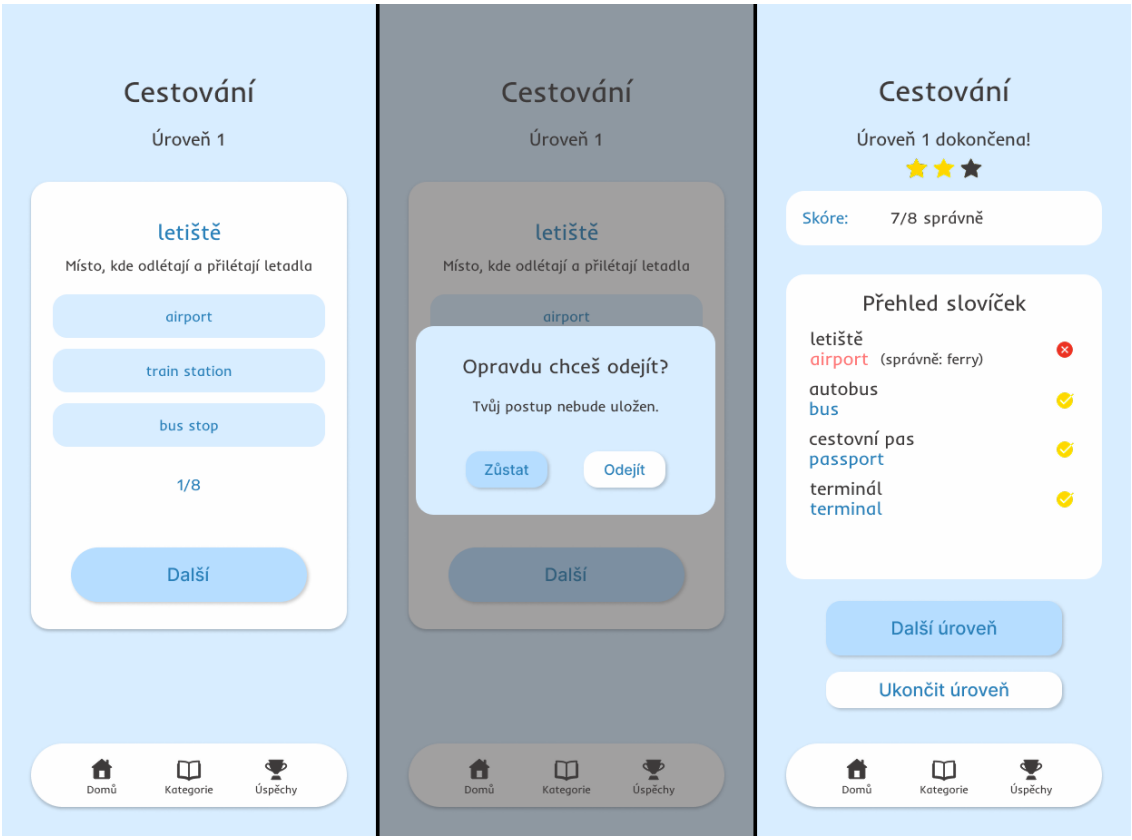
Design obrazovky kategorií a úrovní navazuje opět na jednotný vizuální styl aplikace, zachovává přehledné rozložení a barevné schéma. Obrazovka kategorií umožňuje uživateli výběr témat, ve kterých může procvičovat slovní zásobu. Jednotlivé kategorie jsou zobrazeny formou dlaždic, které obsahují ikonku a název. Navržené řešení podporuje rychlou orientaci a intuitivní výběr. Po výběru kategorie je uživatel přesměrován na obrazovku úrovní, které jsou rozděleny podle obtížnosti. Každá úroveň obsahuje označení a vizuální prvek hvězdiček, které znázorňují úspěšnost uživatele v dané úrovni.



Obr. 9: UI návrh pro kategorie a úrovně

Zdroj: vlastní práce

Obrazovka zkoušení slovíček zobrazuje název kategorie a číslo úrovně. Uživateli je zobrazen český výraz a tři anglické varianty překladu pro výběr správné odpovědi. Součástí je také indikace postupu, která informuje uživatele o postupu v úrovni. Proti nechtěnému opuštění úrovně během zkoušení je zobrazen upozorňující dialog o odchodu. Po dokončení úrovně je zobrazena obrazovka s výsledky zkoušení, která poskytuje souhrn dosaženého skóre, počet hvězdiček, které za úroveň získal a přehled slovíček, kde se uživateli zobrazí všechny špatné i správné odpovědi. Součástí jsou také ovládací prvky (tlačítka) umožňující pokračovat na další úroveň nebo ji ukončit. Obrazovky jsou navrženy tak, aby bylo možné se soustředit na obsah zkoušení bez rušivých prvků.



Obr. 10: UI návrh pro zkoušení a hodnocení

Zdroj: vlastní práce

Obrazovka úspěchů slouží k přehledu dosažených výsledků uživatele v jednotlivých kategoriích. Konkrétně, když uživatel dokončí celou kategorii slovíček, úspěch se odemkne a uživatel uvidí svůj výsledek (dosažený úspěch). Návrh zachovává jednotný vizuální styl aplikace a využívá přehledné rozložení prvků pro snadnou orientaci. Jednotlivé úspěchy jsou zobrazeny formou dlaždicového rozložení pro lepší přehled. Každá dlaždice reprezentuje konkrétní kategorii. Odemčené úspěchy jsou zvýrazněny bílým pozadím a ikonou zlatého poháru, zatímco uzamčené jsou označené šedým pozadím a ikonou zámku. Daný vizuální rozdíl pomůže uživateli k rychlé identifikaci svého pokroku a podporuje motivaci k dokončení dalších kategorií.

Vzhled obrazovky s úspěchy je vidět na obrázku 11. Obrazovka má přívětivý vizuální vzhled a aplikace je díky oknům s kategoriemi snadno ovladatelná.

Obr. 11: UI návrh pro úspěchy
Zdroj: vlastní práce

2.4 Návrh architektury aplikace

Návrh architektury aplikace vychází z doporučení pro vývoj Android aplikací, která byla popsána v teoretické části práce. Představuje důležitou fázi vývoje, jejíž cílem je vytvořit přehlednou, udržitelnou a snadno rozšiřitelnou strukturu aplikace, která odděluje jednotlivé části systému podle jejich účelu. Díky správnému návrhu lze předejít nejasnostem či komplikacím v průběhu implementace.

Pro navrhovanou aplikaci byla zvolena architektura Model–View–ViewModel (MVVM), která umožňuje oddělení uživatelského rozhraní, aplikační logiky a datové vrstvy. Zvolený přístup přispívá k lepší organizaci kódu a umožňuje navrhovat jednotlivé části aplikace samostatně.

Uživatelské rozhraní je realizováno pomocí frameworku Jetpack Compose a slouží k prezentaci dat a zpracování uživatelských vstupů. Aplikační logika je soustředěna do vrstvy ViewModel, která zajišťuje řízení průběhu zkoušení, vyhodnocení odpovědí a správu postupu uživatele. Datová vrstva je implementována pomocí knihovny Room, která umožňuje efektivní ukládání a správu dat aplikace, včetně slovní zásoby, kategorií a uživatelského postupu.

Navržená architektura poskytuje pevný základ pro implementaci aplikace a zároveň umožňuje její další rozšíření, například o nové kategorie a slovíčka v databázové vrstvě nebo o nové typy cvičení, bez nutnosti zásadních změn.

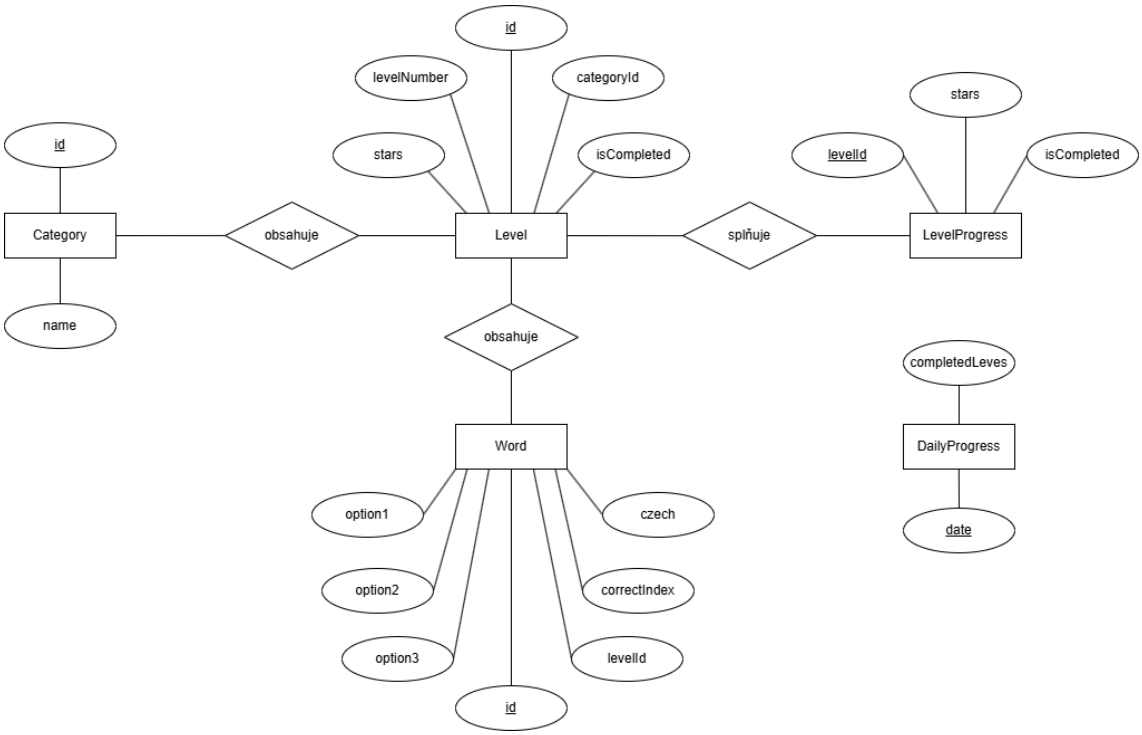
2.5 Návrh databázového schématu

Návrh databázového schématu vychází z požadavků aplikace na ukládání strukturovaných dat a zajištění jejich efektivní správy. Databáze slouží k uchovávání slovní zásoby, tematických kategorií, jednotlivých úrovní a postupu uživatele při zkoušení.

Datová vrstva je navržena s využitím knihovny Room Persistence Library. Databázová struktura je tvořena základními entitami, které reprezentují jednotlivé části aplikace. Základ tvoří entita Category, která obsahuje seznam tematických kategorií, které obsahují více úrovní reprezentovanou entitou Level. Úrovně jsou tvořeny konkrétními slovíčky uloženými v entitě Word, která obsahuje český výraz a tři anglické varianty překladu a index správné odpovědi.

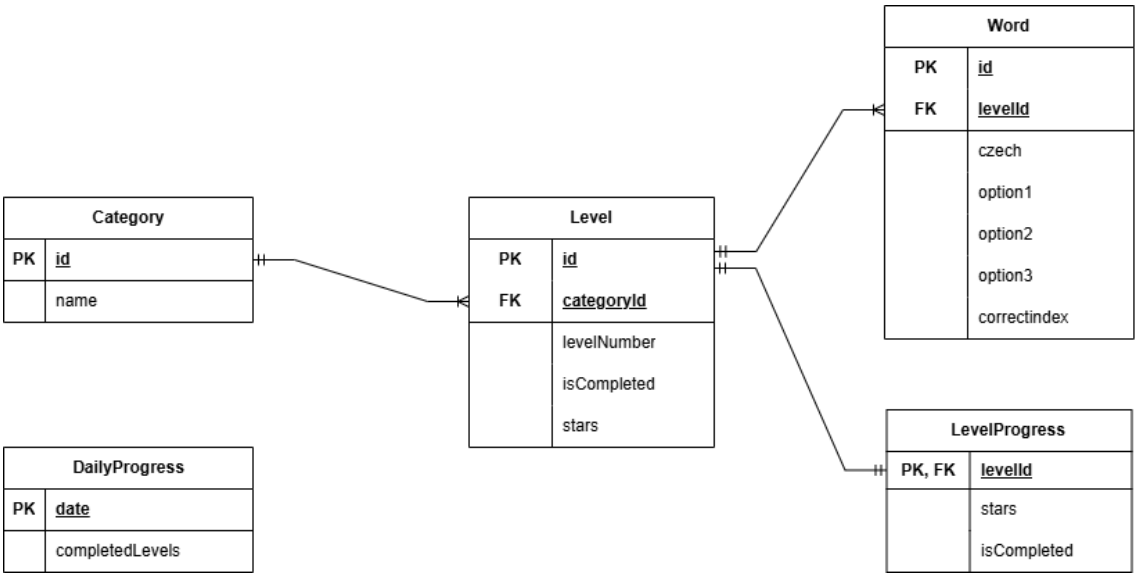
Pro ukládání postupu uživatele jsou využity dvě samostatné entity. Entita LevelProgress slouží k uložení informace o dokončení a počtu získaných hvězdiček. Entita DailyProgress zaznamenává denní aktivitu uživatele a sleduje pravidelnost používání aplikace pomocí streaku.

Databázové schéma je znázorněno pomocí ER diagramu a následně relačního modelu, které přehledně zachycují jednotlivé entity a jejich vzájemné vztahy.



Obr. 12: ER diagram databáze aplikace na zkoušení slovíček

Zdroj: vlastní práce



Obr. 13: Relační model databáze aplikace na zkoušení slovíček

Zdroj: vlastní práce

2.6 Funkce aplikace

Navrhovaná aplikace je zaměřena na procvičování slovní zásoby z českého jazyka do anglického a kombinuje principy mikrovzdělávání s prvky gamifikace. Jejím cílem je umožnit uživateli rychlé, jednoduché a pravidelné procvičování bez nutnosti složitého nastavování nebo připojení k internetu.

Po prvním spuštění aplikace je uživatel vyzván k zadání přezdívky, která slouží k personalizaci prostředí a je následně zobrazována na domovské obrazovce. Hlavní obrazovka poskytuje přehled o aktuálním postupu uživatele prostřednictvím tří základních prvků. Zobrazen je celkový postup vyjádřený počtem dokončených úrovní z celkového počtu, doplněný vizuálním prvkem ve formě progress baru. Dále je zde denní výzva, která motivuje uživatele ke splnění určitého počtu úrovní za den, a ukazatel streaku, který zaznamenává počet po sobě jdoucích dní, kdy uživatel splnil alespoň jednu úroveň. V případě vynechání dne dojde k následnému resetování.

Hlavní funkcí aplikace je zkoušení slovní zásoby. Obsah je rozdělen do tematických kategorií, které jsou dále rozděleny na jednotlivé úrovně s postupně se zvyšující obtížností. Každá úroveň obsahuje sadu slovíček, u kterých je uživateli zobrazen český výraz a tři anglické překlady, kdy pouze jedna odpověď je správná. Volbu může svou volbu odpovědi změnit až do okamžiku přechodu na další otázku.

V průběhu zkoušení je uživatel informován o svém postupu v rámci úrovně. Pro zamezení nechtěného ukončení je navržen potvrzovací dialog, který se zobrazí při pokusu o opuštění úrovně. Uživatel je tak upozorněn, že jeho postup nebude uložen, a může se rozhodnout, zda chce pokračovat, nebo úroveň opustit. Po dokončení úrovně je zobrazena obrazovka s výsledky, která obsahuje dosažené skóre, počet získaných hvězdiček a přehled všech slovíček včetně správných a nesprávných odpovědí. Uživatel má následně možnost pokračovat na další úroveň nebo se vrátit zpět do hlavní nabídky.

Další součástí aplikace je systém úspěchů, který slouží jako motivační prvek. Úspěchy jsou navázány na jednotlivé kategorie a jejich odemčení je podmíněno dokončením všech úrovní v dané kategorii. Na obrazovce úspěchů jsou jednotlivé položky zobrazeny formou dlaždic, které jsou vizuálně odlišené podle stavu. Rozdílné grafické zpracování umožňuje rychlý přehled mezi odemčenými a uzamčenými úspěchy. Po dokončení celé kategorie dojde ke změně vzhledu dané dlaždice, což uživateli poskytuje okamžitou zpětnou vazbu.

Navržené funkce aplikace jsou vzájemně propojené a podporují pravidelné používání. Kombinace jednoduchého ovládání, přehledného uživatelského rozhraní a motivačních prvků vytváří prostředí vhodné pro efektivní procvičování slovní zásoby.

3 Implementace aplikace

Následující kapitola představuje průběh vývoje navržené aplikace, který proběhl ve vývojovém prostředí Android Studio za využití již představeného programovacího jazyka Kotlin a moderního frameworku Jetpack Compose.

3.1 Použité technologie a vývojové prostředí

Během vývoje byly použity moderní nástroje a technologie určené pro platformu Android, které umožňují vytvářet přehledné, funkční a snadno rozšiřitelné mobilní aplikace.

Kotlin
Hlavním programovacím jazykem byl Kotlin. Oproti dříve používané Javě nabízí jednodušší zápis kódu, lepší práci s chybami a obecně přehlednější strukturu programu.

Jetpack Compose
Nástroj Jetpack Compose byl využit pro implementaci uživatelského rozhraní, čímž bylo dosaženo propojení UI s architekturou MVVM.

MVVM
Struktura aplikace vychází z architektury MVVM (Model–View–ViewModel), která rozděluje aplikaci na jednotlivé části podle jejich funkce. Datová vrstva se stará o ukládání a načítání dat, ViewModel obsahuje logiku aplikace a uživatelské rozhraní pouze zobrazuje data uživateli.

Room
Pro ukládání dat byla použita knihovna Room, která zjednodušuje práci s databází SQLite. Umožňuje definovat strukturu databáze pomocí tříd, které reprezentují jednotlivé tabulky a anotací, které určují jejich vlastnosti a vztahy.

Coroutines a Flow
Pro práci s časově náročnými operacemi, jako je například načítání dat z databáze, byly využity Kotlin coroutines. Ty umožňují provádět dané operace na pozadí bez blokování hlavního vlákna aplikace. Byl také využit nástroj StateFlow pro práci s datovými toky. Díky němu dochází k automatické aktualizaci uživatelského rozhraní při změně dat, což zjednodušuje synchronizaci mezi logikou aplikace a UI.

Android studio
Vývoj aplikace probíhal ve vývojovém prostředí Android Studio, které poskytuje nástroje pro psaní kódu, hledání chyb i testování v emulátoru. Pro sestavení projektu a správu knihoven byl využit nástroj Gradle.

3.2 Datová vrstva (Room)

Datová vrstva slouží jako základ celé aplikace, protože uchovává strukturu kategorií, seznam úrovní, jednotlivá slovíčka, postup uživatele, denní aktivitu a počet streaku. Struktura databáze odpovídá logice aplikace, kdy jsou jednotlivé části rozděleny do entit, které reprezentují tabulky v databázi a DAO rozhraní, která slouží pro práci daty.

3.2.1 Datové modely (Entity)

Každá entita je definována jako datová třída a odpovídá konkrétnímu typu dat v aplikaci:

- **CategoryEntity**: Obsahuje seznam kategorií slovíček. Každá kategorie má své categoryId a název.
- **LevelEntity**: Reprezentuje jednotlivé úrovně v kategoriích. Obsahuje vazbu na kategorii (categoryId) a číslo úrovně.
- **WordEntity**: Obsahuje slovíčka ke zkoušení uživatele. Ukládá český výraz, tři anglické možnosti překladu a index správné odpovědi.
- **LevelProgressEntity**: Uchovává postup uživatele v jednotlivých úrovních, tedy počet hvězdiček a stav dokončení.
- **DailyProgressEntity**: Slouží pro sledování denní aktivity uživatele a výpočet streaku.

Výše uvedené entity tvoří základní strukturu databáze, které umožňují ukládání dat.

3.2.2 Přístup k databázi (DAO)

Pro práci s daty jsou použity DAO (Data Access Object) rozhraní. Každá entita má své vlastní DAO, které obsahuje metody pro načítání a ukládání dat. Odděluje databázové operace od zbytku aplikace, což pomáhá k lepší přehlednosti a jednodušší údržbě kódu.

- CategoryDao – Načítá kategorie z databáze.
- LevelDao – Načítá seznam úrovní pro konkrétní kategorii.
- WordDao – Zajišťuje načtení slovíček pro konkrétní úroveň.
- LevelProgressDao – Ukládá postup uživatele v jednotlivých úrovních.
- DailyProgressDao – Ukládá a načítá záznam denní aktivity uživatele, je také využita pro výpočet streaku.

3.2.3 Hlavní databázová třída (Databazos)

Hlavní částí datové vrstvy je třída Databazos, která propojuje jednotlivé entity s jejich DAO rozhraními a definuje strukturu celé databáze.

Je zde uveden seznam všech entit, které databáze obsahuje, a zároveň verze databáze, která se musí změnit při jakékoliv změně. Třída dále obsahuje abstraktní metody pro získání jednotlivých DAO objektů, které jsou následně používány ve ViewModelu.

```kotlin
@Database(
    entities = [
        CategoryEntity::class,
        LevelEntity::class,
        WordEntity::class,
        LevelProgressEntity::class,
        DailyProgressEntity::class
    ],
    version = 4
)
abstract class Databazos : RoomDatabase() {
    7 Usages
    abstract fun categoryDao(): CategoryDao
    2 Usages
    abstract fun levelDao(): LevelDao
    44 Usages
    abstract fun wordDao(): WordDao
    1 Usage
    abstract fun levelProgressDao(): LevelProgressDao
    1 Usage
    abstract fun dailyProgressDao(): DailyProgressDao
}
```

Obr. 14: Databazos.kt
Zdroj: vlastní práce

3.2.4 Inicializace databáze (AppDatabaseProvider)

Třída zajišťuje vytvoření a správu instance databáze. Používá návrhový vzor singleton pattern, který zajišťuje, že existuje pouze jedna instance databáze. Vytvoření databáze provede metoda *Room.databaseBuilder*, která využívá kontext aplikace a název databázového souboru. Součástí je také metoda *fallbackToDestructiveMigration*, která při změně struktury dojde k jejímu smazání a vytvoření nové.

```kotlin
val instance =
    Room.databaseBuilder(
    context.applicationContext,
     klass = Databazos::class.java,
     name = "learnify_db"
)
    //.fallbackToDestructiveMigration()
    .build()
```

Obr. 15: Inicializace v AppDatabaseProvider
Zdroj: vlastní práce

3.2.5 Inicializační data (DatabazosData)

Třída DatabazosData slouží k počátečnímu naplnění databáze při prvním spuštění aplikace. Databáze je navržena lokálním způsobem (offline) což znamená, že všechna data musí být dostupná přímo v zařízení uživatele bez nutnosti připojení k internetu.

Do databáze jsou v rámci inicializace nejprve vloženy jednotlivé kategorie, kde je každé z nich přiřazeno unikátní identifikační číslo. To je následně využíváno při vytvoření úrovní a přiřazování slovíček ke kategoriím. Následně je pro každou kategorii vytvořeno šest úrovní pomocí pomocné funkce, která zajišťuje jednotnou strukturu celé aplikace.

```
suspend fun createLevels(categoryId: Int): List<Int> {
    return (1 ≤ .. ≤ 6).map { levelNumber ->
        db.levelDao().insert(
            LevelEntity(
                categoryId = categoryId,
                levelNumber = levelNumber
            )
        ).toInt()
    }
}
```

Obr. 16: Pomocná funkce pro vytvoření úrovní

Zdroj: vlastní práce

Na začátku vývoje byl použit způsob zápisu slovíček do databáze pomocí jednotlivých objektů třídy WordEntity. Později se daný způsob ukázal jako nevhodný, kvůli velkému množství opakujícího se kódu. Každé slovíčko muselo být definováno se všemi atributy, což vedlo ke špatné čitelnosti a vyšší pravděpodobnosti vzniku chyb. Při velkém množství dat by také byla údržba kódu velmi obtížná.

```
db.wordDao().insertAll( words = listOf(
    WordEntity(
        levelId = animalLevels[0],
        czech = "pes",
        option1 = "dog",
        option2 = "cat",
        option3 = "horse",
        correctIndex = 0
    ),
    WordEntity(
        levelId = animalLevels[0],
        czech = "kočka",
        option1 = "cat",
        option2 = "fox",
        option3 = "fish",
        correctIndex = 1
    )
))
```

Obr. 17: Původní zápis dat

Zdroj: vlastní práce

Ve snaze vyhnout se možným komplikacím a horší přehlednosti kódu byl zvolen způsob zápisu pomocí pomocné funkce, která zápis slovíček výrazně zjednodušuje.

```
fun word(levelId: Int, czech: String, correct: String,
         wrong1: String, wrong2: String) =
    WordEntity(
        levelId = levelId,
        czech = czech,
        option1 = correct,
        option2 = wrong1,
        option3 = wrong2,
        correctIndex = 0
    )
```

Obr. 18: Pomocná funkce pro zápis dat

Zdroj: vlastní práce

Slovíčka jsou tak definována jedním českým výrazem, ke kterému jsou přiřazeny tři anglické překlady, kdy pouze jeden je správný. Správná odpověď je uložena vždy na první pozici. V aplikaci však nedochází k zobrazení odpovědí v pevném pořadí, protože ve ViewModelu je tento stav ošetřen pomocí funkce *getShuffledOptions*, která odpovědi náhodně promíchá.

```
db.wordDao().insertAll( words = listOf(
    word( levelId = animalLevels[0], czech = "pes", correct = "dog", wrong1 = "cat", wrong2 = "horse"),
    word( levelId = animalLevels[0], czech = "kočka", correct = "cat", wrong1 = "fox", wrong2 = "fish"),
    word( levelId = animalLevels[0], czech = "kůň", correct = "horse", wrong1 = "eagle", wrong2 = "mouse"),
    word( levelId = animalLevels[0], czech = "kráva", correct = "cow", wrong1 = "goat", wrong2 = "pig"),
    word( levelId = animalLevels[0], czech = "prase", correct = "pig", wrong1 = "sheep", wrong2 = "camel"),
    word( levelId = animalLevels[0], czech = "ovce", correct = "sheep", wrong1 = "bison", wrong2 = "goat"),
    word( levelId = animalLevels[0], czech = "lev", correct = "lion", wrong1 = "tiger", wrong2 = "jaguar"),
    word( levelId = animalLevels[0], czech = "had", correct = "snake", wrong1 = "lizard", wrong2 = "crocodile")
))
```

Obr. 19: Finální zápis dat

Zdroj: vlastní práce

Důležitým rozhodnutím je, že názvy kategorií jsou v aktuální verzi aplikace pevně definované v kódu společně s reprezentujícím obrázkem. Následující řešení bylo zvoleno pro zjednodušení implementace. Kategorie jsou ale zároveň ukládány do databáze, což umožňuje budoucí možnosti rozšíření.

3.3 ViewModel (AppVM)

Důležitá vrstva, která zajišťuje propojení mezi datovou vrstvou a UI. V aplikaci je reprezentována třídou AppVM, která obsahuje veškerou aplikační logiku. Hlavním úkolem ViewModelu je načítání dat z databáze a poskytnout je uživatelskému rozhraní.

3.3.1 Role ViewModelu v aplikaci

Třída AppVM zajišťuje několik důležitých funkcí:

- Načítání slovíček pro jednotlivé úrovně.
- Řízení průběhu zkoušení (aktuální otázka, vyhodnocení odpovědi, zamíchání odpovědí).
- Ukládání výsledků uživatele do databáze.

- Výpočet hvězdiček podle úspěšnosti.
- Sledování denní aktivity a výpočet streaku.
- Poskytování dat pro zobrazení celkového postupu uživatele.
- Načítání seznamu úrovní.
- Uchovávání jednotlivých odpovědí během zkoušení.
- Automatická aktualizace UI při změně dat pomocí nástroje StateFlow.

ViewModel také využívá přístup k databázi prostřednictvím DAO objektů, které jsou získány z třídy AppDatabaseProvider při vytvoření instance, čímž dochází k oddělení aplikační logiky od datové vrstvy.

3.3.2 StateFlow a práce se stavem aplikace

Pro předání dat mezi ViewModelem a UI je použit nástroj *StateFlow*. Jednotlivá data jsou ve ViewModelu ukládána do objektů typu *MutableStateFlow*, například seznam slovíček pro aktuální úroveň, index aktuální otázky nebo denní progres. Dané hodnoty jsou následně zpřístupněny jako *StateFlow*. Díky tomu dochází k automatické aktualizaci UI při každé změně dat.

3.3.3 Načítání dat

Načítání dat probíhá asynchronně pomocí coroutine v rámci *viewModelScope*. Daný mechanismus zajišťuje, že operace nad databází neblokují hlavní vlákno aplikace.

ViewModel načítá:

- Slovíčka pro konkrétní úroveň.
- Úroveň pro vybranou kategorii.
- Všechny úrovně pro výpočet celkového postupu.

Načtená data jsou ukládána do objektu *StateFlow*, což umožňuje jejich automatické zobrazení v uživatelském rozhraní.

3.3.4 Ukládání dat

Po dokončení úrovně dochází k uložení výsledku uživatele do databáze. Nejprve je vypočítán počet získaných hvězdiček a následně je vytvořen záznam o průběhu úrovně. Současně je aktualizována denní aktivita. Po dokončení úrovně se zvýší počet splněných úrovní pro aktuální den. Příslušná hodnota je následně využita pro denní výzvu, která je v aktuální verzi aplikace nastavena na dvě úrovně. Uložená data jsou také využita pro výpočet streaku.

3.3.5 Ukládání dat

- **Výpočet hvězdiček**: Po dokončení úrovně je počet hvězdiček určen na základě chybných odpovědí. Menší počet chyb znamená lepší hodnocení. Daný způsob hodnocení slouží k motivaci uživatele se více soustředit na výběr správné odpovědi.

```
fun calculateStars(): Int {
    val total = _results.value.size
    val correct = _results.value.count { it.isCorrect }
    val wrong = total - correct

    return when {
        wrong <= 1 -> 3
        wrong <= 4 -> 2
        else -> 1
    }
}
```

Obr. 20: Funkce pro výpočet hvězdiček

Zdroj: vlastní práce

- **Výpočet streaku**: Představuje počet dní po sobě, kdy uživatel splnil alespoň jednu úroveň. Výpočet funguje na základě dat z databáze, kdy vezme dnešní datum a porovnává ho s uloženými záznamy.
- **Denní progres**: Slouží k evidenci počtu úrovní, které uživatel dokončil během dne. Hodnota se ukládá bez omezení a při každém dokončení úrovně se zvyšuje. Následně jsou získaná data použita pro zobrazení denní výzvy, která je omezena na splnění dvou úrovní za den. Omezení denní výzvy slouží jen jako motivační prvek a nijak neomezuje uživatele v dalším studiu.
- **Zamíchání odpovědí**: Před zobrazením otázky dochází k promíchání nabízených odpovědí, aby se předešlo, že správná odpověď je vždy na prvním místě.

3.3.6 Práce s výsledky zkoušení

Během zkoušení jsou jednotlivé odpovědi ukládány do seznamu výsledků. Každá odpověď obsahuje české slovíčko, správný překlad, zvolenou odpověď a jestli je správná. Výsledky se následně zobrazují na obrazovce s vyhodnocením, kde uživatel může vidět svůj přehled odpovědí.

3.4 Uživatelské rozhraní

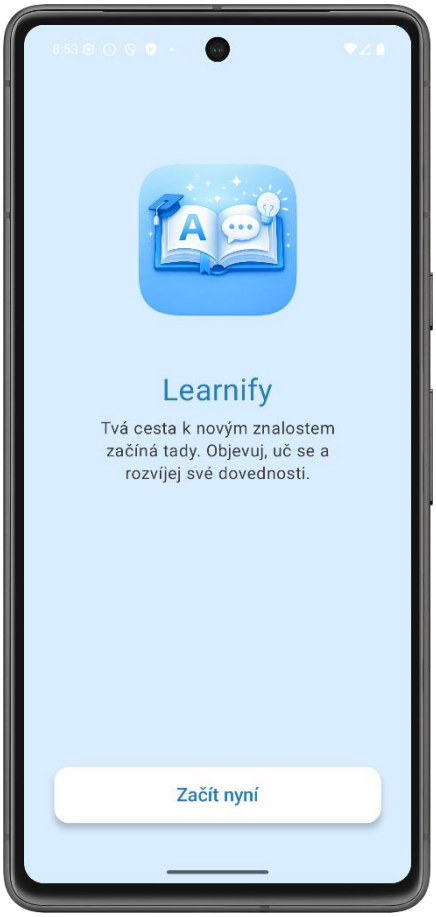
Uživatelské rozhraní aplikace vychází z návrhu vytvořeného v nástroji pro grafický návrh Figma a bylo implementováno pomocí frameworku Jetpack Compose. Při vývoji byl kladen důraz na zachování jednoduchosti, přehledném uspořádání a intuitivním ovládání.

V následujících podkapitolách jsou jednotlivé obrazovky aplikace představeny z pohledu uživatele, včetně obsahu a způsobu interakce.

3.4.1 Onboarding

Onboarding představuje úvodní část aplikace, která se zobrazí při prvním spuštění aplikace po instalaci. Cílem je seznámit uživatele s aplikací a také získat základní informaci ve formě přezdívky, potřebnou pro další používání.

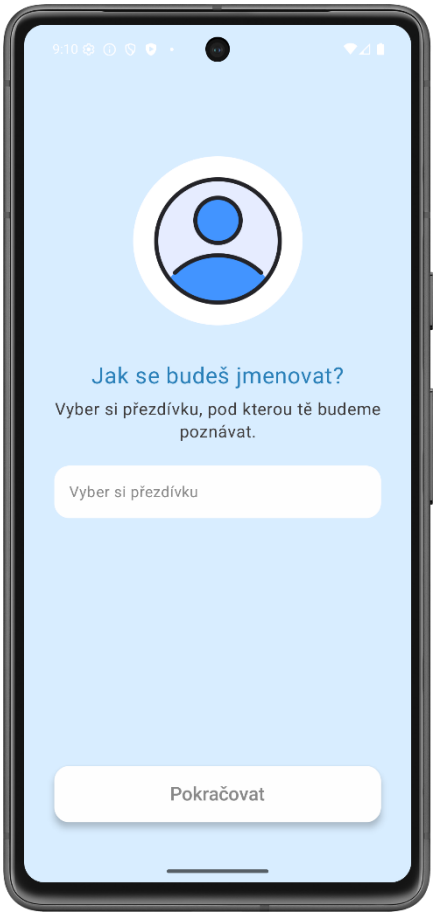
První obrazovka má úvodní charakter. Obsahuje logo, název aplikace „Learnify“ a krátký popis, který uživateli předá informaci o účelu aplikace. Hlavním prvkem je tlačítko „Začít nyní“, které umožňuje přejít na další krok onboarding procesu.



Obr. 21: Náhled OnboardingFirst z emulátoru

Zdroj: vlastní práce

Druhá obrazovka slouží k zadání přezdívky, pod jakou bude uživatele vítat při dalších spuštění aplikace. Tlačítko „Pokračovat“ je aktivní ve chvíli, kdy je pole vyplněno, čímž je zajištěno, že je zadána platná hodnota.



Obr. 22: Náhled OnboardingSecond z emulátoru
Zdroj: vlastní práce

Po kliknutí na tlačítko je přezdívka uložena do lokálního uložiště aplikace pomocí komponenty *SharedPreferences*, která ukládá data ve formátu klíč-hodnota a onboarding je označen za dokončený. Uživatel je poté přesměrován na domovskou obrazovku aplikace.

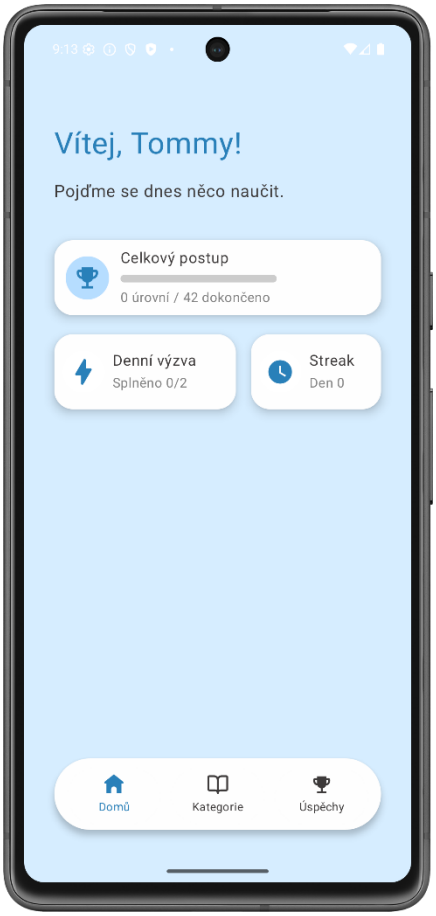
3.4.2 Domovská obrazovka

Domovská obrazovka představuje přehledné rozcestí, kde uživatel získává informace o svém postupu. V horní části se nachází uvítací text, který je personalizovaný podle zadané přezdívky v předchozím kroku. Pod úvodním textem je umístěna hlavní karta zobrazující celkový postup uživatele, která obsahuje:

- Grafické znázornění postupu pomocí progress baru.
- Informaci o počtu dokončených úrovní.
- Celkový počet úrovní v aplikaci.

Pod hlavní kartou se nacházejí karty Denní výzva a Streak. Obě karty slouží jako motivační nástroje. Všechny karty jsou zvýrazněné jemným stínem, kvůli oddělení od pozadí, což zlepšuje čitelnost.

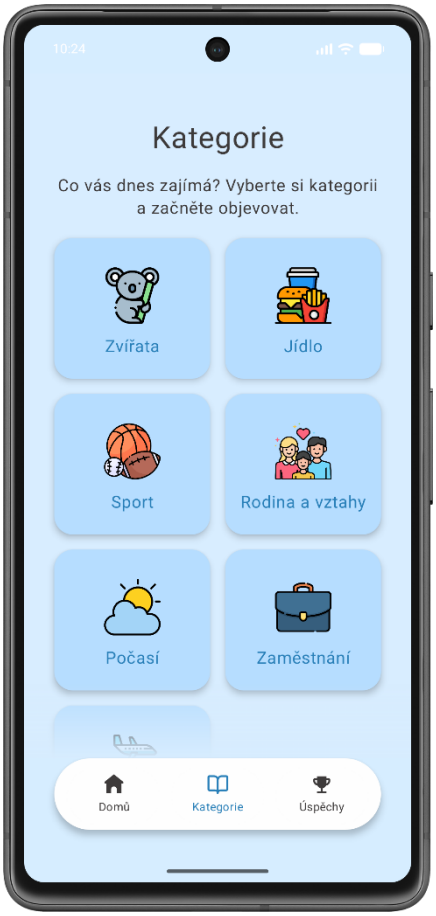
Ve spodní části obrazovky je umístěna navigační lišta (BottomNavigation), která umožňuje přechod mezi domovskou stránkou, kategoriemi a úspěchy. Vizuální indikace aktivní sekce je vyznačená modrou barvou.



Obr. 23: Náhled HomePage z emulátoru
Zdroj: vlastní práce

3.4.3 Kategorie

Obrazovka kategorií slouží k výběru tematického okruhu, ve kterém se chce uživatel vzdělávat. Hlavním prvkem rozhraní je mřížkové uspořádání (grid layout), které zajišťuje přehledné zobrazení. Každá karta obsahuje název kategorie a reprezentující obrázek, což umožňuje snadnější identifikaci tématu.



Obr. 24: Náhled CategoriesPage z emulátoru

Zdroj: vlastní práce

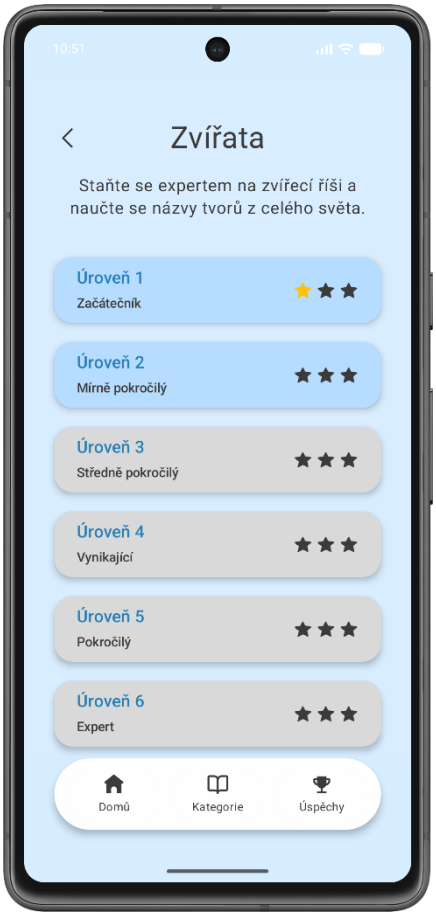
Vizuálního styl jednotlivých dlaždic je navržen se zaoblenými rohy a jemným stínem. Dané prvky vytvářejí dojem hloubky a oddělují interaktivní prvky od pozadí. Po kliknutí na kategorii dochází k přechodu na obrazovku s úrovněmi, kde může uživatel pokračovat ve studiu zvoleného tématu.

3.4.4 Výběr úrovní

Po výběru kategorie je uživatel přesměrován na obrazovku se seznamem jednotlivých úrovní, které jsou seřazeny podle obtížnosti. V horní části se nachází název zvoleného okruhu doplněný o krátký popis. Vedle názvu je umístěno tlačítko umožňující návrat zpět na přehled kategorií.

Každá úroveň je označena číslem a popisem obtížnosti. Uživateli je na začátku vždy dostupný pouze první úroveň a ostatní zůstávají uzamčené. Odemčení probíhá postupně na základě splnění předchozí položky.

Součástí je také vizuální indikátor ve formě hvězdiček. Pokud není daná položka dokončena, jsou všechny hvězdičky zobrazeny šedou barvou. Po splnění úrovně se zobrazí jedna až tři hvězdičky zlaté barvy podle dosažené úspěšnosti, která je vyhodnocena podle počtu chyb. Jednotlivé prvky jsou opět zvýrazněny jemným stínováním a odděleny dostatečnými rozestupy.



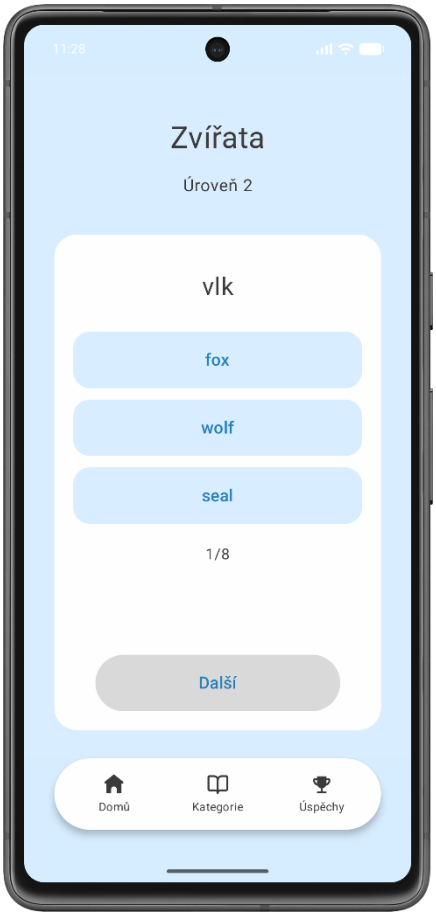
Obr. 25: Náhled LevelPage z emulátoru
Zdroj: vlastní práce

3.4.5 Zkoušení slovíček

Obrazovka zkoušení slovíček slouží k procvičování daného okruhu. V horní části se nachází název kategorie a označení aktuální úrovně.

Hlavní část rozhraní tvoří karta obsahující české slovíčko, ke kterému jsou přiřazeny tři anglické možnosti překladu, které jsou zobrazeny ve formě tlačítek a umožňují jednoduchý výběr kliknutím. Po zvolení jedné z možností dochází k jejímu zvýraznění. Volbu lze jednoduše změnit výběrem jiné odpovědi, což umožňuje opravu rozhodnutí před potvrzením a přechodem na další kartu.

Tlačítko „Další“ je neaktivní, pokud není vybrána žádná z nabízených možností, čímž je zamezeno pokračování bez zvolení odpovědi. Aktivace proběhne až po výběru jedné z možností a dojde tak ke změně barvy tlačítka. Součástí karty je také indikátor průběhu, který informuje o aktuální pozici v úrovni. Zápis ve formátu „1/8“ označuje pořadí zobrazeného slovíčka vzhledem k celkovému počtu. Po zodpovězení všech položek následuje přechod na obrazovku s vyhodnocením.



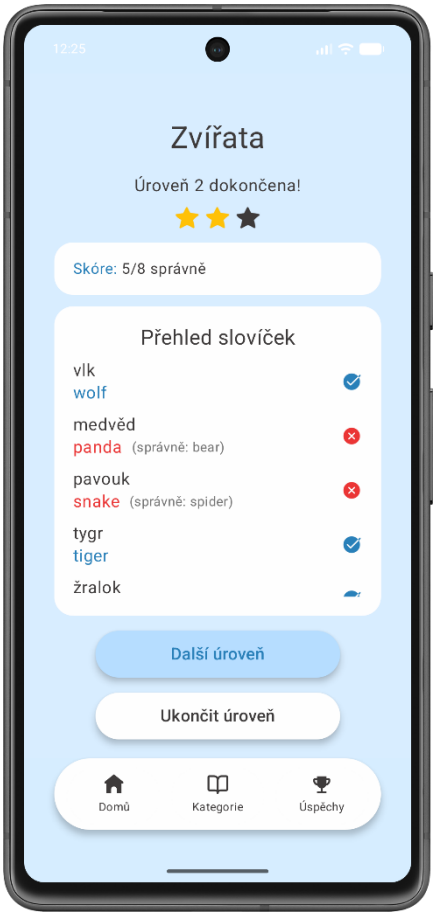
Obr. 26: Náhled QuizPage z emulátoru
Zdroj: vlastní práce

3.4.6 Přehled slovíček

Obrazovka se zobrazí po dokončení úrovně a slouží k přehledu dosažených výsledků. V horní části se nachází název okruhu doplněný o informaci dokončení úrovně. Následuje hodnocení pomocí hvězdiček, které vizuálně znázorňuje úspěšnost podle počtu chyb. Dále je zobrazena karta s ukazatelem celkového skóre, který informuje o počtu správných odpovědí. Daný údaj poskytuje rychlý přehled o dosaženém výsledku.

Hlavní část obrazovky tvoří karta s přehledem slovíček, kde jsou zobrazeny jednotlivé české výrazy se zvolenou odpovědí. V případě nesprávného výběru je uvedena správná varianta v závorce. Každý záznam je doplněn ikonou, která vizuálně rozlišuje správnou a chybnou odpověď. Seznam odpovědí umožňuje uživateli vertikální posun pro zobrazení kompletního přehledu.

Ve spodní části se nacházejí dvě tlačítka pro další akce. První umožňuje pokračovat na další úroveň. Druhé nabízí ukončení a návrat na domovskou obrazovku. Uživatelské rozhraní zachovává jednotný vizuální styl s ostatními obrazovkami.



Obr. 27: Náhled PreviewPage z emulátoru
Zdroj: vlastní práce

3.4.7 Úspěchy

Obrazovka úspěchů slouží jako motivační prvek. Rozhraní je vytvořeno pomocí mřížkového uspořádání (grid layout), které vizuálně odpovídá přehledu kategorií. Ve výchozím stavu jsou všechny dlaždice zobrazeny jako uzamčené, což je znázorněno šedým pozadím a ikonou zámku. Součástí je také krátká textová informace, která uživatele vyzývá k dokončení všech úrovní.

Po dokončení všech úrovní dané kategorie dochází k vizuálnímu odemčení úspěchu, které je znázorněno změnou barvy pozadí a změnou ikony zámku na zlatý pohár. Výsledná změna poskytuje okamžitou zpětnou vazbu uživateli.

Obr. 28: Náhled AchievementPage z emulátoru
Zdroj: vlastní práce

3.5 Navigace aplikace

Navigace v aplikaci zajišťuje plynulý přechod mezi jednotlivými obrazovkami a umožňuje snadnou orientaci v celém prostředí aplikace. Při vývoji byla použita navigační komponenta frameworku Jetpack Compose, která pracuje s principem definovaných cest (routes) mezi obrazovkami.

Důležitou součástí je také navigační lišta, která slouží jako hlavní rozcestník. Dále je v kapitole popsána struktura navigace, způsob přechodu mezi obrazovkami a řešení specifických situací, jako je ochrana proti nechtěnému opuštění rozpracovaného zkoušení.

3.5.1 Navigační struktura (NavHost)

Základ navigace je tvořen komponentou *NavHost*, která definuje vztahy mezi jednotlivými částmi aplikace. Funguje jako centrální prvek, ve kterém jsou registrovány všechny dostupné cesty (routes), díky kterým dochází k přepínání mezi jednotlivými stránkami. Součástí konfigurace je také určení výchozí obrazovky (startDestination) po spuštění aplikace.

Navigace je řízena pomocí objektu *NavController*, který zajišťuje samotné přechody. Každá stránka má přiřazenou unikátní identifikaci ve formě textového řetězce, díky které je možné se

na danou obrazovku přemístit. Definované cesty jsou navázány na konkrétní Composable funkce, které reprezentuje jednotlivé části rozhraní a určují jejich podobu.

V rámci aplikace jsou definovány následující cesty:

- domovská obrazovka,
- přehled kategorií,
- výběr úrovní,
- zkoušení slovíček,
- vyhodnocení výsledků,
- přehled úspěchů

Některé části vyžadují při navigaci předání dodatečných parametrů. Například obrazovka zkoušení, která potřebuje znát identifikátor úrovně, její číslo a název kategorie. Dané hodnoty jsou předávány jako součást navigační cesty.

Navržená struktura umožňuje jednoduché rozšiřování aplikace o další části bez nutnosti zásadních změn v kódu. Zároveň zajišťuje přehlednou organizaci navigační logiky v celé aplikaci.

```
NavHost(
    navController = navController,
    startDestination = "home", // výchozí obrazovka po spuštění appky
    modifier = Modifier.padding( paddingValues = padding)
) {

    // homepage
    composable( route = "home") {
        HomePage()
    }

    // kategorie
    composable( route = "categories") {
        CategoriesPage(navController)
    }

    // úspěchy
    composable( route = "achievements") {
        AchievementPage()
    }

    // levelpage (výběr úrovní)
    composable( route = "level/{category}") { backStackEntry ->

        // získání parametru z route (např. "Zvířata")
        val category = backStackEntry.arguments?.getString( key = "category")

        LevelPage(
            navController = navController,
            category = category
        )
    }
```

Obr. 29: Ukázka implementace NavHost

Zdroj: vlastní práce

3.5.2 Přechody mezi obrazovkami

Přechody mezi jednotlivými částmi aplikace jsou realizovány pomocí objektu *NavController*, který umožňuje navigaci na základě definovaných cest. K přesunu slouží metoda *navigate()*, do které je předána cílová cesta ve formě textového řetězce.

Navigace může probíhat bez parametrů, například při přechodu mezi hlavními sekcemi aplikace, nebo s parametry, které jsou součástí cesty. Dané parametry umožňují předat potřebná data mezi jednotlivými částmi aplikace. Například přechod do zkoušení slovíček, kde je nutné předat identifikátor úrovně, název kategorie a číslo úrovně.

```
val levelId = backStackEntry.arguments
    ?.getString( key = "levelId")!!
    .toInt()

val category = backStackEntry.arguments
    ?.getString( key = "category")!!

val levelNumber = backStackEntry.arguments
    ?.getString( key = "levelNumber")!!
    .toInt()
```

Obr. 30: Načtení parametrů z navigace

Zdroj: vlastní práce

Parametry jsou předány přímo v rámci navigační cesty a následně jsou získány pomocí objektu *NavBackStackEntry*. Dané hodnoty jsou poté využity pro načtení správných dat a zobrazení odpovídajícího obsahu.

Součástí navigace je také možnost návratu na předchozí část aplikace pomocí metody *popBackStack()*. Daná metoda umožňuje vrátit se zpět v navigační historii bez nutnosti znovu vytvářet jednotlivé části rozhraní.

V případě přechodu mezi hlavními sekcemi pomocí spodní navigační lišty je využito rozšířené nastavení navigace, které zabraňuje vytváření duplicitních instancí. Pomocí parametrů *popUpTo* a *launchSingleTop* je zajištěna přehlednost navigačního zásobníku a optimalizované chování aplikace.

3.5.3 Navigační lišta (BottomNavigation)

Navigační lišta slouží jako hlavní prvek pro přechod mezi základními obrazovkami aplikace. Je propojena s objektem *NavController*, který zajišťuje přepínání mezi jednotlivými částmi aplikace. Pro správné fungování je nejprve sledována aktuální navigační cesta, díky které lze určit, která část aplikace je právě aktivní.

```
val backStackEntry by navController.currentBackStackEntryAsState()
val currentRoute = backStackEntry?.destination?.route
```

Obr. 31: Ukázka kódu aktuální navigační cesty

Zdroj: vlastní práce

Samotná navigace je řešena pomocnou funkcí, která sjednocuje způsob přechodu mezi jednotlivými částmi aplikace. Funkce zároveň umožňuje zachytit speciální situace, kdy není vhodné provést přechod okamžitě.

```
fun navigate(route: String) {

    if (onNavigateRequest != null) {
        // pokud jsme v QuizPage neodcházíme hned ale pošleme požadavek
        // (zobrazí se dialog)
        onNavigateRequest(route)

    } else {
        // klasická navigace
        navController.navigate(route) {
            popUpTo( route = "home")      // vyčistí stack až po home
            launchSingleTop = true // zabrání duplicitám
        }
    }
}
```

Obr. 32: Vlastní funkce pro navigaci

Zdroj: vlastní práce

V případě, že není definována speciální navigační akce, dojde ke standardnímu přechodu pomocí *NavController*. Parametry *popUpTo* a *launchSingleTop* zajišťují, že nedochází k vytváření duplicitních instancí a navigační zásobník zůstává přehledný. Jednotlivé položky navigační lišty následně využívají danou funkci pro přechod mezi hlavními částmi aplikace.

```
onClick = { navigate( route = "home") }
onClick = { navigate( route = "categories") }
onClick = { navigate( route = "achievements") }
```

Obr. 33: Ukázka volání navigační funkce

Zdroj: vlastní práce

Důležitou součástí je také rozšířená logika, která se uplatňuje během zkoušení slovíček. V daném případě je navigace zachycena pomocí vlastního parametru *onNavigateRequest*, který umožňuje přerušit standardní přechod a reagovat na něj jiným způsobem.

Uvedený mechanismus slouží jako ochrana proti nechtěnému opuštění rozpracované úrovně. Namísto okamžitého přechodu je vyvolána potvrzovací akce ve formě dialogu, ve kterém se rozhoduje o dalším postupu. Uživatel tak má možnost buď pokračovat ve zkoušení, nebo aplikaci opustit bez ztráty kontroly nad svým rozhodnutím.

3.6 Možnosti rozšíření

Navržená aplikace poskytuje mnoho možností pro budoucí rozšíření funkcionality. Díky zvolené architektuře je možné aplikaci postupně rozšiřovat.

Jedna z možností je zavedení uživatelského účtu. V aktuální verzi jsou data ukládána lokálně v zařízení. Implementace uživatelských účtů by umožnila synchronizaci dat mezi více zařízeními a zálohování postupu. Uvedené řešení je spojeno s přechodem na online databázi, například

využití služby Firebase, která umožňuje ukládání dat do cloudu. Zavedení zmíněné funkcionality by znamenalo značné rozšíření aplikace o síťovou komunikaci.

Další možností je úprava systému zkoušení, která by umožnila opakované zkoušení dokončených úrovní. Uživatel by mohl úrovně opakovat vícekrát a zlepšovat tak své výsledky. Pro uvedenou úpravu by bylo vhodné rozšíření databáze o větší množství slovíček a zároveň zajistit variabilitu otázek, které lze dosáhnout náhodným výběrem slovíček z databáze a generováním různých kombinací českých výrazů a odpovědí. Daný způsob by zajistil, že se jednotlivé pokusy nebudou opakovat ve stejné podobě.

Mezi další možnosti patří například doplnění nových kategorií. Struktura databáze i aplikace je momentálně navržena tak, aby bylo možné jednoduše přidávat nové kategorie a úrovně bez zásahu do aplikační logiky. Zmíněné rozšíření by vedlo k delší využitelnosti aplikace.

Aplikaci také lze rozšířit o další cizí jazyky. V aktuální podobě je zaměřena na překlad z češtiny do angličtiny. V budoucnu by bylo možné přidat další jazyky, například němčinu, což by vedlo k úpravě datové struktury a způsobu ukládání slovíček.

Navržené řešení aplikace poskytuje flexibilní základ, který umožňuje další rozvoj podle budoucích požadavků a potřeb uživatelů. Díky přehledné architektuře a oddělení jednotlivých vrstev je možné implementovat nové funkce postupně a bez zásadních komplikací.

3.7 Zhodnocení realizace práce

Během implementace aplikace se neobjevily zásadní problémy, které by znemožnily pokračování vývoje, nicméně bylo nutné řešit několik komplikací souvisejících především s návrhem aplikační logiky a správou dat. Jednou z řešených oblastí bylo zajištění správného ukládání a načítání dat v databázi Room. V průběhu vývoje byly zvažovány různé způsoby práce s entitami, přičemž původní přístup se ukázal jako méně přehledný a vedl k opakování kódu při vytváření jednotlivých záznamů. Následně byl zvolen jednodušší a efektivnější způsob ukládání za použití pomocné funkce, který lépe odpovídá zvolené architektuře a zároveň usnadňuje práci s daty. Zároveň usnadnil budoucí úpravy datové struktury, protože změny je možné provádět na jednom místě bez nutnosti zásahu do větší části kódu.

Další oblastí, která vyžadovala úpravu, byl výpočet streaku. Původní řešení bylo založeno na práci s časem pomocí třídy *Date*, kdy byl interval jednoho dne nastaven pevně danou hodnotou v milisekundách. Daný přístup se ukázal jako nevhodný, protože docházelo k resetování streaku při překročení časového limitu ihned po půlnoci. Uživatel tak neměl možnost ve streaku pokračovat následující den. Pro odstranění uvedeného problému byl navržen nový přístup, který pracuje s kalendářními dny. Výsledné řešení používá třídy *Calendar* a *SimpleDateFormat*, které umožňují přesné porovnání jednotlivých dní bez ohledu na konkrétní čas. Data jsou načtena z databáze Room, seřazena podle data a následně je ověřována jejich návaznost na aktuální den. Algoritmus postupně kontroluje, zda na sebe jednotlivé kalendářní dny navazují a zda v daný den došlo k dokončení alespoň jedné úrovně.

Zvolený postup implementace byl orientován na jednoduchost, přehlednost a efektivitu vývoje. Cílem bylo ověřit, že i aplikaci podobného rozsahu lze v jazyce Kotlin navrhnout a realizovat bez zbytečné složitosti při zachování kvalitní struktury kódu. Využití architektury MVVM v kombinaci

s nástroji platformy Android umožnilo oddělit jednotlivé vrstvy aplikace a usnadnilo orientaci v kódu i budoucí možnosti rozšíření.

Při implementaci uživatelského rozhraní nebylo cílem dodržovat doporučení Material Design, ale vytvořit vlastní design vycházející z návrhu UI v nástroji pro grafický design Figma. Zvolený přístup poskytl větší volnost při návrhu komponent a umožnil vytvořit konzistentní, čisté a vizuálně klidné prostředí. Výsledné řešení podporuje soustředění uživatele při učení a zároveň umožňuje lépe přizpůsobit vzhled konkrétním požadavkům aplikace. Implementace vlastního návrhu proběhla bez komplikací díky využití nástroje Jetpack Compose, kdy bylo možné jednotlivé prvky uživatelského rozhraní flexibilně přizpůsobovat a upravovat jejich vzhled přímo v kódu.

Realizace práce ukázala, že zvolený přístup je vhodný pro vývoj menších až středně rozsáhlých mobilních aplikací. Kombinace jednoduché implementace, přehledné struktury a využití moderních nástrojů vede k řešení, které je nejen funkční, ale také dobře udržovatelné a připravené na případné rozšíření. Zvolená architektura zároveň umožňuje snadnou orientaci v kódu a efektivní práci s jednotlivými částmi aplikace, což je důležité zejména při dalším vývoji nebo úpravách.

4 Testování aplikace

Testování aplikace bylo provedeno v prostředí Android Studio za využití emulátoru na virtuálních zařízeních Pixel 7, Pixel 9 Pro a dále na reálném mobilním zařízení Xiaomi Redmi 14C. Cílem testování bylo ověřit správné zobrazení uživatelského rozhraní na různých typech zařízení a zároveň správné funkčnosti aplikace a implementovaných funkcí.

4.1 Testování uživatelského rozhraní

V rámci testování uživatelského rozhraní bylo ověřeno správné zobrazení jednotlivých částí aplikace na různých zařízeních s odlišnými rozměry a poměry stran obrazovky. Na základě testování byly provedeny drobné úpravy, které se týkaly odsazení karet denní výzvy a streaku pro zajištění lepší vizuální rovnováhy. Dále byl identifikován problém s pozicí navigační lišty, která na zařízení Xiaomi Redmi 14C zasahovala do spodního systémového ovládacího panelu. Poslední úpravy se týkaly obrazovky úspěchů, kde byl text v jednotlivých dlaždicích při různých rozlišení obrazovek původně zarovnán vlevo. Následně byl upraven na středové zarovnání, čímž došlo ke zlepšení vizuální konzistence uživatelského rozhraní.

4.2 Testování funkčnosti

Testování funkčnosti bylo zaměřeno na ověření správného průběhu zkoušení, ukládání dat a aktualizace uživatelského rozhraní.

Nejprve byla testována funkčnost zkoušení slovíček, zejména správné vyhodnocení odpovědí a plynulý přechod mezi jednotlivými otázkami. Součástí bylo také ověření náhodného pořadí odpovědí, aby nedocházelo k zobrazování správné odpovědi vždy na stejné pozici. Pozornost byla dále věnována ukládání výsledků do databáze a jejich následnému načítání. Současně bylo potvrzeno, že po dokončení celé kategorie dochází ke správnému odemykání jednotlivých úspěchů.

Další část testování se zaměřila na sledování denní aktivity. Při testování byl identifikován problém, kdy se při spuštění aplikace zobrazovala hodnota streaku nula, i když uživatel v předchozích dnech plnil jednotlivé úrovně. Uvedený stav poukazoval na nedostatečnou inicializaci dat při spuštění aplikace. Po úpravě logiky výpočtu a načítání dat při startu aplikace byla funkčnost znovu ověřena a následně se hodnota streaku zobrazovala správně.

Pozornost byla věnována také aktualizaci uživatelského rozhraní v návaznosti na změny dat. Bylo ověřeno, že po dokončení úrovně dochází k okamžité aktualizaci hodnot, jako je celkový postup a denní výzva. Díky využití nástroje StateFlow se změny automaticky promítají do uživatelského rozhraní.

Závěr

Teoretická část práce se zaměřila na principy a technologie potřebné pro vývoj moderních Android aplikací v programovacím jazyce Kotlin. Cílem bylo vytvořit teoretický základ pro návrh a realizaci mobilní aplikace pro zkoušení slovíček.

Nejprve byly popsány nedostatky tradiční architektury Android aplikací a princip oddělení zájmů jako klíčové východisko pro návrh kvalitního a udržovatelného řešení. Na dané poznatky navázal popis architektonického vzoru Model–View–ViewModel, včetně role vrstvy ViewModel a využití knihoven Android Jetpack při správě životního cyklu a stavu aplikace.

Další část byla věnována databázovým řešením v prostředí Android se zaměřením na knihovnu Room Persistence Library. Byla popsána architektura knihovny, integrace s jazykem Kotlin a podpora asynchronních databázových operací pomocí Kotlin Coroutines, které jsou nezbytné pro zachování plynulosti uživatelského rozhraní.

Na základě získaných teoretických znalostí byla v praktické části navržena a implementována mobilní aplikace pro zkoušení slovíček. Aplikace umožňuje uživateli procvičovat slovíčka prostřednictvím krátkých testovacích úloh, sledovat svůj pokrok, denní aktivitu a využívat aplikaci bez připojení k internetu.

Cíl práce, kterým bylo vytvoření funkční mobilní aplikace pro platformu Android s využitím moderních technologií, byl splněn. Implementované řešení odpovídá stanoveným požadavkům a představuje praktické využití teoretických principů v reálném vývoji. Testování potvrdilo správnou funkčnost jednotlivých částí aplikace i konzistentní chování uživatelského rozhraní na různých zařízeních.

Přínosem práce je vytvoření mobilní aplikace, která slouží jako jednoduchý nástroj pro procvičování slovní zásoby a zároveň ukazuje využití moderních přístupů při vývoji Android aplikací. Navržené řešení klade důraz na přehledné a vlastní (custom) uživatelské rozhraní, které je navrženo s cílem působit nerušivě a podporovat soustředěné a klidné prostředí pro učení. Důležitým aspektem je intuitivní ovládání a možnost používání bez připojení k internetu, což zvyšuje jeho praktickou využitelnost v každodenním životě. Práce současně reaguje na problém spojený s obtížným zapamatováním slovní zásoby při nepravidelném používání cizího jazyka. Na rozdíl od již existujících aplikací je řešení zaměřeno na okamžitou použitelnost bez nutnosti registrace nebo složitého nastavování. Aplikace nabízí efektivní způsob opakování prostřednictvím krátkých testovacích úloh, které podporují pravidelné procvičování a pomáhají uživateli upevňovat znalosti. Přínosem je také využití odložené zpětné vazby po dokončení úrovně, která nenarušuje plynulost zkoušení a implementace sledování pokroku a denní aktivity, které působí jako motivační prvek pro dlouhodobé používání aplikace.

Do budoucna je možné aplikaci dále rozšířit například o podporu uživatelských účtů a synchronizaci dat, rozšíření databáze slovíček, přidání dalších cizích jazyků nebo implementaci pokročilejších metod učení. Navržená architektura umožňuje následující úpravy realizovat bez zásadních změn stávajícího řešení.

Seznam použité literatury

ANDROID DEVELOPERS. Activity lifecycle. Online. 2025-02-10. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle. [cit. 2026-04-14].

ANDROID DEVELOPERS. Android 7.0 for Developers. Online. 2024-05-20. Dostupné z: https://developer.android.com/about/versions/nougat/android-7.0. [cit. 2026-04-14].

ANDROID DEVELOPERS. Android KTX. Online. 2026-04-08. Dostupné z: https://developer.android.com/kotlin/ktx. [cit. 2026-04-14].

ANDROID DEVELOPERS. android.database.sqlite. Online. 2025-09-17. Dostupné z: https://developer.android.com/reference/android/database/sqlite/package-summary. [cit. 2026-04-14].

ANDROID DEVELOPERS. Android's Kotlin-first approach. Online. 2026-03-06. Dostupné z: https://developer.android.com/kotlin/first. [cit. 2026-04-14].

ANDROID DEVELOPERS. Application fundamentals. Online. 2025-02-10. Dostupné z: https://developer.android.com/guide/components/fundamentals. [cit. 2026-04-14].

ANDROID DEVELOPERS. Data and file storage overview. Online. 2026-03-30. Dostupné z: https://developer.android.com/training/data-storage. [cit. 2026-04-14].

ANDROID DEVELOPERS. Getting started with Android Jetpack. Online. 2026-03-06. Dostupné z: https://developer.android.com/jetpack/getting-started. [cit. 2026-04-14].

ANDROID DEVELOPERS. Guide to app architecture. Online. 2026-03-05. Dostupné z: https://developer.android.com/topic/architecture. [cit. 2026-04-14].

ANDROID DEVELOPERS. Handling lifecycles with lifecycle-aware components. Online. 2026-03-05. Dostupné z: https://developer.android.com/topic/libraries/architecture/lifecycle. [cit. 2026-04-14].

ANDROID DEVELOPERS. Kotlin coroutines on Android. Online. 2026-03-06. Dostupné z: https://developer.android.com/kotlin/coroutines. [cit. 2026-04-14].

ANDROID DEVELOPERS. Platform architecture. Online. 2024-05-20. Dostupné z: https://developer.android.com/guide/platform. [cit. 2026-04-14].

ANDROID DEVELOPERS. Processes and threads overview. Online. 2024-01-03. Dostupné z: https://developer.android.com/guide/components/processes-and-threads. [cit. 2026-04-14].

ANDROID DEVELOPERS. Recommendations for Android architecture. Online. 2026-04-02. Dostupné z: https://developer.android.com/topic/architecture/recommendations. [cit. 2026-04-14].

ANDROID DEVELOPERS. Save data in a local database using Room. Online. 2026-03-05. Dostupné z: https://developer.android.com/training/data-storage/room. [cit. 2026-04-14].

ANDROID DEVELOPERS. ViewModel overview. Online. 2026-03-05. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. Android Runtime and Dalvik. Online. 2024-08-26. Dostupné z: https://source.android.com/docs/core/runtime. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. Android security features. Online. 2026-04-10. Dostupné z: https://source.android.com/docs/security/features. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. AOSP overview. Online. 2026-04-10. Dostupné z: https://source.android.com/docs/setup/about. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. Application Sandbox. Online. 2026-04-10. Dostupné z: https://source.android.com/docs/security/app-sandbox. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. Dalvik bytecode. Online. 2024-08-26. Dostupné z: https://source.android.com/docs/core/runtime/dalvik-bytecode. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. HAL overview. Online. 2026-04-10. Dostupné z: https://source.android.com/docs/core/architecture/hal. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. JIT compiler. Online. 2026-04-10. Dostupné z: https://source.android.com/docs/core/runtime/jit-compiler. [cit. 2026-04-14].

ANDROID OPEN SOURCE PROJECT. Kernel overview. Online. 2026-04-10. Dostupné z: https://source.android.com/docs/core/architecture/kernel. [cit. 2026-04-14].

ARTICULATE. What Is Microlearning and Why Is It Effective? Online. 2025-03-10. Dostupné z: https://www.articulate.com/blog/what-is-microlearning-and-why-is-it-effective/. [cit. 2026-04-14].

BABBEL. Babbel. Online. Dostupné z: https://www.babbel.com/. [cit. 2026-04-14].

DUOLINGO. Duolingo. Online. Dostupné z: https://www.duolingo.com/. [cit. 2026-04-14].

ISPRING. What Is Microlearning? Online. 2026-03-19. Dostupné z: https://www.ispring.com/knowledge-hub/what-is-microlearning. [cit. 2026-04-14].

KOTLIN DOCUMENTATION. Calling Java from Kotlin. Online. 2026-03-16. Dostupné z: https://kotlinlang.org/docs/java-interop.html. [cit. 2026-04-14].

KOTLIN DOCUMENTATION. Coroutines. Online. 2025-08-26. Dostupné z: https://kotlinlang.org/docs/coroutines-overview.html. [cit. 2026-04-14].

KOTLIN DOCUMENTATION. Data classes. Online. 2026-03-14. Dostupné z: https://kotlinlang.org/docs/data-classes.html. [cit. 2026-04-14].

KOTLIN DOCUMENTATION. Extensions. Online. 2025-11-04. Dostupné z: https://kotlinlang.org/docs/extensions.html. [cit. 2026-04-14].

KOTLIN DOCUMENTATION. FAQ. Online. 2026-04-02. Dostupné z: https://kotlinlang.org/docs/faq.html. [cit. 2026-04-14].

KOTLIN DOCUMENTATION. Null safety. Online. 2025-08-28. Dostupné z: https://kotlinlang.org/docs/null-safety.html. [cit. 2026-04-14].

MEMRISE. Memrise. Online. Dostupné z: https://www.memrise.com/. [cit. 2026-04-14].

PCMAG. Babbel Review. Online. 2025-08-11. Dostupné z: https://www.pcmag.com/reviews/babbel. [cit. 2026-04-14].

PCMAG. Duolingo Review. Online. 2025-07-23. Dostupné z: https://www.pcmag.com/reviews/duolingo. [cit. 2026-04-14].

PCMAG. Quizlet Review. Online. 2024-09-26. Dostupné z: https://www.pcmag.com/reviews/quizlet. [cit. 2026-04-14].

QUIZLET. Quizlet. Online. Dostupné z: https://quizlet.com/. [cit. 2026-04-14].

STRIVECLOUD. Gamification examples: Duolingo. Online. Dostupné z: https://strivecloud.io/blog/gamification-examples-boost-user-retention-duolingo. [cit. 2026-04-14].

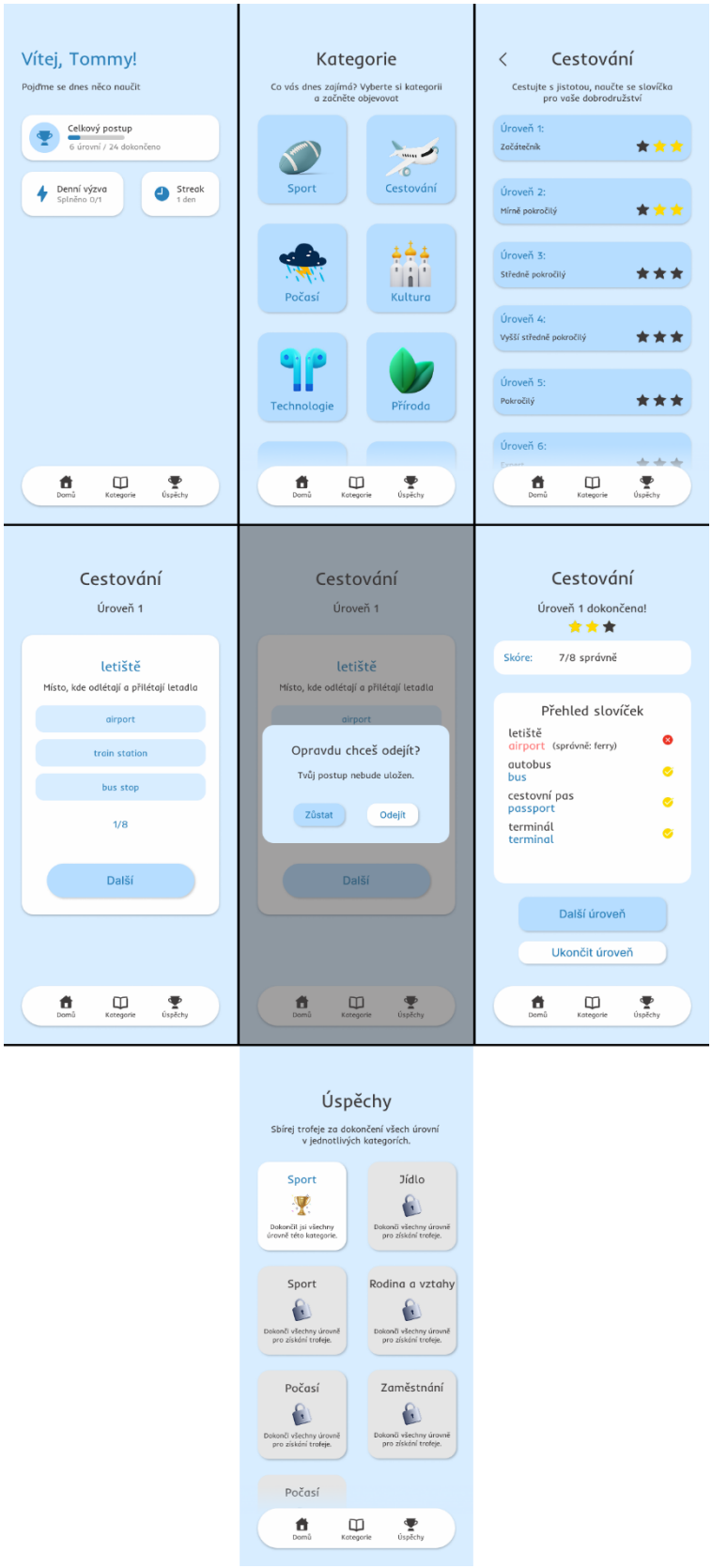
Příloha A Projekt aplikace

Zdrojový kód aplikace je přiložen v elektronické podobě jako samostatná příloha v informačním systému VŠPJ.

Příloha B APK soubor aplikace

Instalační soubor mobilní aplikace ve formátu .apk je přiložen v elektronické podobě jako samostatná příloha v informačním systému VŠPJ.

Příloha C Návrh UI



Obr. 34: Kompletní návrh UI

Zdroj: vlastní práce