ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

Fakulta elektrotechnická
Katedra počítačů

Návrh a implementace podpůrné aplikace pro využití v rámci logických her „šifrovačka“

Kamila Foltýnová

Vedoucí: Ing. Pavel Náplava, Ph.D.

Studijní program: Softwarové inženýrství a technologie

Specializace: Technologie pro multimédia a virtuální realitu

Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Foltýnová** Jméno: **Kamila** Osobní číslo: **493214**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Technologie pro multimédia a virtuální realitu**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Návrh a implementace podpůrné aplikace pro využití v rámci logických her „šifrovačka“

Název bakalářské práce anglicky:

Design and implementation of a support application for use in logic games šifrovačka“

Pokyny pro vypracování:

Navrhněte aplikaci, která bude sloužit jako pomůcka účastníkům logických her, pracovně nazývaných „Šifrovačka“. Postupujte následujícím způsobem:
1) Definujte pojmy logická hra, účastník hry, šifra, typy šifer atd.
2) Analyzujte logickou hru "Šifrovačka", její průběh a potřeby jejích účastníků.
3) Navrhněte, jakým způsobem lze účastníky během hry podporovat a současně jim hru nezjednodušovat. Vyjděte ze současné „papírové“ podpory.
4) Navrhněte a vytvořte mobilní aplikaci, která nahradí stávající „papírovou“ podporu. Tj. nabídne uživateli náhled na principy vybraných šifer. Aplikaci realizujte ve verzi, která bude podporovat minimálně 4 způsoby "šifrování". Při návrhu a realizaci se zaměřte na jednoduchost, použitelnost v terénu a vzhled aplikace.
5) Vytvořenou aplikaci uživatelsky ověřte na vybrané skupině jejích potenciálních uživatelů.

Seznam doporučené literatury:

[1] VONDRUŠKA, Pavel. Kryptologie, šifrování a tajná písma. Praha: Albatros, 2006. ISBN 80-00-01888-8.
[2] LAUŠ, Jan. Moderní kryptoanalytické metody. Brno, 2007. Diplomová práce. Masarykova Univerzita, Fakulta informatiky.
[3] Šifrování dat. Online. 2022. Dostupné z: https://www.sprava-site.eu/sifrovani-dat/.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Náplava, Ph.D. Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Pavel Náplava, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studentky

Poděkování

Chtěla bych jako první poděkovat svému vedoucímu Ing. Pavlu Náplavovi Ph.D. za pomoc při tvorbě práce.
Děkuji Lucienovi a týmu Omnes Virtutes, s nimiž jsem si šifrovačky zamilovala.
Jsem zavázána všem svým testerům, kteří mi pomohli dotáhnou aplikaci do dokonalosti, a především kamarádce An Ho, která mi pomáhala s grafickým designem.
Děkuji Vojtovi a ostatním, že se chtějí společně učit, jinak by se mi studovalo mnohem náročněji.

Nakonec chci poděkovat své rodině, která mě v náročném studiu podporovala a držela mi palce. Věnováno tátovi, byl by na mě hrdý.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně, a že jsem uvedla veškerou použitou literaturu.

V Praze, 24. května 2024

Abstrakt

Cílem této bakalářské práce je vytvoření mobilní aplikace, která dekóduje různé typy šifer používaných na soutěžích s názvem „šifrovačka“. Umožní zrychlení manuální části práce dešifrování. Aplikace klade důraz na design a uživatelskou přívětivost a měla by být dostupná co nejvíce soutěžícím. Uživatelská přívětivost je testována dvěma skupinami lidí s odlišnou úrovní zkušeností s šifrovacími hrami, aby se pokryla co největší cílová skupina. Aplikace je naprogramovaná ve frameworku Flutter programovacího jazyka Dart, což zajišťuje její funkčnost na různých platformách.

Klíčová slova: šifrovačka, šifrovací pomůcky, mobilní aplikace, Flutter, framework, Dart, design uživatelského rozhraní

Vedoucí: Ing. Pavel Náplava, Ph.D.

Abstract

The goal of this theses is to create a mobile app which decodes multiple types of cyphers used in logic games called „puzzle hunts“. It will allow to speed up manual part of the solving process. The app puts emphasis on design and user friendliness and should be accessible to as many people as possible. User accessibility is tested by two groups of people with different levels of expertise with puzzle hunts so it covers majority of the target audience. Application is coded in Flutter, which is a framework of the Dart programming language, which ensures its functionality across multiple platforms.

Keywords: šifrovačka, cypher tools, mobile app, Flutter, framework, Dart, user interface design

Supervisor: Ing. Pavel Náplava, Ph.D.

Obsah

1. Úvod

V zájmových aktivitách můžeme ve větší či menší míře pozorovat určitou soutěživost. Ať se jedná o sport, hry deskové, počítačové nebo vědomostní. Lidé se chtějí zlepšovat, porovnávat svoje schopnosti s ostatními a vyhrávat. Případně pouze sledovat ty nejlepší z nejlepších, nadlidské výkony jsou neskutečné a vzrušující. Emoce se během relativní chvilky napínají na maximum a v tom je něco návykového.

1.1. Cíl práce

Cílem práce je vyvinout mobilní aplikaci, která by mohla být užitečná účastníkům specifickým soutěží zvaných „šifrovačka“ a zrychlit jim tím proces luštění. Za tímto účelem je nezbytné provést podrobnou analýzu těchto logických her a jednotlivých šifer v nich používaných, což pomáhá identifikovat klíčové procesy potřebné pro vytvoření účinné podpory účastníků prostřednictvím mobilní aplikace. Následně bude navržena mobilní aplikace s důrazem na intuitivní a jednoduché používání, přizpůsobená potřebám uživatelů a zaměřená na zrychlení manuálního dekódování již vymyšlených řešení šifer, aniž by aplikace řešila úlohy samy o sobě. Návrh aplikace má dbát na intuici a jednoduchost používání, což je následně potřeba ověřit při testování. Výsledkem práce bude tedy účinný nástroj šitý na míru.

1.2. Motivace

Tato práce vzniká z mé osobní zkušenosti a nadšení pro šifrovací soutěže, kterých se pravidelně účastním již několik let. Mým snem je přilákat širší veřejnost k této zábavné aktivitě a zároveň přispět k růstu šifrovací komunity. Ráda bych, aby nová mobilní aplikace usnadnila a zpříjemnila průběh těchto soutěží pro všechny nadšence (včetně mě samotné), nebo přilákala nové tváře.

V současnosti se účastníci při luštění šifer spoléhají převážně na papírové tabulky, které fungují podobně jako slovníky s tím rozdílem, že překládají místo slov symboly a obrázky. Tento proces je však pomalý a neefektivní. Mobilní aplikace by mohla tento problém vyřešit tím, že by poskytovala rychlejší a pohodlnější způsob dešifrování, podobně jako elektronické překladače oproti tradičním knižním slovníkům.

Vzhledem k tomu, že čas hraje v šifrovacích soutěžích klíčovou roli, by rychlejší metody luštění ocenili nejen začátečníci, ale i zkušení účastníci. Vytvoření takovéto aplikace proto považuji za významný krok k modernizaci těchto logických her, a věřím, že by mohla přinést komfort všem hráčům.

2. Definice pojmů

Tato kapitola se zaměřuje na vysvětlení základních pojmů spojených s šifrovačkami, aby poskytla lepší představu o jejich průběhu a charakteru. Informace jsou čerpány z osobních zkušeností a analýzy šifrovaček z let 2019–2023. Data byla získána přímo z webových stránek jednotlivých šifrovaček, na které odkazuje web www.sifrovacky.cz. V jednotlivých sekcích této kapitoly jsou uvedeny klíčové postřehy a doporučení pro následný vývoj získaná během výzkumu a analýzy dat.

2.1. Logická hra „šifrovačka“

Šifrovačka je logická soutěž, jejímž cílem je co nejrychleji projít trasu od startu do cíle. Tyto hry organizují různé skupiny nadšenců a neexistuje jednotná asociace, která by stanovovala pravidla, proto se jednotlivé šifrovačky mohou výrazně lišit. Přesto mají společné základní principy a průběh, připomínající táborové hry typu honby za pokladem. Trasa je účastníkům neznámá a je rozdělena do několika úseků s úkoly, nazývanými „šifra“ (viz sekce *Šifra*). Po vyřešení úlohy se účastníci dozví polohu dalšího stanoviště na trase. Pokud je šifra příliš obtížná, mohou využít nápovědy, což je však penalizováno. Šifrovačky tedy nejsou pouze o rychlosti, ale také o důvtipu a logickém uvažování.

2.1.1. Charakter šifrovaček

Pro konkrétní obrázek společných znaků šifrovaček byla sesbírána a porovnána data ze 135 soutěží. Typické znaky je důležité identifikovat z důvodu, aby aplikace byla maximálně přizpůsobena potřebám soutěžících. Soutěže byly porovnávány v těchto kategoriích: počet šifer, počet účastníků, délka trasy, zda je internetová nebo noční, a kdy se šifrovačka konala. Soubor s kompletními daty je přiložen v příloze, kde lze najít i soubor s vytvořenými grafy.

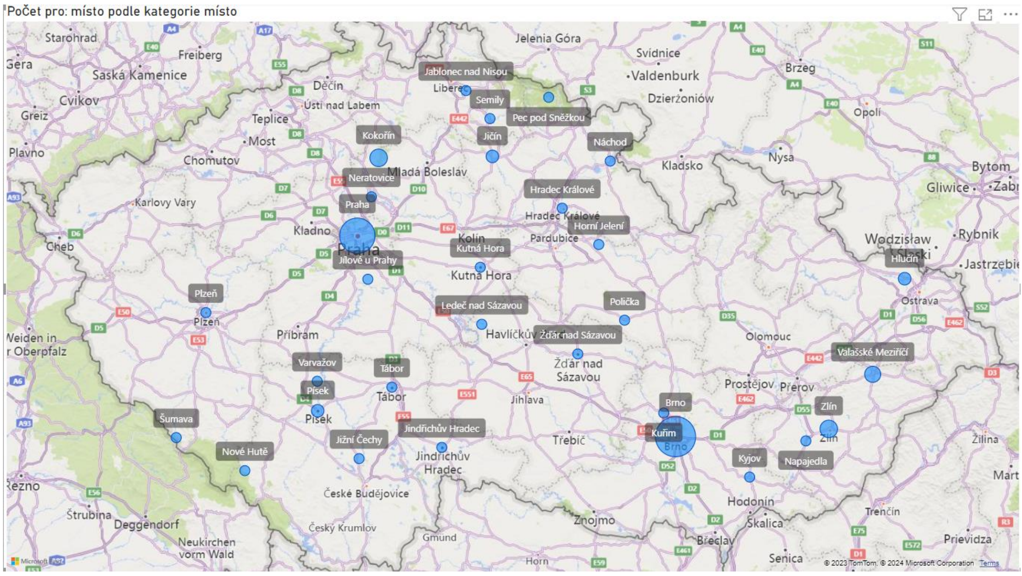
2.1.1.1. Český fenomén?

Portál www.sifrovacky.cz slouží k propagování her a zájemcům nabízí kalendář akcí, na které se mohou přihlásit. Na tomto webu si kdokoliv může přidat svou vlastní šifrovací soutěž, včetně organizátorů ze zahraničí. Mezi známé zahraniční šifrovací akce patří například MIT Mystery Hunt1, která se koná každoročně na Massachusetts Institute of Technology (MIT) v USA nebo Microsoft Puzzle Hunt pořádanou společností Microsoft.

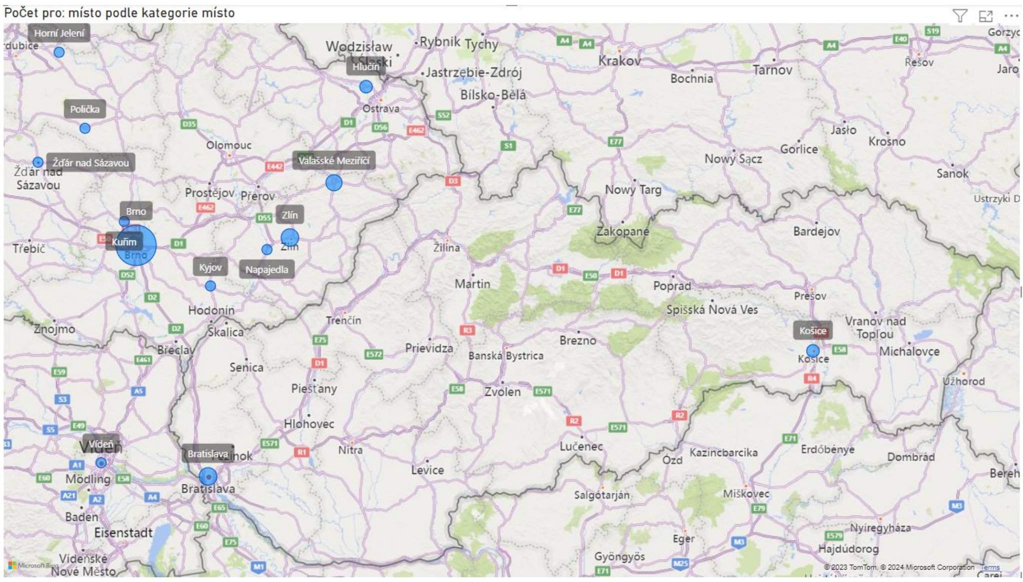
Na obrázcích 2.1 a 2.2 níže jsou ze sesbíraného souhrnu dat na vyznačena místa konání šifrovaček s ohledem na jejich četnost – čím více akcí, tím větší bublina.

Tento i ostatní grafy v této práci byly z nasbíraných dat vytvořeny v aplikaci Microsoft Power-BI.

[1] Pro více informací navštivte: https://puzzles.mit.edu



Obrázek 2.1: Geografický graf České republiky počtu konaných akcí. Zdroj: autorka práce



Obrázek 2.1: Geografický graf Slovenské republiky počtu konaných akcí. Zdroj: autorka práce

Komentář ke geografickým grafům 2.1 a 2.2

Přestože bylo řečeno, že se šifrovačky nekonají pouze v české komunitě, z map 2.1 a 2.2 výše vyplývá opak. Proč se soutěže konané v sousedních státech nepropagují na jediném centralizovaném portálu české společnosti? Je to z důvodu obsahu samotných šifer – kromě logických úloh také často vycházejí ze znalostí lokální popkultury a lingvistických prostředků, což je pro zahraniční účastníky náročné. Díky podobné historii a jazykům je česká komunita propojena s tou slovenskou, proto se některé evidované akce konají i tam.

Poznatky k vývoji aplikace

Přestože šifrovačky nejsou pouze českým fenoménem, tato práce se věnuje českým (konkrétněji česko-slovenským) šifrovačkám a potřebám k jejich řešení.

2.1.1.2. V jakém prostředí se šifrovačky konají?

Jak bylo avizováno, šifrovačky jsou rozdělené do několika úseků trasy – jsou tedy zpravidla konány venku. To přidává soutěžím prvek pěšího výletu, hry jsou situovány, jak v přírodě, tak ve městě a plánovaná cesta často křižuje neznámá a krásná místa, která by možná člověk sám od sebe nenavštívil. Trasy mívají obvykle asi16 kilometrů. Toto číslo není úplně přesné, jelikož informaci o délce tratě uvádělo na svém webu pouze 16 soutěží. Na první pohled se může vzdálenost startu a cíle zdát velká, má to však své opodstatnění. Stanoviště nesmí být blízko sebe, aby týmy na předchozím zastavení neviděly, kam musí jít, aniž by úkol vyřešily. Akce je tedy časově náročná, kromě doby luštění šifer stráví účastníci několik hodin pouhou chůzí.

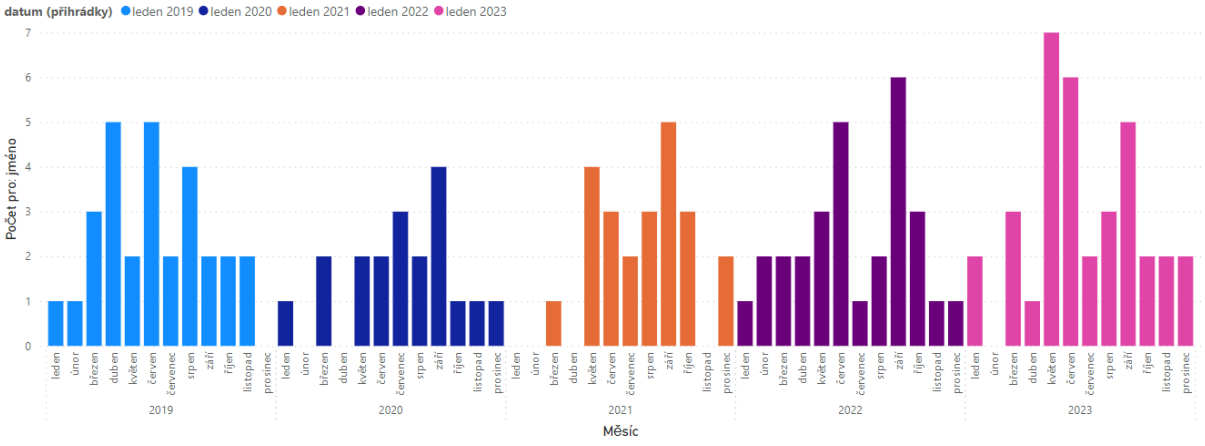
Existuje také druhý typ šifrovaček – internetové. Ty fungují na podobné úrovni, hráči se snaží vyluštit logické úkoly v co nejkratším čase. Chybí tam ale element výletu, jelikož online šifrovačky řeší účastníci z domova.

Poznatky k vývoji aplikace

Vzhledem k pohybu v přírodě se předpokládá, že účastníci nemají možnost své mobilní zařízení dobít pomocí elektrické zásuvky. Aplikace musí být tedy co nejméně náročná na výkon baterie. Pro extrémní případy, kdyby nebyl k dispozici ani signál nebo připojení k internetu, je potřeba, aby aplikace běžela kompletně offline.

2.1.1.3. Kdy se šifrovačky konají?

Akce se nekonají pravidelně, organizátoři si sami určují termíny jejich pořádání. Na obrázku 2.3 níže je vyznačený počet šifrovaček za každý měsíc.

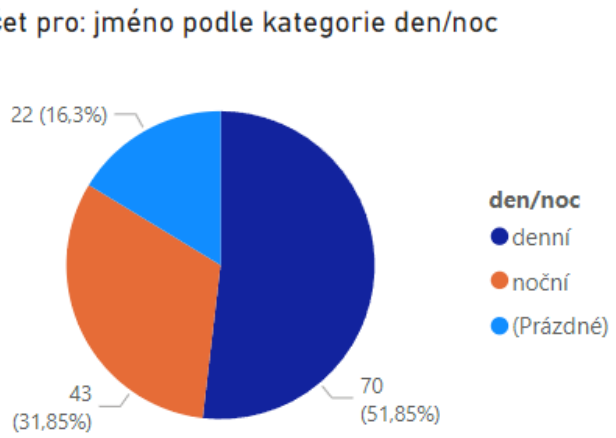


Obrázek 2.3: Graf počtu konaných soutěží. Zdroj: autorka práce

Komentář ke grafu 2.3 četnosti šifrovaček

Jarní a časně podzimní měsíce (období babího léta) jsou ideální roční dobou pro pořádání akcí, jelikož venkovní teploty dosahují nejpříjemnějších teplot a většina akcí je venku. Během prázdninových měsíců (červenec, srpen) je sice také teplo, ale pořádá se méně akcí. Řekla bych, že to je z důvodu právě letních prázdnin a faktu, že lidé raději jezdí na dovolené.

Díky příjemným teplotám na jaře jsou některé šifrovačky noční. Přidává to pocit dobrodružství a zpestřuje se tím celý zážitek. Na obrázku 2.4 níže je vidět poměr mezi denními a nočními akcemi.



Obrázek 2.4: Graf poměru denních a nočních šifrovaček. Zdroj: autorka práce

Komentář ke grafu 2.4 poměru denních a nočních šifrovaček

U některých akcí informace o denní době nebyla na stránkách uvedena, ale i přesto je poměr nočních šifrovaček značný – tvoří téměř třetinu všech konaných akcí.

Poznatky k vývoji aplikace

Z důvodu vysokého poměru denních i nočních akcí je vhodné přizpůsobit mobilní aplikaci pro oba časové režimy. Implementace denního a nočního režimu by zlepšila uživatelský komfort a usnadnila účastníkům používání aplikace za různých světelných podmínek. Denní režim by poskytoval optimální čitelnost a kontrast za denního světla, zatímco noční režim by minimalizoval oslnění a šetřil zrak účastníků během nočních šifrovaček.

2.1.2. Shrnutí pojmu „šifrovačka“

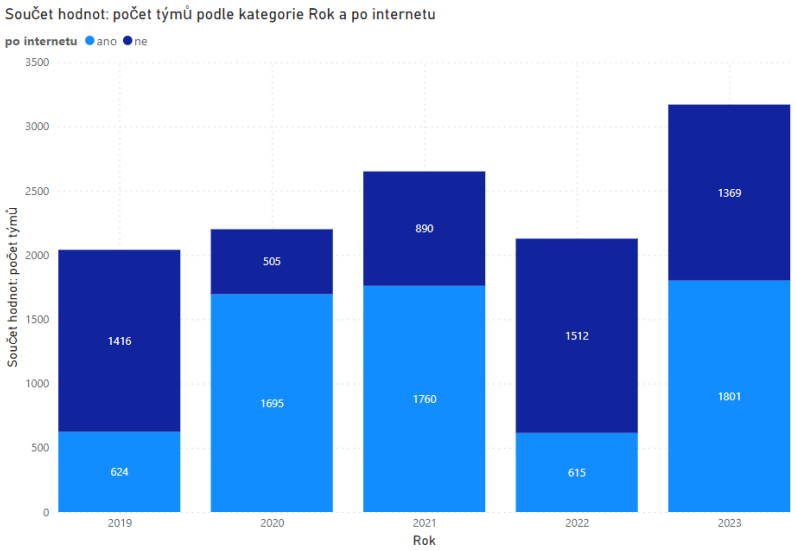
Šifrovačky představují unikátní typ logických soutěží, které kombinují rychlost, důvtip a logické uvažování. Tyto akce jsou zpravidla organizovány nadšenci bez jednotné asociace, což vede k jejich různorodosti, přestože mají společné rysy. Většinou se konají venku a mají podobu výletu rozděleného na několik úseků pomocí stanovišť se šiframi, existují ale i internetové verze, kde se nechodí ale pouze luští. Konají se po celý rok, převážně na jaře a během babího léta, a to během dne nebo noci.

2.2. Účastník logických her

Na akce se zapisují týmy, většinou maximálně 4 nebo 5členné, přesná pravidla si určují organizátoři sami. Přihlašuje se a platí startovné po týmech, účastníkem hry je každý člen týmu. Počet týmů je limitovaný, organizátoři musí myslet na to, aby účastníci moc neblokovali veřejný prostor. Takový problém nemají online šifrovačky, kam se může přihlásit mnohem více družstev (jediným problémem je kapacita serverů odevzdávacího portálu – viz *Tajenka a způsob odevzdání výsledků*).

Hráči jsou amatéři, to znamená, že se soutěžemi se neživí a dělají to dobrovolně. Proto se tolik nehlídají podvody, lidé se účastní pro radost a dobrý pocit. Švindlování by jim ani nic nepřineslo, nejsou žádné cenné nebo peněžité výhry.

Popularita šifrovaček stále roste, na obrázku 2.5 níže je patrný trend růstu zúčastněných týmů.



Obrázek 2.5: Graf počtu přihlášených týmů. Zdroj: autorka práce

Komentář ke grafu 2.5 počtu přihlášených týmů

Jak je z obrázku 2.5 zřejmé, popularita šifrovaček za posledních 5 let roste, protože kromě roku 2022 bylo vždy následující rok evidováno více přihlášených týmů než rok předchozí. Během roku 2019 je patrná menší účast na fyzických šifrovačkách kvůli koronavirovým opatřením a zákazu shromažďování se.

Online šifrovačky zaznamenávají rostoucí zájem díky vyšším kapacitám a možnosti účasti odkudkoliv. Tento komfort je pro mnoho hráčů klíčový, neboť umožňuje vyhnout se nutnosti cestovat na místo startu a vyměnit element dobrodružství za pohodlí domova.

Poznatky k vývoji aplikace

Jelikož se šifrovaček účastní čím dál tím více týmů, lze předpokládat, že chodí na šifrovačky i úplní nováčci, kteří s touto zájmovou aktivitou začínají. Je tedy důležité navrhnout intuitivní design aplikace, aby se v něm orientovali i lidé s nulovými zkušenostmi a poskytnout možnost nápovědy, jak aplikace funguje.

2.3. Šifra

Tato podkapitola se věnuje šifrám. Nejprve popisuje, co to šifra je v kontextu šifrovaček, poté se snaží definovat, pod jaký vědní obor kryptologie spadá, a nakonec popisuje a přibližuje fungování konkrétních šifer na šifrovačkách často používaných.

2.3.1. Charakteristika šifer

Jak již bylo zmíněno v kapitole *Logická hra „šifrovačka“*, úkolům během soutěže se přezdívá „šifra“. Tento pojem může být zavádějící, jelikož šifrování se v praxi používá pro utajování zpráv, které nemají být rozlousknuty. Naopak procesu zakrytí vyřešitelných zpráv se říká kódování. [1]

Šifra není zadána jednoznačně – pochopení kryptické zprávy je totiž součást procesu řešení. Není to tedy forma „otázka-odpověď“, řešitelé musí nejdřív z abstraktního seskupení obrázků či písmenek přijít, na co se autoři šifer ptají. Až pochopení principu můžou pomalu začít skládat tajenku.

K většině úloh jsou k dispozici nápovědy, které si můžou týmy vzít pro případy, že si neví se šifrou rady. Není doporučováno si nechat poradit hned po přečtení zadání, jelikož každá nápověda se rovná penalizaci. Způsob penalizace si určují organizátoři sami, například přidáním trestného času.

Všechny úlohy musí být unikátní, jinak by si účastníci mohli řešení pamatovat či vyhledat na internetu bez námahy. Některé postupy se přirozeně částečně opakují, takže lidé s většími zkušenostmi společné znaky snáze rozpoznají spíše než nováčci, jelikož každá dovednost se dá natrénovat. [2] Nikdy se žádné dvě šifry nebudou řešit úplně identicky.

2.3.2. Dosazení do kontextu kryptologie

Tato sekce se věnuje, jakým způsobem se šifry z šifrovaček dotýkají vědního oboru kryptologie.

Kryptologie je název vědy, která se zabývá šifrováním, šiframi a jejich luštěním, ať už chtěnými příjemci nebo narušiteli ze třetí strany. [3] Chránit informace a zprávy bylo žádoucí celé věky, nedá se tedy počátek tohoto oboru vymezit na přesnou dobu. Můžeme ale předpokládat, že se nějakým způsobem šifrovalo již od dob počátků písma. [4]

Pod termín kryptologie spadají tři další vědy – kryptografie, kryptoanalýza a steganografie, obory zaměřující se na odlišné aspekty šifrování.

2.3.2.1. Kryptografie

Kryptografie je disciplína zabývající se utajováním zpráv do podoby bez speciální znalosti nečitelné za účelem skrytí informací v původním, tzv. otevřeném textu, do textu šifrovaného. Provádí se to pomocí předem daných algoritmů, šifer. Je také potřeba znát klíč neboli informaci obsahující znalosti k de/šifrování. V moderní době je to klíčový obor pro bezpečnou online komunikaci, *jedná se o velice efektivní způsob, jak ochránit elektronická data před tím, aby byly nějak zneužity nepovolanými osobami*. [5]

2.3.2.2. Kryptoanalýza

Laicky se dá popsat jako opak kryptografie, místo utajování zpráv se zabývá jejich prolamováním. *Kryptoanalytici se neustále snaží objevit slabiny používaných kryptografických metod, které by mohly vést k jejich prolomení. Tím se testuje bezpečnost těchto metod.* [6] Nesmí se to ale plést s dešifrováním, při této operaci jsou již známé všechny potřebné informace (převážně klíč), jakým daný text dešifrovat. Kryptoanalytik žádné znalosti nemá a snaží se postup hádat, luštit. [7]

2.3.2.3. Steganografie

Trochu odlišnější disciplína, věnující se fyzickému skrývání zpráv. Metod je spousta, například neviditelný inkoust, digitální ukrývání zprávy do obrázku, či schování tajné zprávy do normální konverzace. Cílem steganografů je, aby nebyla existence tajných zpráv detekována nežádoucí třetí stranou. [8]

2.3.2.4. Kryptologie a šifrovačky

Šifrovačky kombinují všechny tři podobory kryptologie – kryptografii, kryptoanalýzu a steganografii – často i v rámci jediné šifry. Celý koncept soutěže je založen na kryptoanalýze, protože účastníci nedostávají žádné rady ohledně zadaných úkolů a musí je sami rozluštit. Zadání jsou vždy prezentována steganograficky, tedy skryta v různých formách, jako jsou obrázky, tabulky či jiné struktury.

I když se účastníci přímo nevěnují kryptografii ve smyslu šifrování zpráv, často se při řešení úloh setkávají s principy kryptografických metod. Lidský mozek není schopen rychle vyřešit složitější kryptografické úlohy bez technických pomůcek, a proto jsou šifrovačky navrženy tak, aby využívaly jednodušší techniky, které lze řešit bez rozsáhlého výpočetního výkonu. Také se občas během luštění soutěžních úkolů stane, že člověk narazí na skrytou nápovědu, která mu potřebné informace poví a úloha se tedy pak přelévá z kryptoanalytického úkolu na kryptografické dešifrování.

Na obrázku 2.6 na další straně je příklad zadání šifry z šifrovačky Kokořínský komár (2023).

Obrázek 2.6: Ukázka typické šifry na šifrovačkách. Zdroj: [9]

Tento úkol jasně spadá pod kryptoanalýzu – zadání je zdánlivě náhodné a bez jakýchkoli instrukcí. Také o steganografii se určitě jedná, nezasvěcená osoba by si nevšimla, že obrázek obsahuje skrytou tajnou zprávu. Na první pohled se může zdát, že tento úkol nespadá pod kryptografii, protože účastníkům nebyly poskytnuty žádné specifické informace pro dešifrování. Během řešení se však může objevit skrytá nápověda, která vnáší do řešení rovinu kryptografie.

Řešení: Šifra je sítí dvaatřicetistěnu, jehož trojúhelníkové stěny jsou výstražné značky a jeho pětiúhelníkové stěny obrázky dopravních situací. Všimneme si, že pro každý pětiúhelník jedna ze sousedních značek odpovídá obrázku a že na jedné ze značek je napsáno START. Sestavíme cestu přes všech dvanáct pětiúhelníkových stěn a jim odpovídajících značek tak, aby po značce vždy následovala stěna s odpovídající dopravní situací a aby značka, na kterou z pětiúhelníku odcházíme, byla správně orientovaná. Značky převedeme na čísla v jejich standardním číslování a tato následně na písmena, čímž dostaneme mezitajenku ODBOCKY PLUS M. Přečteme tedy zatím nepoužité značky v pořadí cesty, přičemž značku čteme, když se dostaneme na pětiúhelník, ke kterému přiléhá svou spodní stranou. Dostáváme text SENTZRAG. K textu přičteme, podle instrukcí mezitajenky, písmeno M a dostáváme řešení FRAGMENT.

Obrázek 2.7: Řešení předchozí šifry na obrázku 2.6. Zdroj: [10]

V tomto případě mezitajenka „ODBOCKY PLUS M“ napovídá řešiteli, že po získání textu „SENTZTRAG“ má přičíst ke každému písmenku hodnotu 13 (posun v abecedě). Tento posun s klíčem 13 představuje kryptografický prvek, protože luštitel obdrží informaci s klíčem potřebnou k dešifrování.

2.3.3. Časté šifry a postupy

Jak již bylo zmíněno, na šifrovačkách se často opakují určité způsoby finálního dešifrování znaků, i když samotné zadané úlohy bývají velmi odlišné. Tato podkapitola se zaměřuje na vysvětlení principů často používaných metod dešifrování, kterým se také říká „šifra“. Mezi nejběžnější patří přepis čísel na písmena abecedy, Morseova abeceda, Braillovo písmo, binární soustava, semaforová abeceda a velký polský kříž.

2.3.3.1. Abeceda

Nejjednodušším způsobem, jak zamaskovat zprávu, je převedení písmenek zprávy na čísla podle pořadí v abecedě. Standartně se na soutěžích používá anglická abeceda (česká bez diakritiky a „CH“) indexovaná od 1, neboli hodnoty 1-26. Tento jednoduchý převod se často kombinuje s dalšími šiframi.

V seznamu níže jsou vypsána písmena a jejich náležité hodnoty:

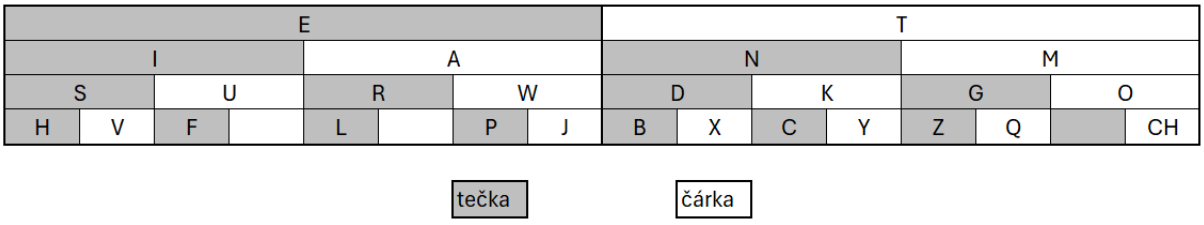
A → 1	B → 2	C → 3	D → 4	E → 5	F → 6	G → 7	H → 8	I → 9
J → 10	K → 11	L → 12	M → 13	N → 14	O → 15	P → 16	Q → 17	R → 18
S → 19	T → 20	U → 21	V → 22	W → 23	X → 24	Y → 25	Z → 26	

2.3.3.2. Morseova abeceda

Abeceda na principu dlouhého a krátkého signálu, kdy kombinace o délce 1-4 znaků těchto dvou druhů reprezentuje písmeno v abecedě. Vícemístné kódy reprezentují číslice, interpunkci a další jazykové prostředky, které se ale v šifrovačkách nevyskytují.

Morseova abeceda se v psané formě reprezentuje tečkou (.) a čárkou (-), které nejlépe reprezentují délku analogového signálu. Během šifrovaček se ale luštitel na tuto psanou konvenci spoléhat nesmí, může nastat situace, kde místo teček a čárek se musí dešifrovat jiné dva symboly. Není předem jasné, který z nich představuje krátký a dlouhý signál, proto musí luštitelé dávat pozor na dvojici opakujících se jevů maximálně po čtveřicích.

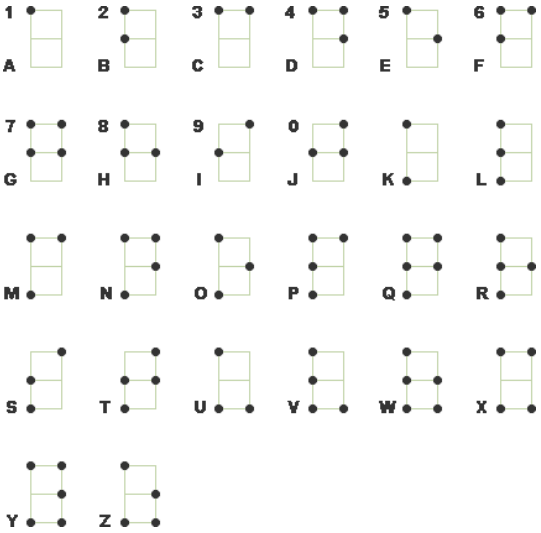
Na obrázku 2.8 níže jsou vypsány reprezentace všech písmenek české abecedy bez diakritiky. Od kořenových písmenek E (.) a T (-) se rozvíjí 2 paralelní stromy, na které se v každém patře přidává do řetězce na konec další znak. Šedivé bloky přidávají tečku a bílé bloky přidávají čárku.



Obrázek 2.8: Rozpis Morseové abecedy. Zdroj: autorka práce.

2.3.3.3. Braillovo písmo

Braillovo bodové písmo představuje fenomén ve způsobech čtení a psaní nevidomých osob a je také již neoddělitelnou součástí jejich kultury. [11] Jedná se body umístěné v tabulce 3x2 (řádek x sloupec), kde každý bod má dva stavy. Pichtův psací stroj, nejrozšířenější nástroj pro nevidomé mění tyto stavy způsobem zvrásnění stránek pro lepší čtení hmatem. Tento způsob psaní Braillova písma se na šifrovačkách nevyskytuje, jelikož by bylo rozklíčování šifry moc lehké. Podobně jako u morseovky se body schovávají za různé obrázky, tentokrát uspořádané do šestice ve specifickém rozložení tabulky 3x2. Na obrázku 2.9 níže je rozepsaná reprezentace abecedy v Braillově písmu a na obrázku 2.10 je tato reprezentace psaná v tabulkové formě.



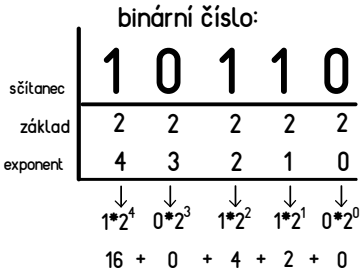
Obrázek 2.9: Braillovo písmo. Zdroj: [12]

	□	'					|
A	C	E	D	á	č	š	ď
,	I	:	J	?	ó	+	ř
B	F	H	G	ě	ň	ť	/
.	í	*	é	-	ú	)	
K	M	O	N	U	X	Z	Y
;	S	!	T	(	ž	"	ů
L	P	R	Q	V	ý	W	

Obrázek 2.10: Tabulka Braillova písma. Zdroj: [13]

2.3.3.4. Binární soustava

Dvojková soustava (binární) je poziční číselná soustava o základu z = 2. Číselný zápis využívá tyto symboly: 0, 1. [14] Základ 2 je mocněn exponentem, který je daný pozicí v řetězci jmenovaných nul a jedniček. Nejmenší exponent 0 je na konci řetězce a největší (záleží na délce řetězce) na začátku. Samotné číslice se násobí s umocněným základem. Součet všech vynásobených hodnot tvoří výsledek. Tato šifra nedává jako výsledek písmenko, proto se vždy kombinuje s další šifrou (převážně *abecedou*). Na obrázku 2.11 níže je graficky znázorněn princip binární soustavy.

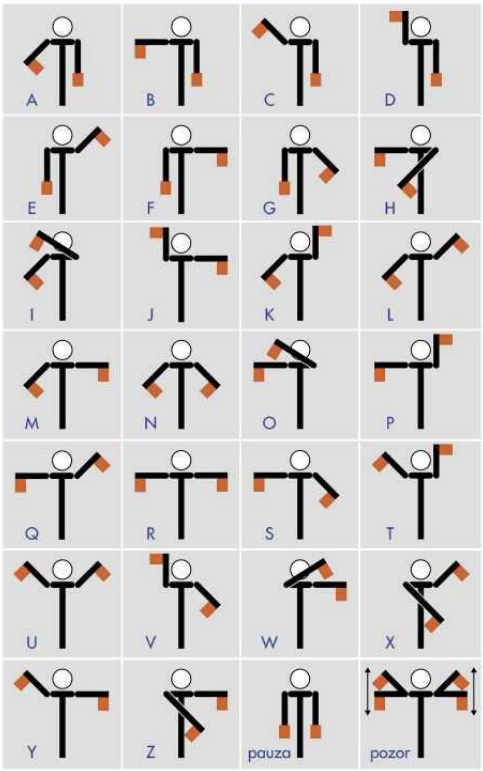


Obrázek 2.11: Vysvětlení binární soustavy. Zdroj: autorka práce.

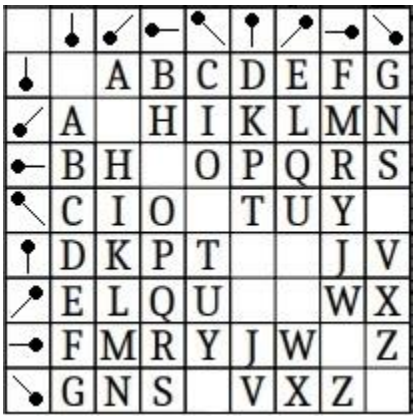
2.3.3.5. Semaforová abeceda

Semafor je dohodnutá skupina znaků vysílaných dvěma vlajkami jednou osobu, kdy jednomu znaku přísluší jeden signál. [15] Také se jí přezdívá praporková abeceda a byla primárně používána v námořní dopravě, což se může lehce splést s námořní vlajkovou abecedou[2], proto je vhodnější používat pojmenování semaforová neboli „semafor“.

V každé ruce má vysílající osoba jednu vlajku, která je umístěná na jedné z 8 poloh, které tvoří pomyslný kruh kolem trupu a hlavy. Pomocí kombinace umístění obou vlajek se dá komunikovat s další stranou díky domluveným překladům mezi pozicemi vlajek a písmenky. Na obrázku 2.12 níže je zobrazen seznam těchto překladů a na vedlejším obrázku 2.13 jsou překlady v kompaktnější tabulkové podobě.



Obrázek 2.12: Semaforová abeceda. Zdroj:[16]



		A	B	C	D	E	F	G
A		H	I	K	L	M	N	
B	H		O	P	Q	R	S	
C	I	O		T	U	Y		
D	K	P	T			J	V	
E	L	Q	U			W	X	
F	M	R	Y	J	W		Z	
G	N	S		V	X	Z		

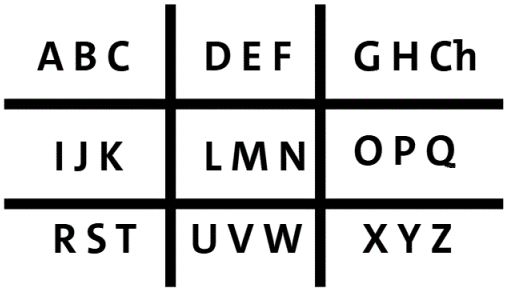
Obrázek 2.13: Tabulka semaforové abecedy. Zdroj: autorka práce.

[2] Podobu námořní vlajkové abecedy lze vidět webu http://www.nekula.cz/abeceda.html

2.3.3.6. Polský kříž

Jedná se o šifru mající dvě podoby. První verze se nazývá velký polský kříž a druhá malý polský kříž[3]. Obě jsou tabulkové, ale velkému stačí pouze jedna tabulka, je tedy mnohem populárnější a používanější než malý, který vyžaduje čtyři tabulky. V této práci se slovní spojení „polský kříž“ bude od této chvíle odkazovat pouze na velký.

Do tabulky 3x3 se postupně vypíše 27místná abeceda (česká bez diakritiky s CH) po řádcích, přičemž každá buňka obsahuje 3 písmena. Je důležité, aby tabulka měla výrazné vnitřní ohraničení buněk, ale venku ohraničená nebyla. Podle kombinace ohraničení + pozice v buňce se dá jednoduše zjistit, o jaké písmenko se jedná. Na obrázku 2.14 níže je předvedené rozložení tabulky a písmen uvnitř.



Obrázek 2.14: Velký polský kříž. Zdroj: [17]

2.3.4. Shrnutí pojmu „šifra“

Šifra v kontextu šifrovaček označuje logické úkoly, které nejsou zadány stylem otázka-odpověď a účastníci musí nejprve vymyslet princip úlohy a poté ji vyřešit. Každé zadání úlohy je unikátní, ale procesy finálního dešifrování se často opakují. Těmto procesům se také říká „šifra“. Mezi nejčastější šifry spadá převod písmen na čísla, Morseova abeceda, Braillovo písmo, binární soustava, semaforová abeceda a polský kříž.

[3] Podobu malého polského kříže lze vidět webu https://sifry.sourceforge.net/sff_kriz_maly.html

2.4. Tajenka a způsob odevzdání výsledků

Tajenka šifer mívá různou podobu, kterou si určují sami organizátoři Ze sesbíraných zkušeností to většinou bývají buď podstatná jména v prvním pádě nebo slovní spojení napovídající lokaci další šifry.

V prvním případě se řešení vkládá do odevzdávacího systému, který si každý organizátor vytváří a spravuje sám. Neexistuje žádný unifikovaný systém, který by sjednocoval všechny šifrovačky. Po zadání správné tajenky se hráči zobrazí poloha dalšího stanoviště. Ve stejném systému je také možnost brát nápovědy k právě řešeným šifrám, která vede k penalizaci.

V druhém případě se tajenka nikam nezadává a hráči musí na mapě najít místo odpovídající jejich řešení. Tento způsob je používán převážně na akcích, které se odehrávají na místech bez internetového signálu. Všechny nápovědy v šifrovačkách bez mobilního připojení obdrží účastníci na startu v zapečetěných obálkách a jejich využití se vyhodnocuje v cíli.

Bohužel není možnost ukázat konkrétní odevzdávací systém, jelikož většinou bývají otevřené pouze během hry.

Poznatky k vývoji aplikace

Jelikož většina šifrovaček svůj systém má, bylo by vhodné mít možnost zkopírovat řešení přímo z aplikace do odevzdávacího formuláře.

2.5. Shrnutí kapitoly definic

V této kapitole byl sepsán teoretický základ o tom, co to šifrovačky jsou a jak fungují. Pomocí analýzy 135 her a osobních zkušeností se vytyčily společné znaky všech těchto soutěží. Také se vysvětlily důležité pojmy spojené se šifrovačkami, které pomohly dokreslit představu o akcích. Tento obrázek byl klíčový pro vývoj aplikace.

3. Podpora účastníků

V této kapitole se rozebírají způsoby, jakým zjednodušit luštitelům proces řešení šifry. Výstupem není nalézt způsob, který celé zadání vyřeší sám, soutěže by tím totiž ztratily pointu, ale pomoci luštitelům s finální fází dešifrování tajenky. Tento manuální krok je pomalý a velice náchylný k chybám z nepozornosti.

3.1. Etický způsob podpory

Před hledáním řešení je vhodné se zamyslet, zda je pomůcka podporující účastníky etická vůči ostatním. K luštění šifer si soutěžící mohou přinést cokoliv, co by jim pomohlo vyřešit úkoly. Bylo by totiž náročné si zapamatovat všechny možné druhy šifer, svátků a dalších všeobecných znalostí. Soutěže jsou o uvažování, logickém myšlení, a ne mechanické paměti, proto jsou všechny pomocné materiály povolené (včetně používání internetu). Problém tohoto přístupu je, že pátrání po různých webových stránkách zabírá hodně času.

3.2. Centralizace pomocných materiálů

Pro urychlení práce je vhodné soustředit pomůcky k často používaným úlohám na jedno místo. Pokusy o zpříjemnění manuálního hledání předem vymyšleného řešení již v minulosti byly. V sekcích níže jsou přiblíženy papírové tabulky a již existující mobilní aplikace.

3.2.1. Papírové tabulky

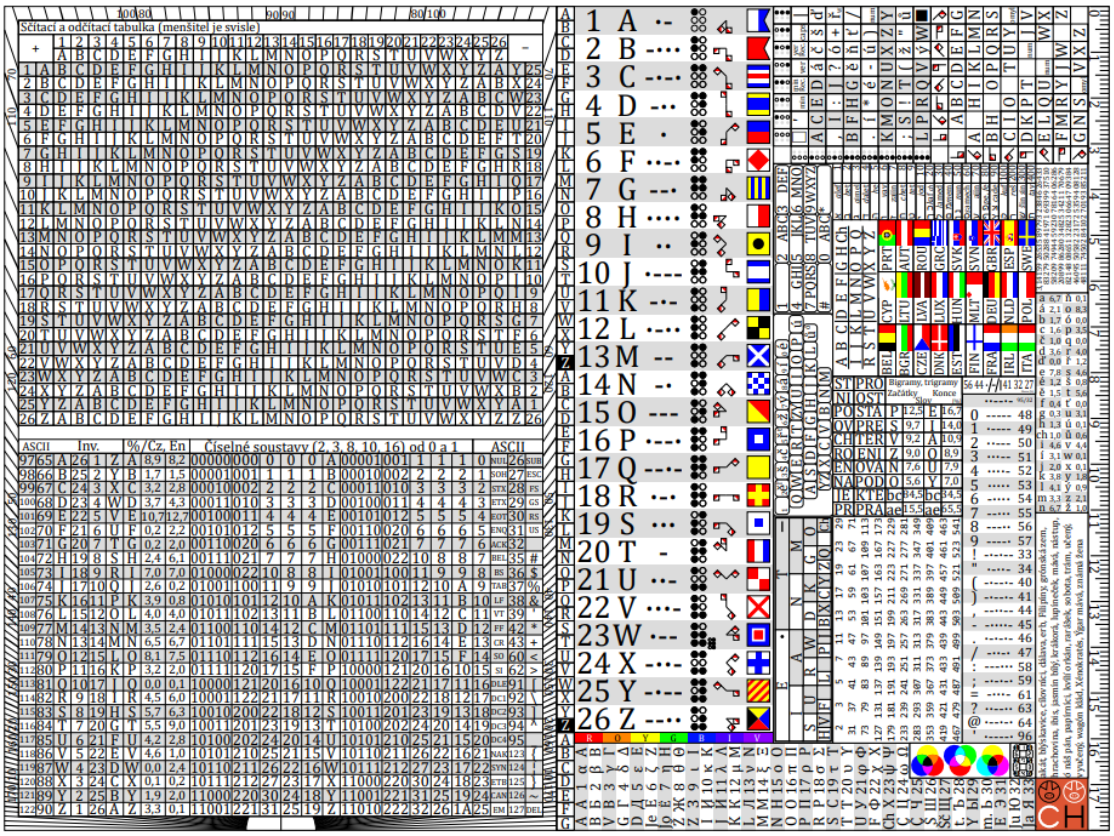
Často používané abecedy (například Morseova nebo semaforová) byly seskládány skupinou *Chlýftým*[4] do kompaktní formy tabulek, které se vejdou na jeden papír formátu A4. Tato forma je podporována většinou organizátorů, která tabulky poskytuje přihlášeným k dispozici. Není to jediný pokus o vytvoření šikovného pomocníka, existuje i druhá méně populární kolekce od týmu *NaPALM*[5].

Bohužel to však není ideální řešení, níže je sepsaný výčtový seznam výhod a nevýhod:

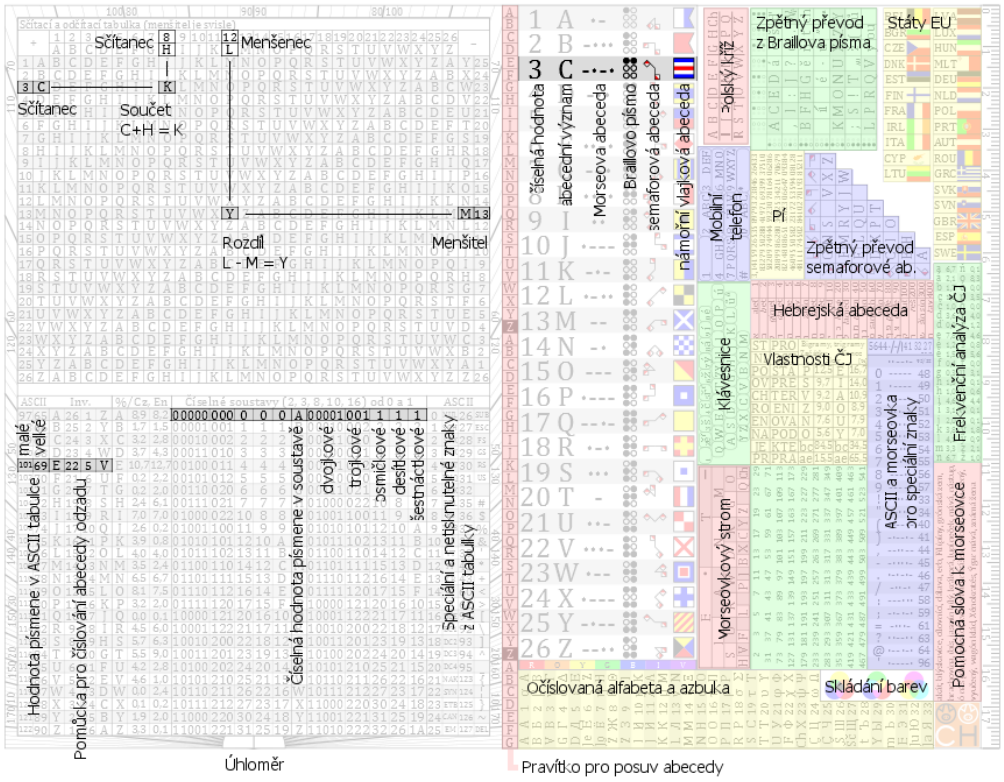
Výhody	**Nevýhody**
• Kompaktnost informací	• Nepřehlednost
• Nevybije se	• Lehce se zničí
• Lehce dostupné pro všechny	• V noci bez baterky nejde použít
	• Manuální hledání

Na obrázku 3.1 níže jsou ukázány tabulky týmu *Chlýftým* i s vysvětlivkami na obrázku 3.2 k jednotlivým blokům.

[4] Tým organizující šifrovačky *TMOU* a tvůrci nejpopulárnějších šifrovacích tabulek. https://chlyftym.cz
[5] Šifrovací pomůcky týmu *NaPalm*: https://www.napalmne.cz/pomucky/NaPALM_pravitko221015.pdf



Obrázek 3.1: Šifrovací pomůcky týmu Chlýftým. Zdroj: [18]



Obrázek 3.2: Vysvětlení k šifrovací pomůckám týmu Chlýftým. Zdroj: [19]

3.2.2. Mobilní aplikace *Absolutno*

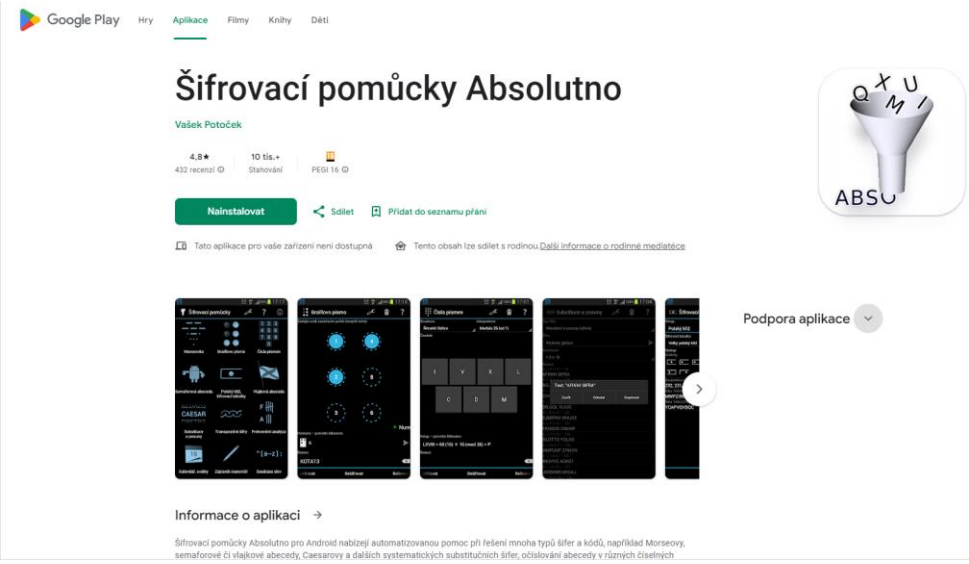
Šifrovací pomůcky Absolutno[6] je aplikace sloužící k řešení různých druhů šifer. V rychlosti soutěže je zadávání vstupu do aplikace mnohem jistější, než hledání na nepřehledném papíře (viz papírové pomůcky). Bohužel je však aplikace pouze pro zařízení s operačním systémem Android, takže nebyla k dispozici všem. Zároveň je již nedostupná, jelikož ji obchod s mobilními aplikacemi Google Play nedovoluje stáhnout. Funguje spolehlivě a rychle, avšak design by mohl být intuitivnější a přehlednější. Na obrázku 3.3 níže je screenshot stránky s aplikací v obchodě.

Výhody

- Rychlost řešení
- Množství šifer a dekodérů
- Malý prostor pro chyby z nepozornosti

Nevýhody

- Nepřehlednost aplikace
- Pouze noční režim
- Pouze pro systém Android
- Nedostupná



Obrázek 3.3: Screenshot Google Play stránky aplikace *Absolutno*. Zdroj: [20]

3.3. Vyhodnocení aktuálních pomůcek

Účastníkům her lze luštění zjednodušit zrychlením manuálního hledání překladů „symbol-písmenko“. Spojení materiálů do kompaktní podoby je efektivní metoda, jak příjemnějšího zážitku docílit. V tuto chvíli existují dvě formy ucelených pomůcek, ale bohužel ani jedna není ideální. Manuální hledání v papírových tabulkách je zdlouhavé a náchylné k chybovosti a mobilní aplikace *Absolutno* není pro nováčky intuitivní a také není možné ji aktuálně nově získat.

Vyvinout novou aplikaci, která bude vycházet z obou šifrovacích pomůcek smysl má, jelikož by mohla eliminovat nedostatky jak papírových tabulek, tak mobilní aplikace *Absolutno*. Nová aplikace by měla být dostupná pro různé operační systémy, nejen pro Android, a měla by mít intuitivní a přehledné uživatelské rozhraní.

[6] Odkaz na Google Play: https://play.google.com/store/apps/details?id=cz.absolutno.sifry&hl=cs

4. Návrh aplikace *Šifrovací pomůcky*

Tato kapitola se věnuje návrhu samotné aplikace. Je rozdělená na dvě velké podsekce zabývající se odlišnými aspekty návrhu. První část se věnuje funkčnímu návrhu, kde jsou podrobně popsány funkcionality a chování aplikace a druhá se zaměřuje na grafický design a náčrty vizuálu projektu.

4.1. Funkční návrh

Tato sekce se zaměřuje na detailní popis funkcionalit aplikace. Prostřednictvím funkčních a nefunkčních požadavků se definuje, jaké má mít aplikace schopnosti a vlastnosti. Dále je chování aplikace vůči uživateli ilustrováno pomocí diagramu případů užití a procesního modelu. Na závěr je struktura projektu vymodelována pomocí diagramu komponent a tříd, přičemž chování uvnitř aplikace je popsáno prostřednictvím sekvenčních diagramů.

4.1.1. Funkční požadavky

Definice pojmu požadavek není jednoznačná, každý si pod tímto pojmem dokáže představit svoje, pokusy o unifikování definic historicky už byly, například tyto tři body popisující požadavky:

1. Podmínku nebo funkci, kterou uživatel potřebuje pro řešení problému nebo dosažení nějakého cíle.
2. Podmínku nebo funkci, kterou musí systém nebo jeho část splňovat, aby vyhověl smlouvě, standardu, specifikaci nebo jinému dokumentu, jenž se na něj formálně vztahuje.
3. Dokumentovanou podobu některého z předchozích dvou bodů. [21]

Laicky řečeno, požadavek je chování systému, které je chtěné a vyžadované. Cílem aplikace je správnost, intuitivnost a přívětivost. Detailní rozpis všech jejích funkčních požadavků:

- **FR01 – Výběr dekodéru** – Uživatel si bude moct vybrat jeden ze 6 dekodérů podle potřeby.
- **FR02 – Zobrazit nápovědu** – Uživatel si bude moct zobrazit nápovědu ke každé šifře o jejím principu a způsobu zadávání vstupu k dešifrování.
- **FR03 – Zadání vstupu** – Uživatel bude moct zadat vstup k dešifrování.
- **FR04 – Smazání posledního znaku vstupu** – Uživatel bude moct smazat poslední zadaný znak vstupu.
- **FR05 – Smazání vstupu** – Uživatel bude moct smazat vstup.
- **FR06 – Uložení výsledku** – Uživatel bude moct uložit výsledek do aktuálního řetězce.
- **FR07** – **Smazání posledního uloženého vstupu** – Uživatel bude moct smazat poslední uložené řešení.
- **FR08 – Zkopírování výsledků** – Uživatel bude moct zkopírovat řešení i uložené řešení.
- **FR09 – Změna barevného režimu** – Uživatel bude moct manuálně změnit barvy na noční a denní režim.

4.1.2. Nefunkční požadavky

Nefunkční požadavky rozšiřují vlastnosti systému, a to podmínkami a vlastnostmi, které by měl systém dodržet. Příkladem jsou požadavky na výkon, bezpečnost, spolehlivost a vizuální stránku. [22]

Pro zadanou aplikaci je důležitá vizuální přívětivost na spoustě zařízeních s různými velikostmi obrazovek, tj. konzistence v uživatelském rozhraní.

- **NFRQ01 – Stejné měřítko** – Uživatelské rozhraní by mělo mít přívětivé měřítko pro každou obrazovku na různých zařízeních.
- **NFRQ02 – Změna barevného režimu** – Tmavý a světlý barevný režim se bude automaticky nastavovat podle režimu zařízení.

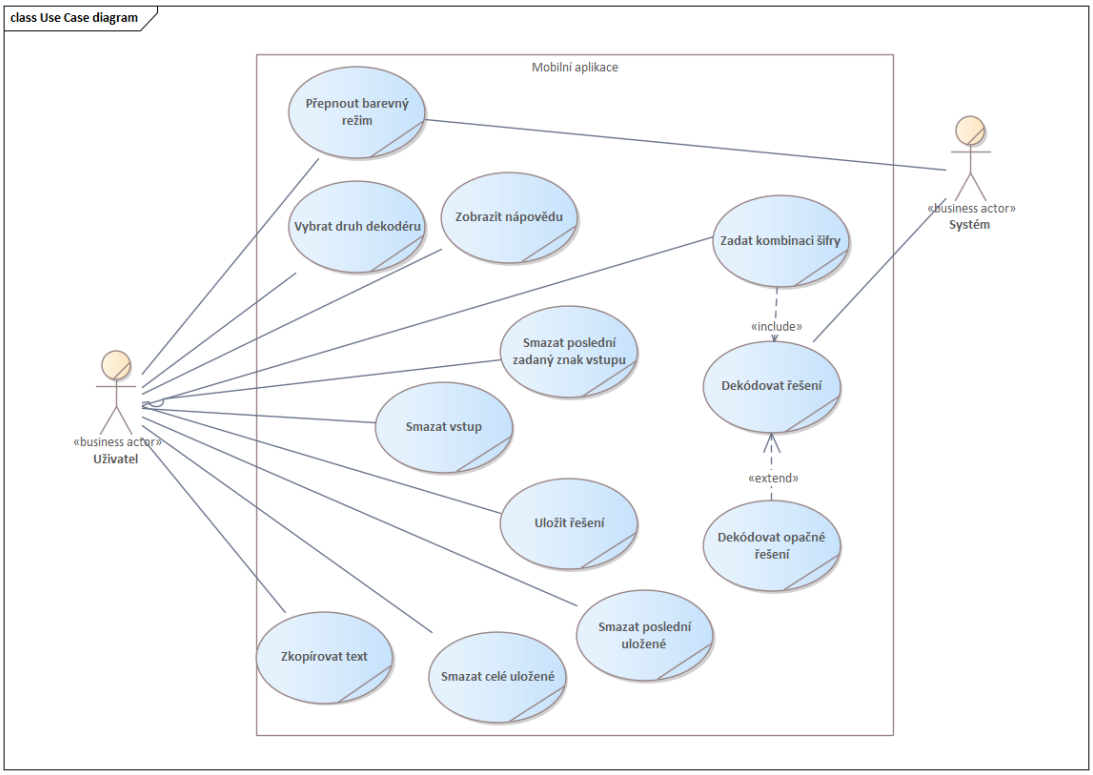
4.1.3. Diagram případu užití

Případ užití (anglicky use case) je činnost, kterou vykonává systém. Tato činnost je iniciována aktérem (přidělená role převážně osobám interagujícím se systémem, jemu samotnému nebo času) a je popisována z aktérova pohledu v infinitivu. [23]

V aplikaci *Šifrovací nástroje* figurují dva aktéři:

- **uživatel** – každý, kdo ji bude používat na jakémkoliv zařízení
- **systém** – má na starosti automatické procesy

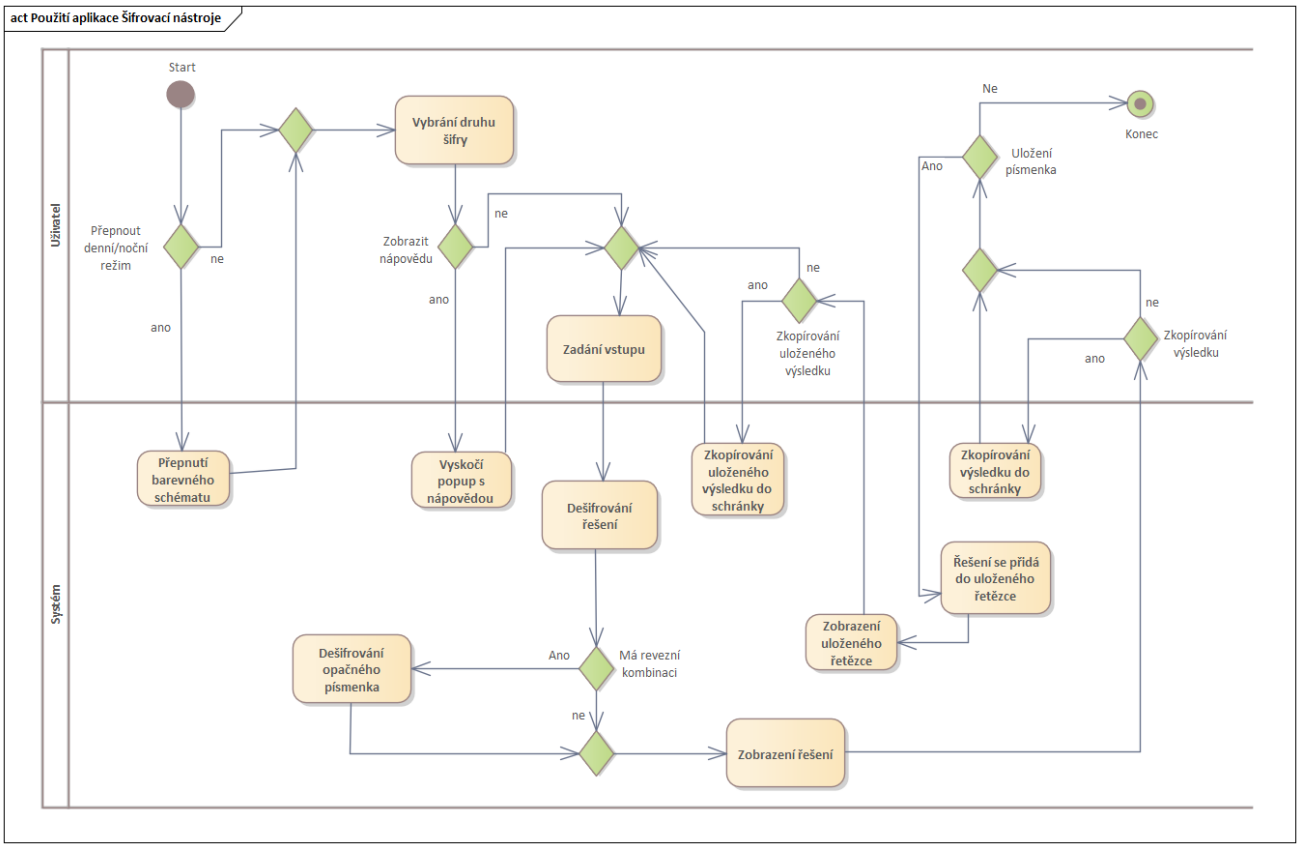
Na obrázku 4.1 níže jsou vyobrazeny případy užití aplikace.



Obrázek 4.1: Diagram případu užití. Zdroj: autorka práce.

4.1.4. Procesní diagram

Procesní diagram, také nazývaný „diagram aktivit“ je grafické znázornění jednotlivých kroků a toku činností v rámci nějakého procesu. [24] V tomto projektu se jedná o sled kroků, které se dějí při používání aplikace. Existuje více způsobů vykreslování, pro diagram níže byla zvolena notace BPMN[7]. Na obrázku 4.2 níže je vyobrazen proces používání aplikace *Šifrovací nástroje*.



Obrázek 4.2: Procesní diagram. Zdroj: autorka práce.

- Aktivitu začíná uživatel při otevření aplikace. Zakončuje ji vypnutím aplikace.
- Systém reaguje na podněty od uživatele a vykonává příslušné procesy.

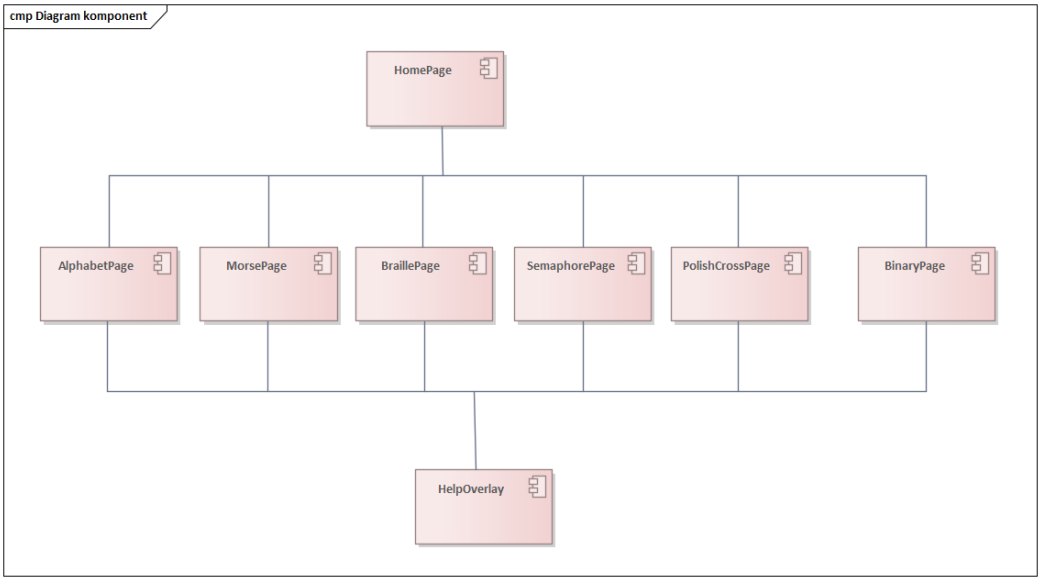
[7] Pro více informací navštivte: https://www.microsoft.com/cs-cz/microsoft-365/visio/business-process-modeling-notation

4.1.5. Diagram komponent

Diagram komponent znázorňuje vztahy mezi složkami systému, jak vypadají a jak mezi sebou komunikují. Určuje pravidla, která složky musí dodržovat k fungující spolupráci. [25]

Diagram komponent je pro představení této aplikace nejjednodušší, proto byl po dohodě s vedoucím vybrán jako způsob vizualizace bez znázornění komunikace.

Na obrázku 4.3 níže je vyobrazen vztah komponent aplikace.



Obrázek 4.3: Diagram komponent. Zdroj: autorka práce.

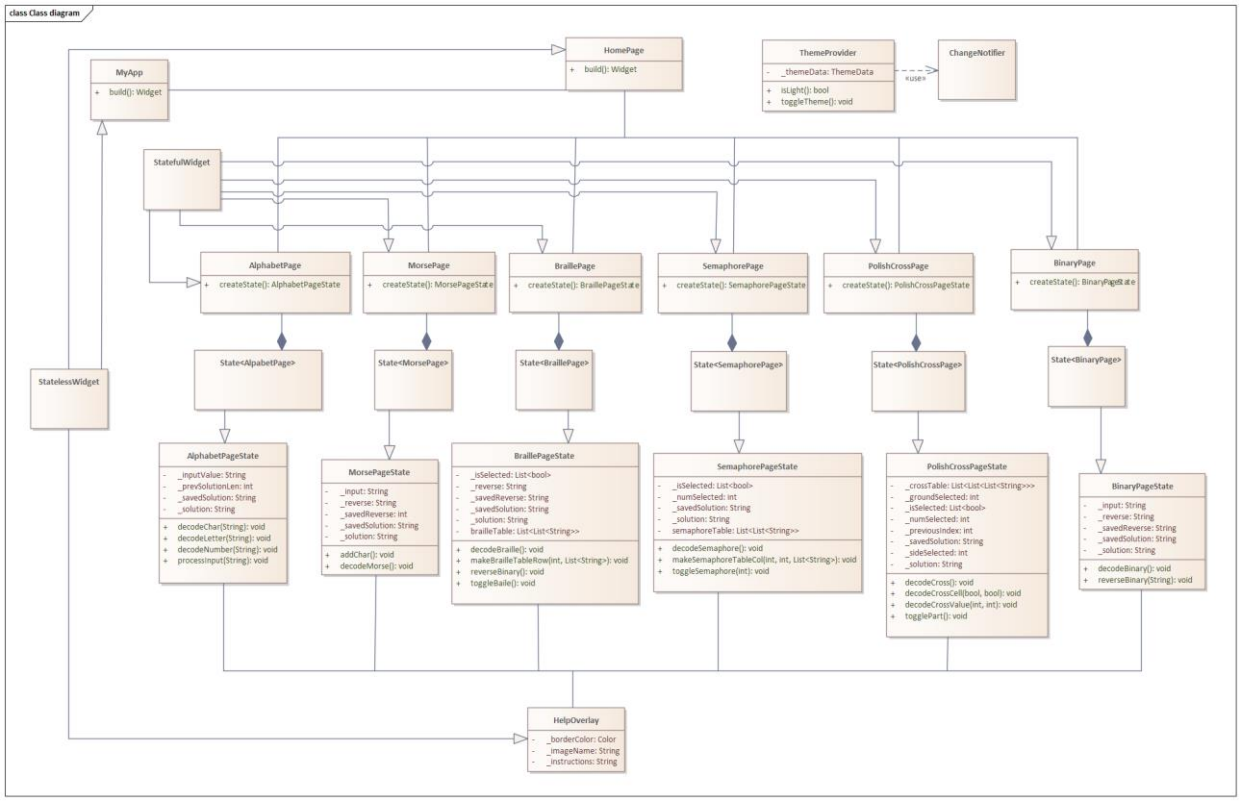
- Aplikace *Šifrovací pomůcky* se skládá z 8 komponent, kde 6 z nich reprezentují stránku pro vybraných 6 dekodérů (**AlphabetPage**, **MorsePage**, **BraillePage**, **SemaphorePage**, **PolishCrossPage**, **BinaryPage**). Na každou z nich jsou napojeny komponenty dvě – jedna pro hlavní stránku (**HomePage**), která slouží jako rozcestník, ze kterého si uživatel vybírá požadovaný dekodér a druhá pro stránku s nápovědou (**HelpOverlay**). Nápověda se nabízí ke každé šifře vlastní.

4.1.6. Diagram tříd

Zobrazují strukturu systému pomocí tříd a jejich společných vztahů. Případně každá třída může nezávazně na ostatních zobrazovat vybrané atributy a operace, které poskytují další náhled a informace o struktuře. [26]

Diagram tříd *Šifrovacích nástrojů* je na první pohled méně přehledný, proto slouží jako detailnější náhled fungování každé komponenty, a ne jako představení celé aplikace. Konkrétní popsání fungování metod bude v sekci *Implementace*.

Na obrázku 4.4 níže je vyobrazen diagram tříd aplikace.



Obrázek 4.4: Diagram tříd. Zdroj: autorka práce.

Vezměme si například komponentu *AlphabetPage*, která se skládá ze tří tříd:

- **AlphabetPage** – třída reprezentující stránku s abecedovým dekodérem, generalizuje tzv. *StatefulWidget*, nestatickou součástku poskytovanou frameworkem Flutter jazyka Dart.
- **State<AlphabetPage>** - označení generického typu, který reprezentuje stav stránky AlphabetPage.
- **AlphabetPageState** – tzv. *State třída* starající se o změny stavu třídy AlphabetPage, které se pak projevují na obrazovce. Tato třída obsahuje logiku pro překreslování widgetu (prvek stránky) v závislosti na změnách stavu.

Ostatní komponenty dekodérů jsou složeny ekvivalentně.

Třídy **HomePage** a **HelpOverlay** jsou oproti tomu generalizované tzv. *StatelessWidgety* (statická součástka Flutteru), jelikož to nejsou dynamické stránky a během běhu aplikace se nebudou nikdy měnit. Proto se u nich nemusí hlídat ani ovládat žádný stav.

Třída **ThemeProvider** starající se o změnu barevného schématu a přepínání mezi denním a nočním režimem používá **ChangeNotifier**, což je základní nástroj pro informování posluchačů o změnách stavu.

4.1.7. Sekvenční diagramy

Typ diagramu zachycující vzájemnou spolupráci mezi objekty a jejich společnou komunikaci. Je to nejpoužívanější typ interakčních diagramů díky své přehlednosti a návaznosti posílaných zpráv. Zprávy mezi položkami můžou být různých typů, ať už volání funkcí, předávání proměnných, či popis činnosti. Komunikace se značí oboustranně, tedy jak žádosti, tak odpovědi. Nejčastěji se popisují jeden či více případu užití v jednoho diagramu. [23]

Základními prvky sekvenčního diagramu:

- **Čára života** (lifeline) – reprezentuje existenci objektu (aktéra, třídy, komponenty apod.) v průběhu času
- **Zpráva** – komunikace mezi dvěma čárami života (speciální případ je, když čára volá sebe sama)
- **Fragment** – oblast se specifickým chováním (např. **alt** – provede se pouze při splnění určité podmínky nebo **loop** – cycklus, využívá se při opakované činnosti)

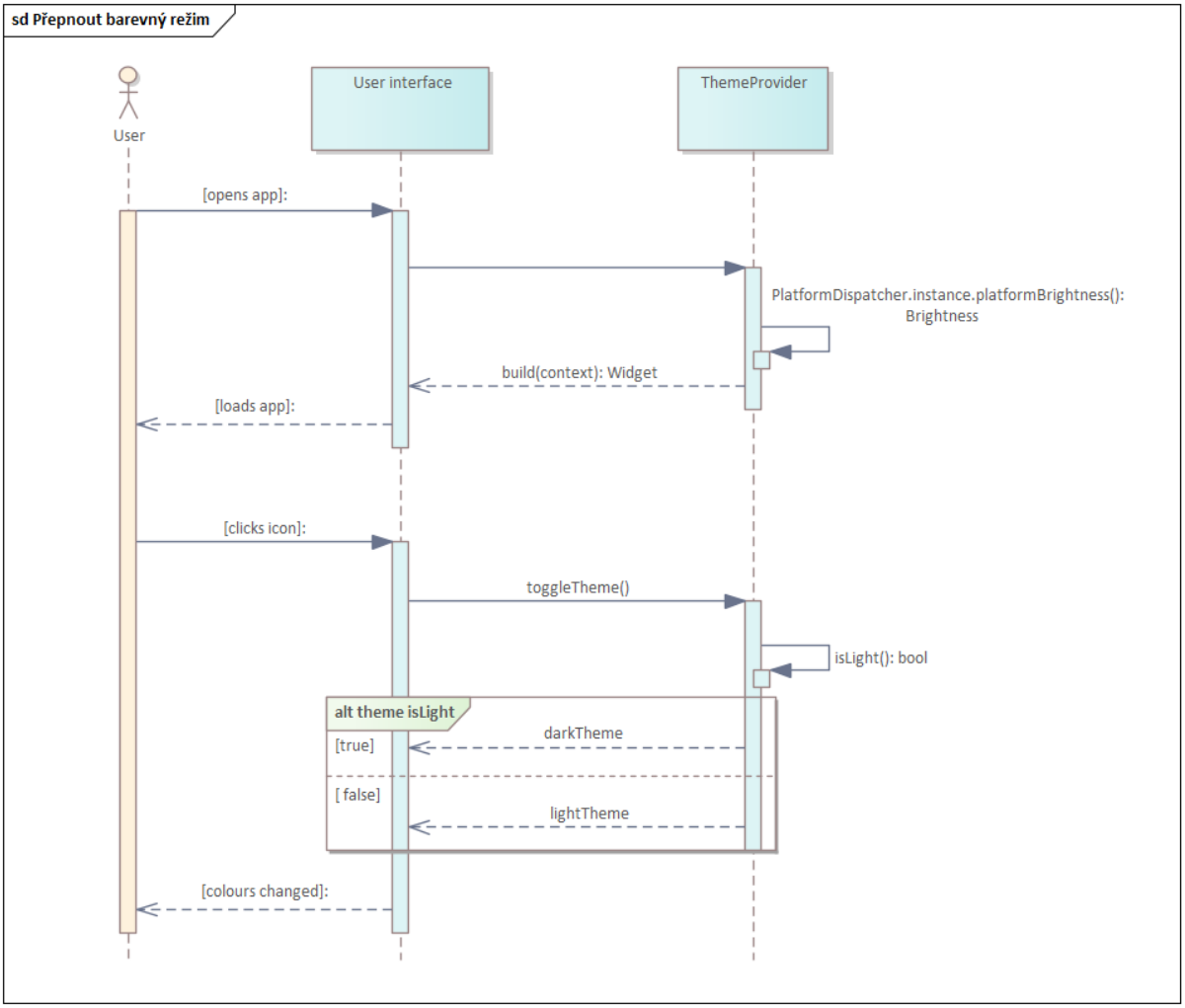
Po konzultaci s vedoucím jsem se rozhodla nedělat diagramy všech use-casů, jelikož spousta z nich se opakuje. Zároveň nejsou často moc složité a pro čtenáře zajímavé, proto jsem některé spojila dohromady.

Pár poznámek k notaci komunikace:

- Plná šipka doprava značí příkaz, přerušovaná doleva značí změněnou proměnou (odpověď)
- popisek v hranatých závorkách [ ] je slovy popsaná činnost, ne jména metod/proměnných
- Volané metody mají za jménem kulaté závorky ( ) a proměnné jsou bez značek

4.1.7.1. Přepnout barevný režim

Na obrázku 4.5 níže je vyobrazen sekvenční diagram procesu přepínání barevného režimu.

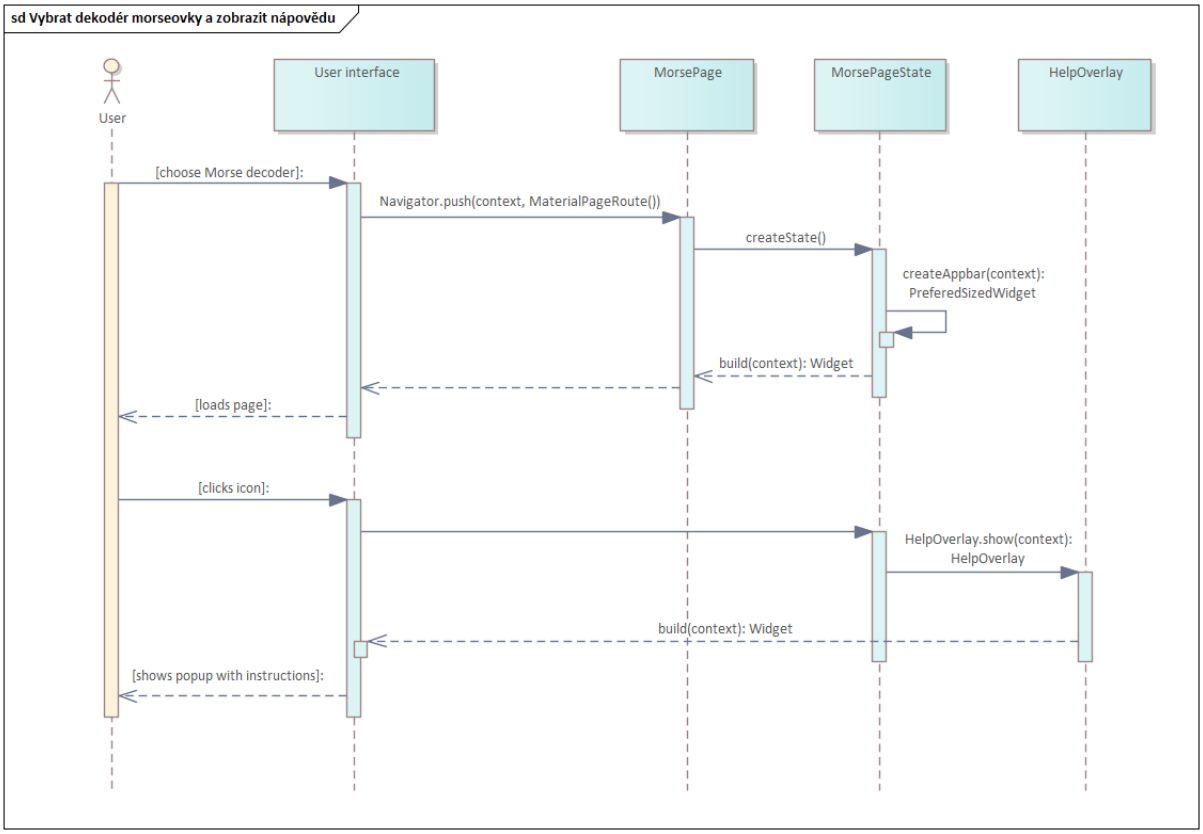


Obrázek 4.5: Sekvenční diagram přepínání nočního/denního režimu. Zdroj: autorka práce.

- Po zapnutí aplikace se denní nebo noční režim automaticky nastaví podle režimu zařízení.
- Uživatel má pak možnost manuálně měnit režim pomocí tlačítka, které spustí metodu **toggleTheme()**, která změní téma na neaktuální.

4.1.7.2. Vybrat dekodér morseovky a zobrazit nápovědu

Na obrázku 4.6 níže je vyobrazen sekvenční diagram procesů vybírání dekodéru morseovky a zobrazení nápovědy.

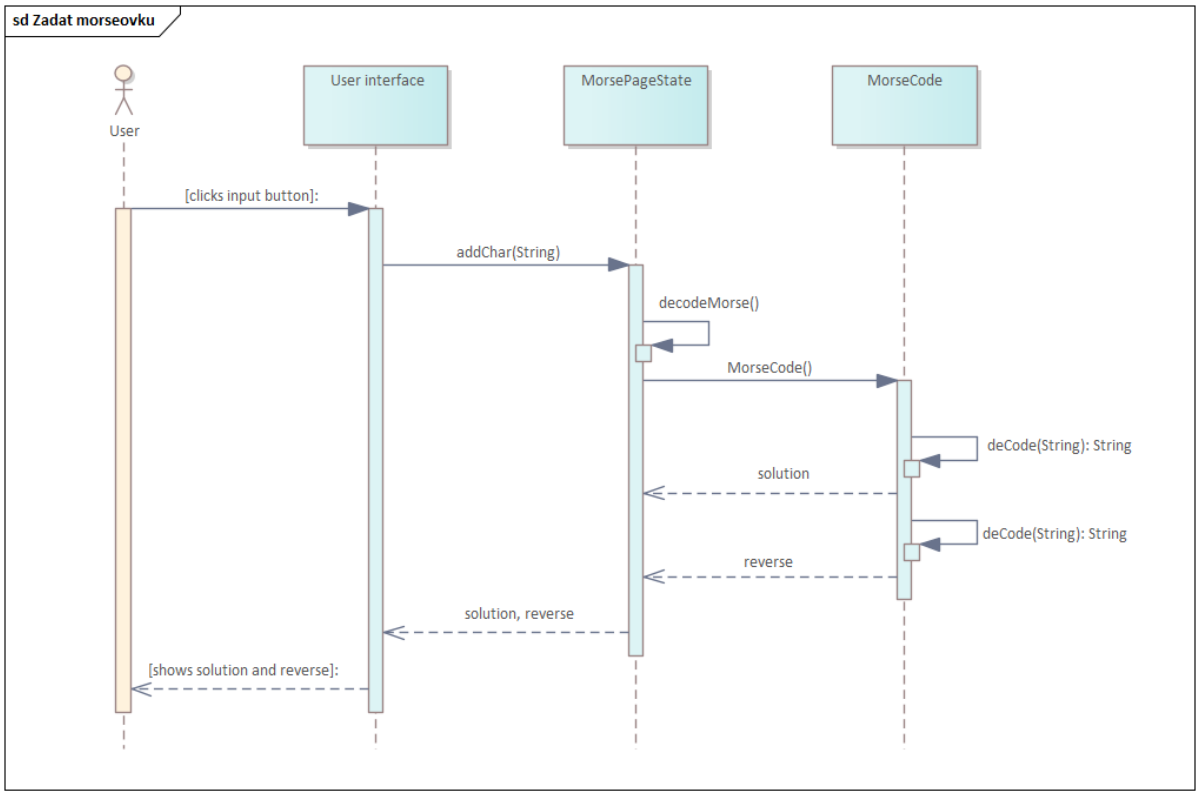


Obrázek 4.6: Sekvenční diagram vybírání dekodéru morseovky a zobrazení nápovědy. Zdroj: autorka práce.

- Z hlavní stránky si uživatel vybere dekodér Morseovy abecedy, který se postaví ze třídy **MorsePage**. Ta však dále nereaguje na příkazy od uživatele, o to se stará State třída **MorsePageState**, která se stará dále o komunikaci s uživatelem.
- **createAppbar(context):PreferredSizedWidget** vytváří záhlaví aplikace.
- **build(context):Widget** vrací postavený widget. Používá se tedy na tvorbu vlastních úseků (widgetů) stránky.

4.1.7.3. Zadat jeden znak morseovky

Na obrázku 4.7 níže je vyobrazen sekvenční diagram procesu zadání jednoho znaku morseovky.

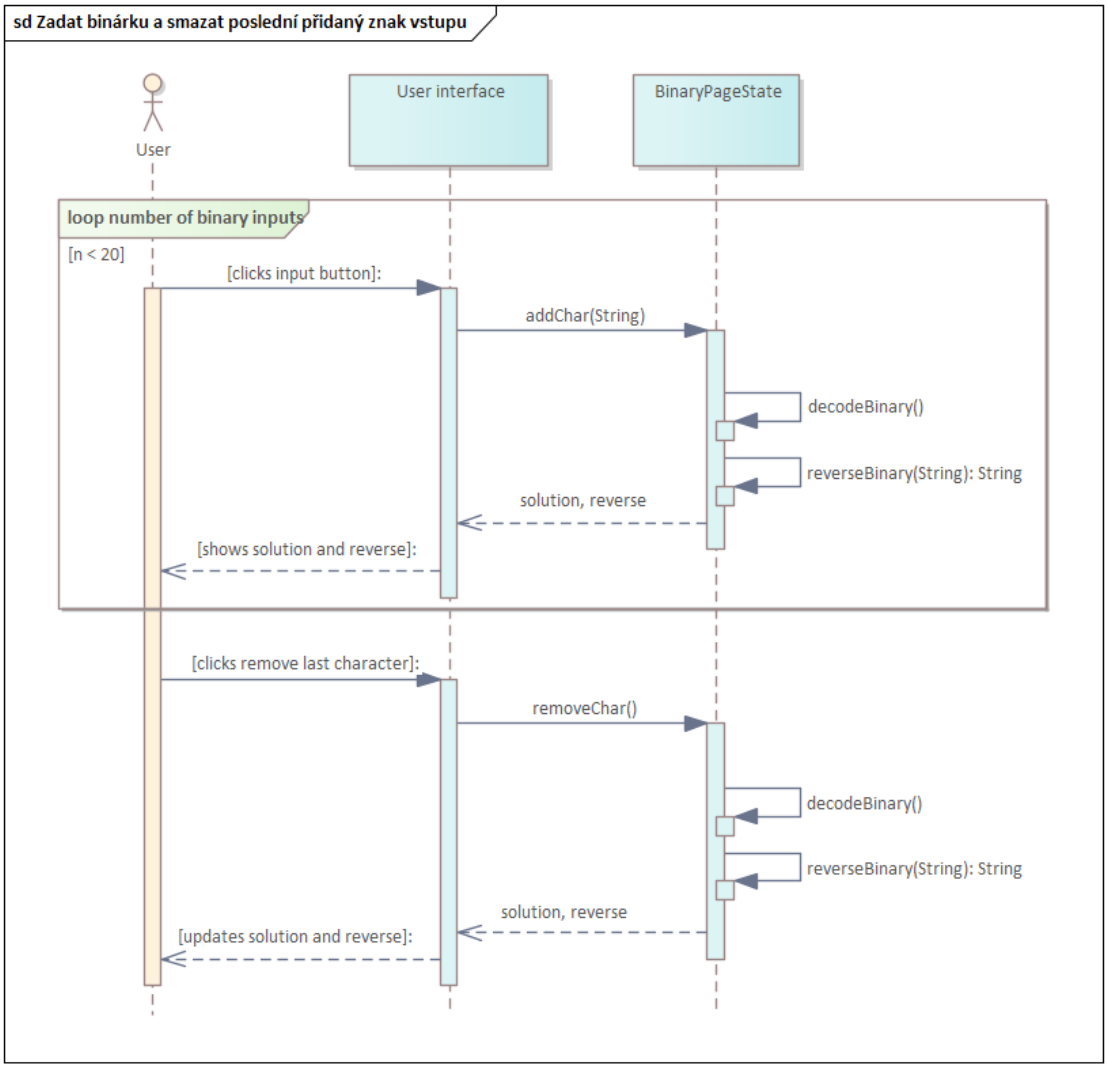


Obrázek 4.7: Sekvenční diagram zadání jednoho znaku morseovky. Zdroj: autorka práce.

- Uživatel klikne na tlačítko tečky nebo čárky, znak se přidá jako parametr do metody **addChar(String)**.
- **decodeMorse()** vytvoří objekt **MorseCode**, který zavolá metodu **deCode(String):String** a ta dekóduje řetězec.

4.1.7.4. Zadat binární soustavu a smazat poslední přidaný znak

Na obrázku 4.8 níže je vyobrazen sekvenční diagram procesů zadání vstupu binární soustavy a smazání posledního přidaného znaku.



Obrázek 4.8: Sekvenční diagram zadání vstupu binární soustavy a smazání posledního vstupu. Zdroj: autorka práce.

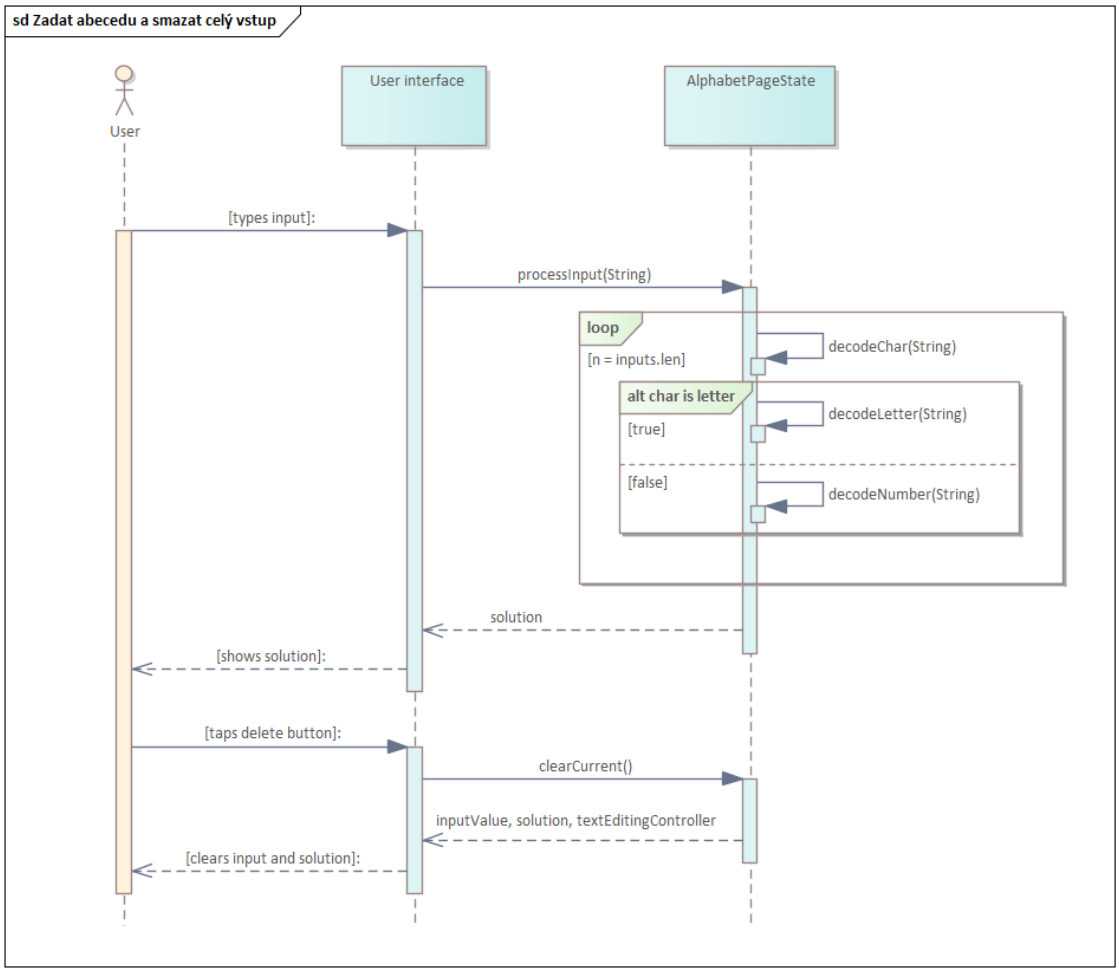
- Uživatel klikne na tlačítko nuly nebo jedničky, znak se přidá jako parametr do metody **addChar(String)**.
- **decodeBraille()** dekóduje řešení. Z metody se zavolá funkce **reverseBinary(String):String**, která vymění jedničky a nuly ve vstupu. Z tohoto výstupu se dekóduje opačný výsledek.

- Je možnost mazání posledního přidaného znaku pomocí metody **removeChar()**.

- Zmáčknutím tlačítka na smazání všech uložených řešení se neodebere pouze poslední, ale všechny znaky. Proces ale v diagramu vypadá identicky, proto ho znázorňovat je zbytečné. Je tomu tak u každého dekodéru.

4.1.7.5. *Zadat abecedu a smazat celý vstup*

Na obrázku 4.9 níže je vyobrazen sekvenční diagram procesů zadání vstupu abecedy a smazání celého vstupu.



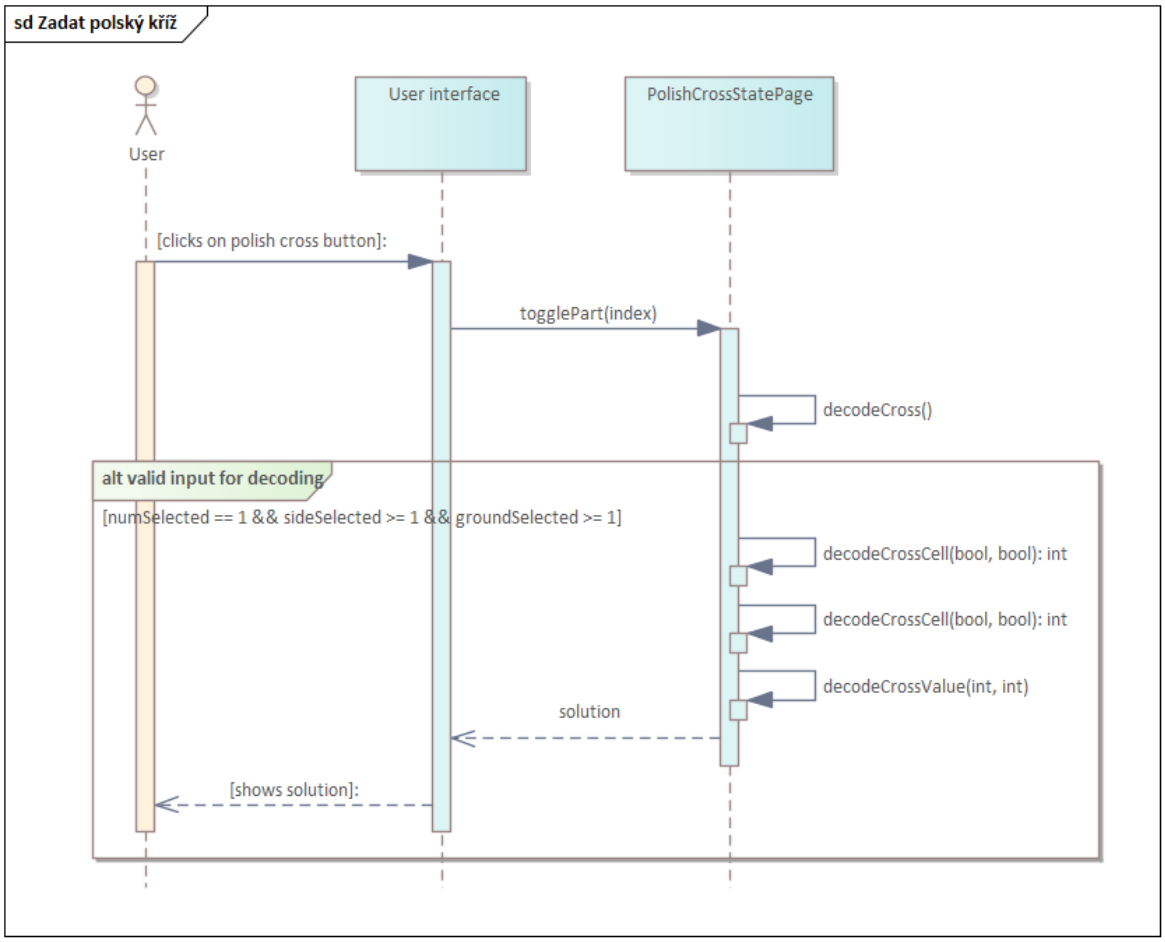
Obrázek 4.9: Sekvenční diagram zadání abecedy a smazání celého vstupu. Zdroj: autorka práce.

- Do dekodéru abecedy se jako v jediném může zadat více vstupů na jednou, každý jednotlivý vstup se pak zpracuje sám.
- Metoda **decodeChar()** rozpoznává, zda se jedná o validní vstup splňující kritéria – písmena anglické abecedy (česká abeceda bez diakritiky a „Ch") nebo celé číslo.
- **decodeChar()** pak přesměruje na korespondenční metody – **decodeLetter()** pro zadaná písmena a **decodeNumbe()** pro zadaná slova.

- Smazáním vstupu se vyprázdní všechny zadané znaky a řešení. Tak je tomu u každého dekodéru.

4.1.7.6. Zadat polský kříž

Na obrázku 4.10 níže je vyobrazen sekvenční diagram procesu zadávání vstupu polského kříže.



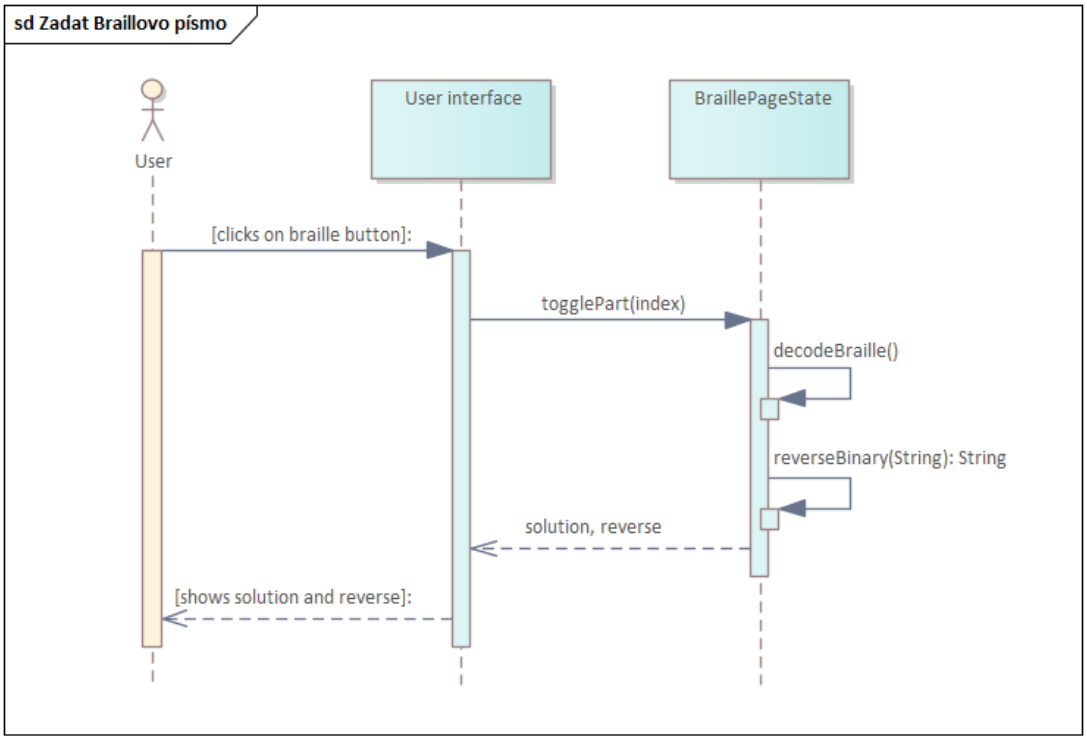
Obrázek 4.10: Sekvenční diagram zadávání vstupu polského kříže. Zdroj: autorka práce.

- Při kliknutí na tlačítko polského kříže se zavolá metoda **togglePart(int)**, která zjistí status konkrétního tlačítka (vybrán/nevybrán) a změní ho.
- Zavolá se metoda **decodeCross()**, která zjišťuje, zda vstup je vhodný k dekódování, ta v kladném případně postupně zavolá další metody.
- **decodeCrossCell(bool, bool):int** bere jako parametr, status tlačítek a hledá, která buňka v mřížce 3x3 je aktivní. První zavolání vyhledá aktivní řádek a druhé zavolá aktivní sloupec.
- **decodeCrossValue(int, int)** je metoda, která bere jako parametr výsledky **decodeCrossCell(bool, bool):int** a v konkrétní buňce hledá jednu z 3 aktivních pozic. Tyto informace dohromady stačí k jednoznačnému dešifrování písmenka.

- Obdobný proces probíhá i u dekodéru semaforové abecedy, tam však stačí volat metodu **decodeSemaphore()**, ve které se najde hned výsledek.

4.1.7.7. Zadat Braillovo písmo

Na obrázku 4.11 níže je vyobrazen sekvenční diagram procesu zadání vstupu Braillova písma.



Obrázek 4.11: Sekvenční diagram zadávání Braillova písma. Zdroj: autorka práce.

- Při kliknutí na tlačítko Braillova písma se zavolá metoda **togglePart(int)**, která zjistí status konkrétního tlačítka (vybrán/nevybrán) a změní ho.
- Zavolá se metoda **decodeBraille(),** která dekóduje výsledek. Z ní se spustí metoda **reverseBinary(String):String**, která pomůže k dekódování opačného výsledku.

4.1.8. Shrnutí funkčního návrhu

V této kapitole se definovaly funkční a nefunkční požadavky a vytvořily diagramy: procesní diagram, diagram komponent, tříd a sekvenční diagramy. Funkční požadavky specifikovaly, co všechno by měla aplikace splňovat. Procesní diagram poskytl vizuální reprezentaci probíhaných aktivit uživatele a systému. Diagram komponent rozčlenil systém na jednotlivé části a ukázal závislosti mezi nimi. Diagram tříd definoval strukturu systému, třídy, jejich atributy a metody. Sekvenční diagramy ukazovaly chování systému a interakce mezi objekty v konkrétních scénářích. Dohromady tyto nákresy vytvořily komplexní dokumentaci, která je základem pro úspěšný vývoj softwarového systému.

4.2. Grafický návrh

Tato kapitola popisuje, jak vznikal vzhled aplikace, jak a proč se měnil a vysvětluje určitá rozhodnutí ohledně vizuálu. Ten měl být jednoduchý, přehledný a (subjektivně) hezký.

Nejprve je přiblížena teorie a filozofie návrhu a poté jsou ukázány konkrétní náčrty, které byly vytvořeny v aplikaci *Figma*. Design ikonek stránek i celé aplikace byl vyroben v programu *Inkscape*, ve kterém se dají vyrábět vektorové obrázky, které jsou vhodné pro loga díky škálovatelnosti.

4.2.1. Low-fidelity (lo-fi)

Jedná se o takzvaný drátěný model (wireframe). V low-fidelity prototypu se neřeší vzhled, barevnost, písmo atd. Prototypování má za úkol pouze vyřešit stavbu stránek jako takovou. [27] Hlavním cílem těchto nákresů je zhodnotit, jaké interakce uživatele a aplikace jsou potřeba, jaká všechna tlačítka, dialogová a textová okna musí být k dispozici. Stačí předat sdělení bez dalšího pozlátka, proto bývají pouze černo-bílé.

4.2.2. High-Fidelity (hi-fi)

Low-fidelity nákresy se v dalších fázích překreslují za účelem reálnější představy konečného produktu do tzv. high-fidelity návrhů. V této fázi se klade důraz na design, barevné schéma, přesné rozložení stránek a jejich komponent. *Kvalitní grafika podporuje pozici značky, zvyšuje důvěryhodnost webu a zvyšuje tolerantnost návštěvníků k chybám.* [28]

4.2.3. Barevné schéma

Každá stránka má velice podobné rozložení, které dodává aplikaci uniformitu. Každá se rozlišuje barvou tak, aby bylo možné v rychlosti soutěže na první pohled poznat, na jaké stránce se uživatel nachází. Byla tedy nutnost vybrat 6 barev, které si nejsou moc podobné a nejsou splývavé s barvami pozadí.

V seznamu níže jsou vypsány nejlepší barvy a jejich hexadecimální reprezentace:

	Fialová	**Žlutá**	**Zelená**	**Modrá**	**Oranžová**	**Růžová**
Tlačítka	#DBD6F8	#FCE5AD	#CAE2D8	#90C9FF	#FFC1A7	#FFB9C8
Texty	#867ACA	#AD8932	#558F77	#397DBC	#C86941	#EC6885

- **Bílá** – pozadí denního režimu
  Komentář: Bílá barva nejlépe odráží světlo, při venkovních šifrovačkách na slunci bude text na obrazovce nejčitelnější. [29]
- **Černá** – pozadí nočního režimu
  Komentář: Během nočních akcí je pro oči příjemnější tmavé pozadí. [30]

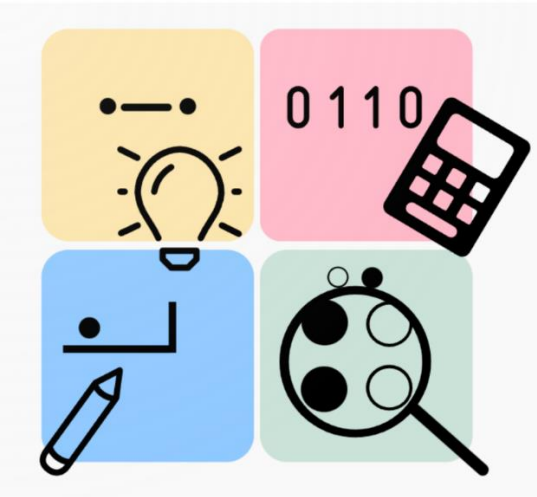
4.2.4. Vlastní návrhy

V této kapitole je představen design aplikace a vysvětlení filozofie návrhů. Nejprve se předvedou navržené ikonky a poté samotné prototypy stránek.

4.2.4.1. Ikonky

4.2.4.1.1. Ikonka aplikace

Každá aplikace musí mít ikonku na hlavní stránce telefonu. Přestože je Flutterem k dispozici základní, ta však o aplikaci *Šifrovací nástroje* nic neříká. Chtělo to tedy takovou, z níž je možné jasně vidět, oč se jedná. Na obrázku 4.12 níže je vyobrazena ikonka aplikace.



Obrázek 4.12: Ikonka aplikace. Zdroj: autorka práce.

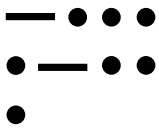
- 4 bloky reprezentující dekodéry v aplikaci: Morseovu abecedu, binární soustavu, polský kříž a Braillovo písmo.
- Barvy bloků odpovídají barvám přímo v aplikaci, aby se v uživateli utvrdil pocit, že se jedná opravdu o ikonku k programu.
- V každém bloku je zjednodušená ukázka šifry a symbol spojovaný s logickými a luštícími úkoly.

- Pro pozorné byly do ikonky ukryty dva skryté významy (v angličtině takzvané „easter egg“). Řetězec u morseovky vypadá jako znuděný obličej a poloha žárovky barevně odpovídá rozsvícené žárovce (symbolu nápadu/osvícení), kde žlutá barva reprezentuje světlo a modrá stříbřitou barvu objímky.

4.2.4.1.2. Miniatury šifer

Pro hlavní stránku aplikace byly vytvořeny zjednodušené podoby jednotlivých šifer pro jednodušší orientaci, jelikož si uživatel všimne dříve ikonky než textu. [31] Níže jsou vyobrazeny obrázcích 4.13. – 4.18 ikonky na jednotlivých dekodérů.

Morseova abeceda

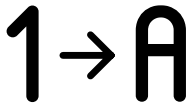


- Jednoduchá kombinace teček a čárek.
- Vybrána písmenka maximální i minimální velikosti na znázornění rozsahu vstupu.

Binární soustava

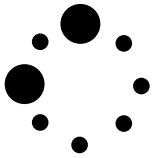
- Jednoduchá kombinace nul a jedniček.

Abeceda



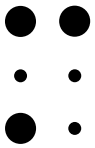
- Převod mezi číslicí a písmenem.
- Použité bylo A, aby bylo jednoznačně vidět, že šifra indexuje od 1.

Semaforová abeceda



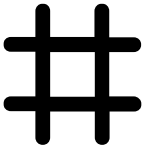
- Znázorněné rozložení kruhu semaforové abecedy.
- Vybrané pozice jsou větší, aby přitahovaly pozornost.
- Vybrány dvě pozice jako nápověda, jak má vypadat vstup.

Semaforová abeceda



- Znázorněné rozložení šestice kruhů.
- Vybrané pozice jsou větší, aby přitahovaly pozornost.

Polský kříž



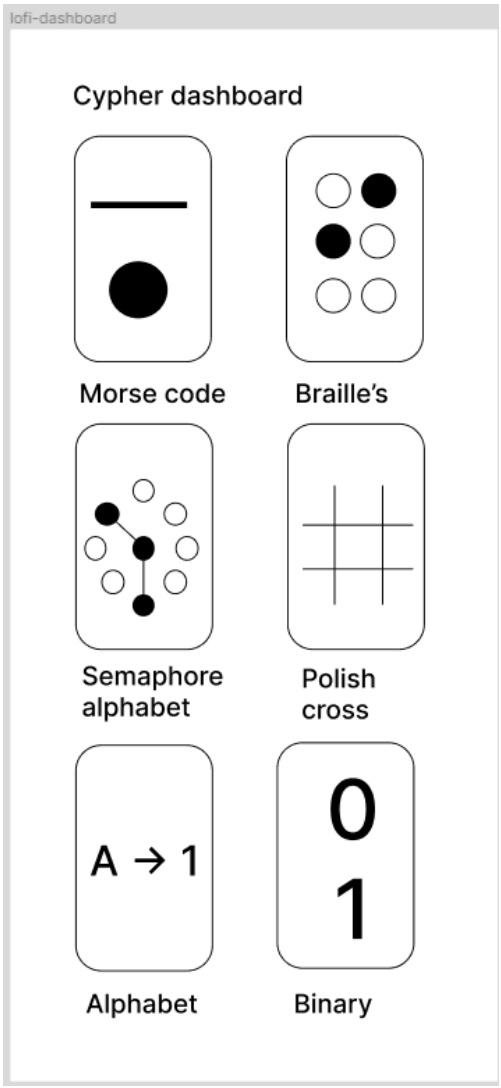
- Znázornění celého polského kříže by bylo nepřehledné.
- Lidem známý symbol křížku (hashtag) odpovídá rozložení hranic.

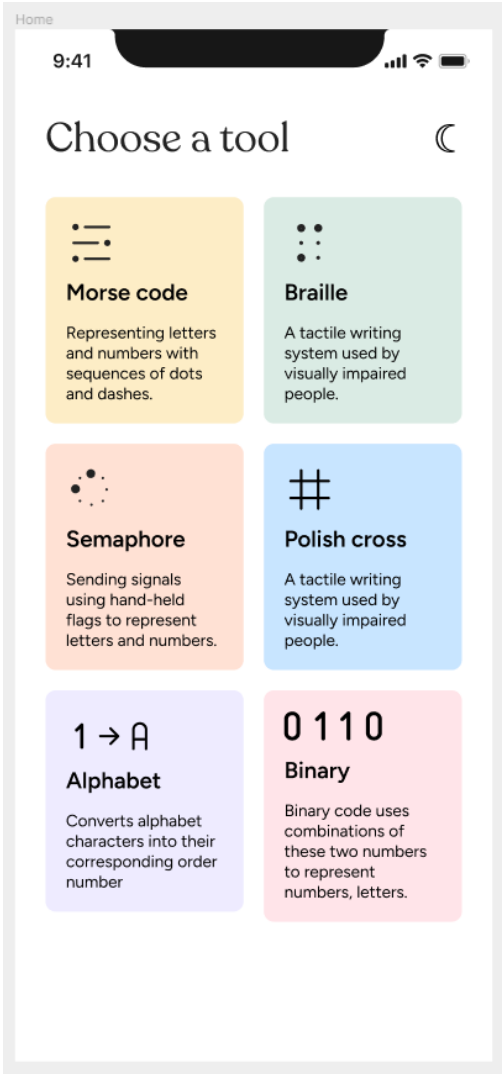
Obrázky 4.13-4.18: Ikonky dekodérů v aplikaci. Zdroj: autorka práce.

4.2.4.2. *Stránky aplikace*

V této části se na obrázcích 4.19 až 4.33 porovnávají low a high-fidelity návrhy s komentáři o volbách a změnách mezi jednotlivými stadii. Návrhy jsou anglicky, jelikož byly vytvořeny pod dohledem designérky, která nemluví česky.

4.2.4.2.1. Hlavní stránka

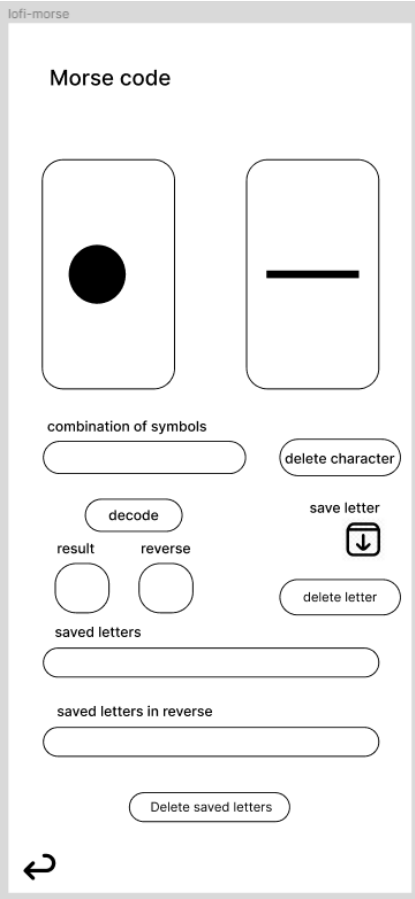


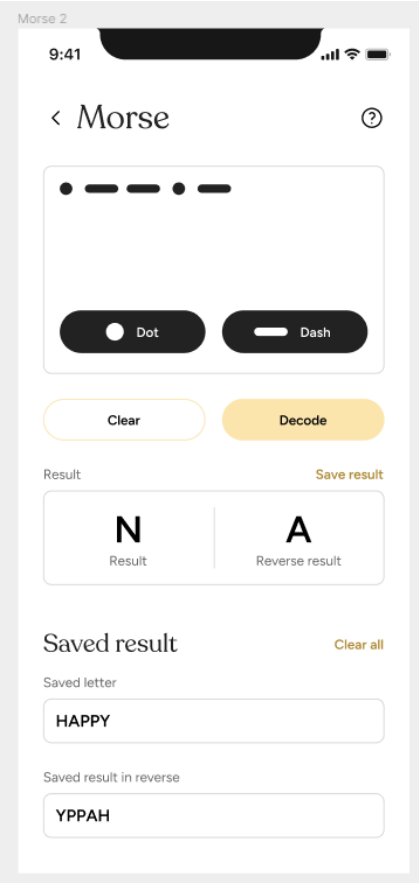


Obrázek 4.19 a 4.20: Low a high fidelity prototypy hlavní stránky. Zdroj: autorka práce.

Low-fidelity	High-fidelity
• Hlavní stránka slouží jako rozcestník, volby musí být zřetelně viditelné. • Na kartě dekodéru je ikonka napovídající princip šifry pro případ, že by uživatelé podle jména nepoznali, o jaký dekodér se jedná.	• Zmenšení ikonek a přidání jména dekodéru do karty pro ucelený vzhled • Přidání popisu ke každé šifře • Barevné oddělení karet • Přidání symbolu měsíce na změnu denního a nočního režimu

4.2.4.2.2. Morseova abeceda

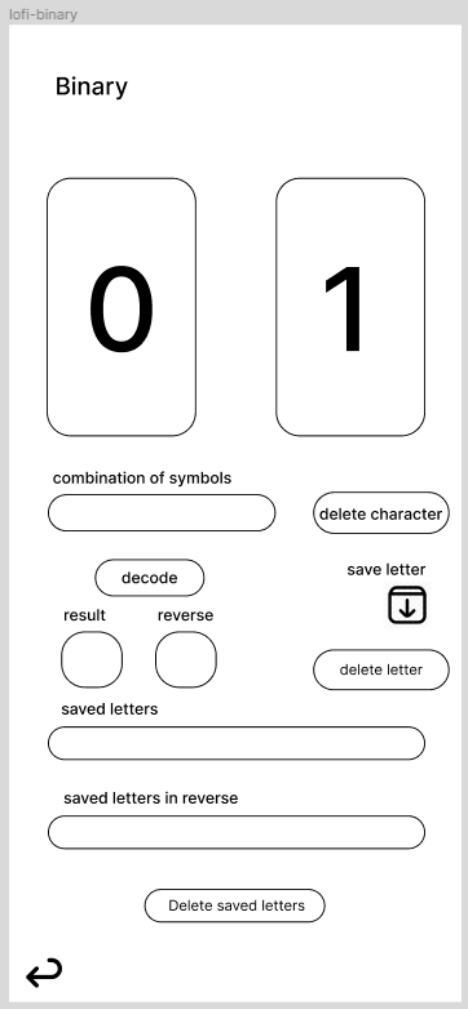


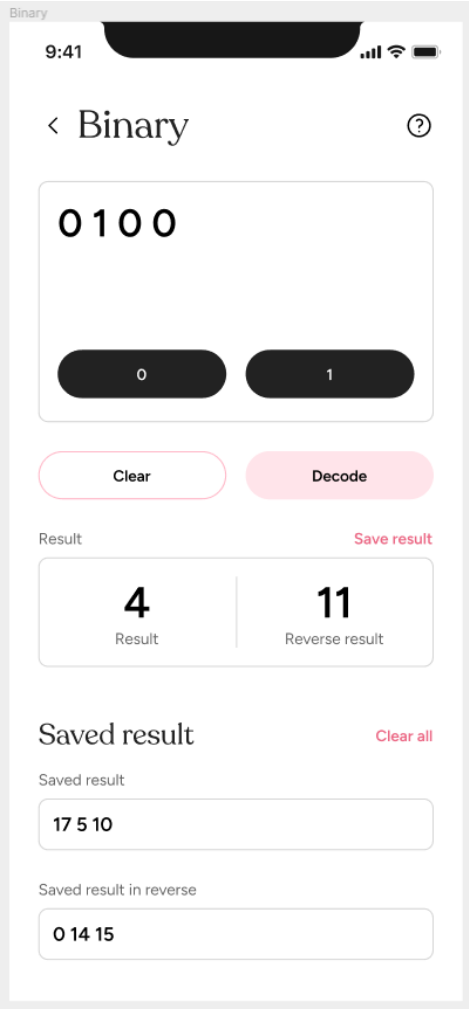


Obrázek 4.21 a 4.22: Low a high fidelity prototy dekodéru morseovky. Zdroj: autorka práce.

Low-fidelity	High-fidelity
• Tři bloky stránky – vstup, řešení, uložené řešení • Dvě tlačítka, při kliknutí přidá na konec vstupu tečku nebo čárku. • Aktuální vstup se vypisuje do kolonky *combination of symbols*. • Tlačítko *delete character* odebere poslední znak vstupu. • Tlačítko *decode* dekóduje vstup. • Tlačítko *delete letter* odebere celý vstup a aktuální řešení (i opačné). • Ikona *save letter* přidá aktuální řešení na konec řetězce uložených výsledků do pole *saved letters* (ekvivalentně opačné řešení). • Tlačítko d*elete saved letters* smaže uložený řetězec řešení (i opačných).	• Zmenšení tlačítek s tečkou a čárkou • Zvětšení aktuálního vstupu (na první pohled viditelné) • Zvětšení řešení (na první pohled viditelné) • Přidání ikonky pro nápovědu. • Přesunutí tlačítka zpět nahoru, více intuitivní • Odebrání tlačítka na smazání posledního znaku, moc přeplácané • Tlačítko *delete letter* zkráceno na *clear,* s bezbarvým pozadím, kontrastuje s důležitějším tlačítkem *decode* • Tlačítka *save result* a *clear all* textová, rozbíjí monotónnost ostatních tlačítek

4.2.4.2.3. Binární soustava

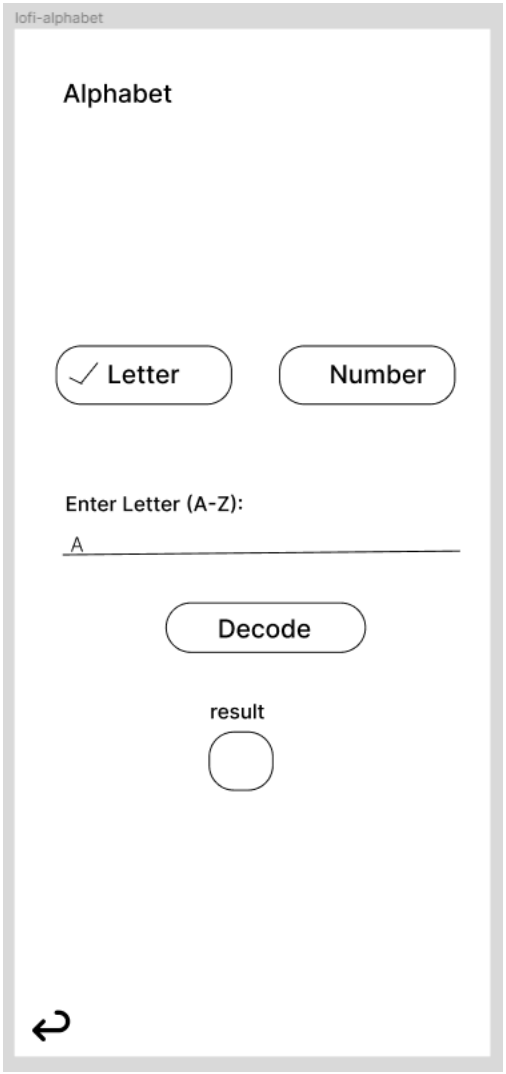


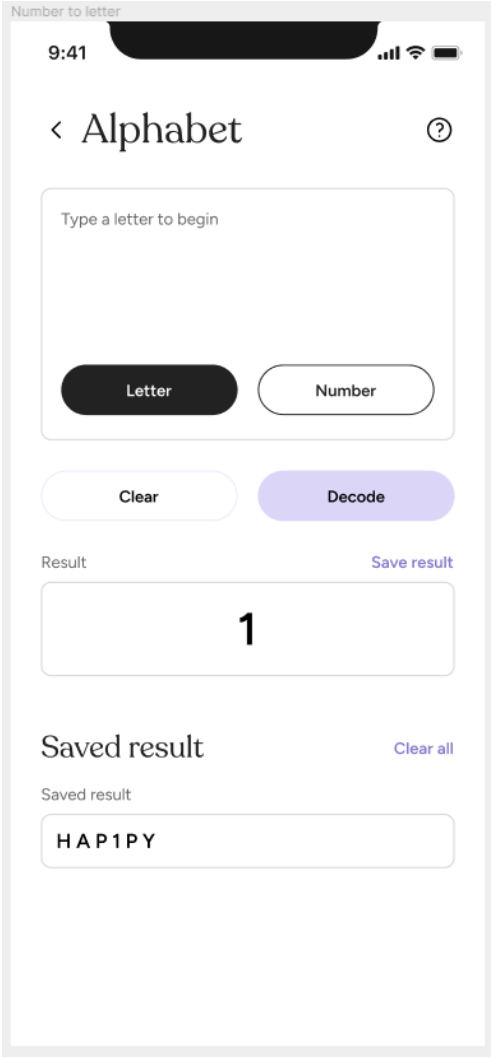


Obrázek 4.23 a 4.24: Low a high fidelity prototypy dekodéru binární soustavy. Zdroj: autorka práce.

Low-fidelity	High-fidelity
• Funkčně identické jako stránka morseovky, kromě vstupu. • Tlačítka vstupu místo teček a čárek jsou nuly a jedničky.	• Stejné změny jako u stránky s morseovkou • Uložené výsledky mají mezi sebou mezeru z důvodu rozpoznávání víceciferných čísel.

4.2.4.2.4. Abeceda

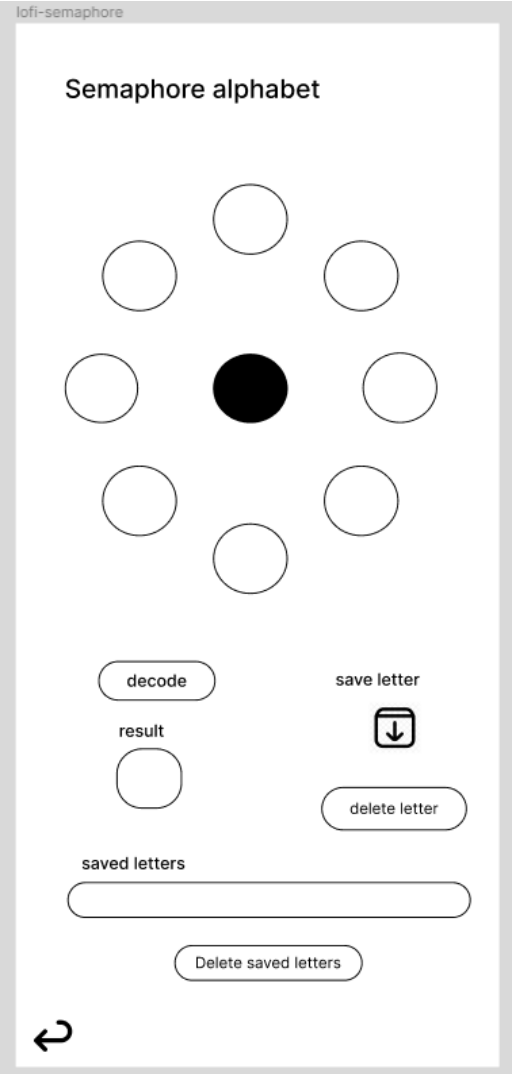


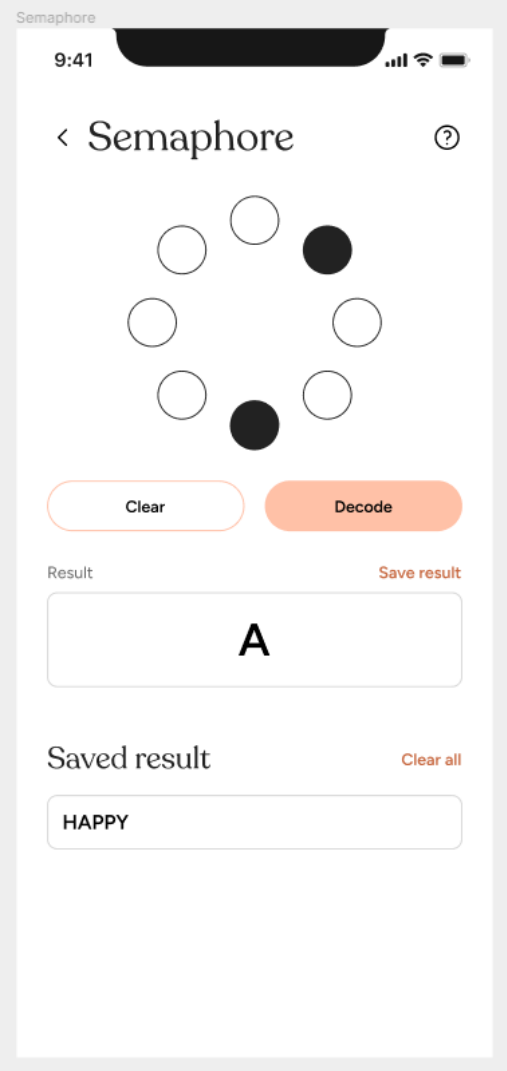


Obrázek 4.25 a 4.26: Low a high fidelity prototypy dekodéru abecedy. Zdroj: autorka práce.

Low-fidelity	High-fidelity
• Tlačítka výběru, zda se bude zadávat písmenko nebo číslo. • Textové pole, kam se bude zadávat vstup pomocí klávesnice. • Vstupem je jedno číslo nebo písmenko. • Nebylo v plánu ukládat výsledky, nebylo jasné, jak vyřešit kombinování čísel a písmenek.	• Design předělán, aby byl podobný ostatním. • Tlačítka *„letter“* a *„number“* mění mód dekódování. • Tato šifra nemá opačné řešení, sekce s řešením stále ale vypadá jako u předchozích šifer. • Další změny ve stejném rázu jako u předchozích stránek.

4.2.4.2.5. Semaforová abeceda



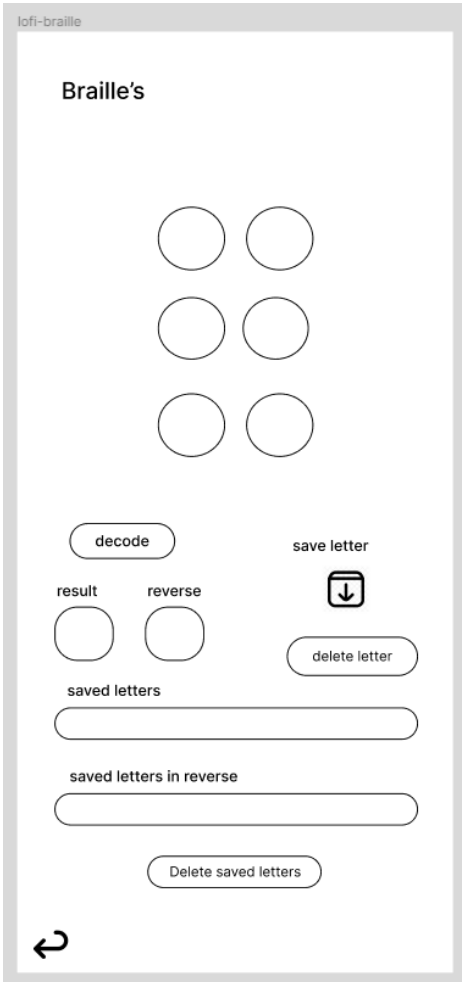


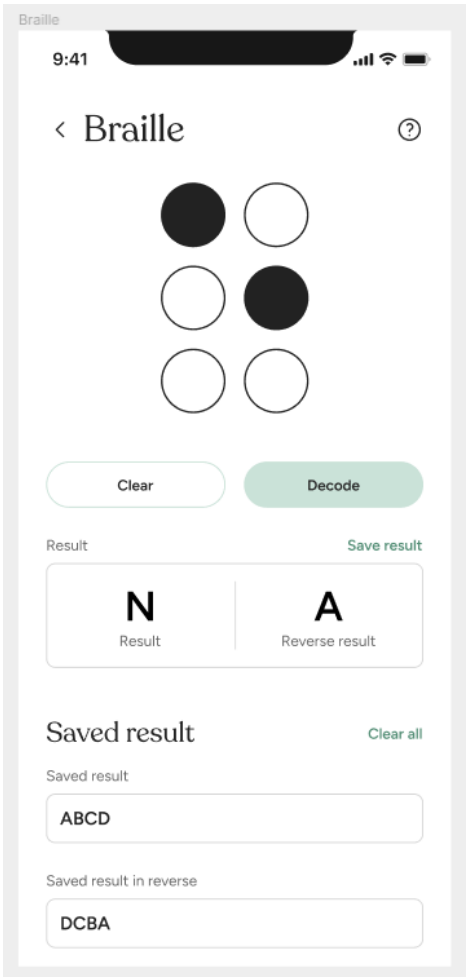
Obrázek 4.27 a 4.28: Low a high fidelity prototypy dekodéru semaforové abecedy.

Zdroj: autorka práce.

Low-fidelity	High-fidelity
• Kruhová tlačítka se kliknutím vyberou/nevyberou. • Zbytek funguje jako u předchozích dekodérů.	• Smazání prostředního nefunkčního kolečka, přehlednější pro malé obrazovky. • Další změny ve stejném rázu jako u předchozích stránek.

4.2.4.2.6. Braillovo písmo





Obrázek 4.39 a 4.30: Low a high fidelity prototypy dekodéru Braillova písma. Zdroj: autorka práce.

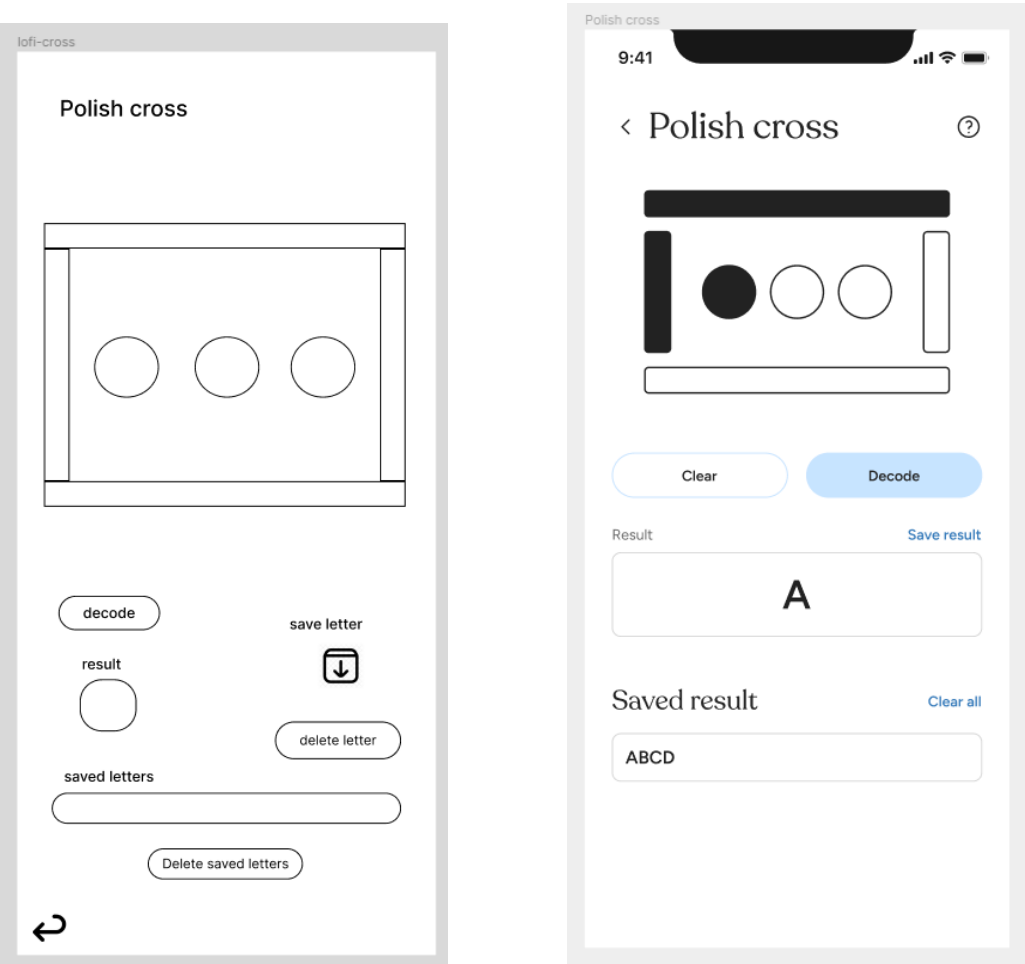
Low-fidelity	High-fidelity
• Kruhová tlačítka se kliknutím vyberou/nevyberou. • Zbytek funguje jako u předchozích dekodérů.	• Další změny ve stejném rázu jako u předchozích stránek.

4.2.4.2.7. Otevřená nápověda

- Po stisknutí ikonky nápovědy vyskočí pop-up s nápovědou.
- Barva bloku se shoduje s barvou tlačítka každé stránky.

Obrázek 4.31: Prototyp nápovědy. Zdroj: autorka práce.

4.2.4.2.8. Polský kříž



Obrázek 4.32 a 4.33: Low a high fidelity prototypy dekodéru polského kříže. Zdroj: autorka práce.

Low-fidelity	High-fidelity
• Kruhová a obdélníková tlačítka se kliknutím vyberou/nevyberou. • Zbytek funguje jako u předchozích dekodérů.	• Zkrácení postranních hranic, vypadá moderněji. • Další změny ve stejném rázu jako u předchozích stránek.

4.2.5. Vyhodnocení grafického designu

High-fidelity prototyp nebyl sám otestován, v rámci uživatelských testů (viz kapitola 6.2 Uživatelské testování) se zkoušející zaměřovali nejen na funkčnost aplikace, ale i její vzhled. Finální úpravy byly tedy vytvořeny až v dalších fázích projektu.

Tmavý režim nebyl načrtnutý, jelikož vypadá stejně jako high-fidelity návrhy, pouze se vymění černá a bílá barva.

4.3. Shrnutí kapitoly návrhu

V této kapitole bylo definováno, jak by aplikace měla vypadat z hlediska funkcionalit a grafického návrhu. Identifikovaly se hlavní procesy a popsaly se jejich požadavky. V grafickém návrhu byla vybrána barevná paleta a uspořádány prvky, aby se zajistil intuitivní a estetický design. Utříděné myšlenky a návrhy se zformovaly v přehledných diagramech a prototypech aplikace.

5. Implementace mobilní aplikace

Tato kapitola se zaměřuje na implementaci aplikace *Šifrovací nástroje*. Nejprve jsou stručně představeny různé metody vývoje mobilních aplikací a jejich vzájemné srovnání. Následně je popsán samotný projekt: jeho struktura, použité technologie a ukázky kódu s vysvětlením.

5.1. Cross-platform vývoj

Výběr, jaké technologie použít pro vývoj mobilních aplikací je velice klíčový, jelikož určuje, na jakých zařízeních může aplikace běžet. Dva hlavní operační systémy u telefonů jsou Android a IOS a každý z nich primárně podporuje jiné jazyky – u Androidu se jedná o Kotlin (dříve Javu) a u IOSu o Swift (dříve Objective-C).

Výše zmíněným jazykům se říká nativní pro danou platformu, není to však jediný způsob vytváření programu. Druhým je tzv. cross-platform, který, jak název napovídá, může sdílet stejný kód napříč různými platformami. Není to však stoprocentní zachránce, oproti nativnímu vývoji má své i nějaké nevýhody.

Výčtový seznam níže popisuje vztah cross-platform vývoje k nativnímu:

Výhody	**Nevýhody**
• Stejný kód na vše • Rychlejší vývoj • Levnější vývoj • Lehčí správa • Přepisování změn v kódu při běhu	• Pomalejší kód • Náročnější na úložiště • Menší komunita • Ne všechny funkce podporovány na všech platformách

Po zvážení, jaký přístup zvolit, se autorka rozhodla zvolit cross-platform vývoj. Mínusy této metody aplikaci *Šifrovací nástroje* tolik nezatěžují, aplikace nebude příliš velká a výpočetně náročná, takže si může dovolit být neoptimalizovaná. Naopak výhoda toho, že by se dala nasadit jak na telefony s Androidem, tak i IOSem (na kterém žádný ekvivalent nebyl nikdy k dispozici), je klíčová.

Je možné z kódu vytvořit i desktopovou aplikaci, která sice není primárním cílem, ale může být užitečná jako další výstup práce. Spousta soutěžících se účastní přes internet (viz graf počtu přihlášených týmů v kapitole *Účastník logických her*) a v pohodlí domova je pro soutěžící příjemné mít alternativu na počítači, který má větší obrazovku.

5.1.1. Populární cross-platform jazyky

Výběrem techniky vývoje však ještě rozhodování neskončilo, jelikož cross-platform jazyků je několik. Mezi nejznámější patří:

- **React Native** – framework od Mety (např. Facebook) stavěná na Reactu (JavaScriptu)
- **Flutter** – framework od Googlu, stavěný na jazyce Dart
- **Xamarin** – framework od Microsoftu, stavěný na jazyce C#

Po dlouhém rozmýšlení si autorka vybrala k práci Flutter, jelikož jsem ani s jedním frameworkem dříve nikdy nepracovala a Dart funguje na podobném principu jako jazyky C++ a Java, se kterými se již oproti Reactu a C# v minulosti setkala.

5.2. Aplikace *Šifrovací nástroje*

5.2.1. Použité technologie pro vývoj

- Verze Flutteru: **Flutter 3.19.3**
- IDE: **Android Studio 2023.1.1 Hedgehog** – IDE s integrovaným virtuálním zařízením, lze testovat aplikace na mobilu bez nutnosti připojení fyzického zařízení
- Importované balíčky (dependence):
    - **morse_code_translator**: ^0.0.1 – překladač morseovky
    - **flutter_svg**: ^2.0.10+1 – pro přidání vlastních .svg (vektorových) obrázků
    - **string_validator**: ^1.0.2 – pro kontrolu Stringů
    - **auto_size_text**: ^3.0.0 – škálovatelný text podle rodiče
    - **window_manager**: ^0.3.8 – pro nastavení velikosti obrazovky u desktopu

5.2.2. Struktura projektu

Flutterové projekty jsou všechny standardizované, z kořenové složky vedou cesty do dalších důležitých adresářů. Zde je stručný výpis struktury důležité pro programátora:

- **android/:** a **ios/:** tvoří funkční mobilní aplikaci pro svůj operační systém
- **lib/:** zde se nachází soubory **.dart**, ve kterých se vyvíjí kód
- **test/:** zde se nachází testy zkoumající přesnost aplikace
- **web/:** zaručují běh aplikace ve webovém prohlížeči
- **Linux/:** a **Windows/:** a **Macos/:** tvoří funkční desktopovou aplikaci pro svůj operační systém
- **assets/:** přidaný adresář, do kterého se vkládají obrázky/ikony používané v projektu
- **fonts/:** přidaný adresář držící vlastní písma použitá v projektu
- **pubspec.yaml:** soubor obsahující metadata o verzi Flutteru, importovaných dependencích, assetech a dalších.

Detailnější rozbor projektu se dá najít v tomto článku[8].

[8] https://medium.com/@logeshgcp/understanding-the-flutter-project-structure-84de4ec3ce5f

5.3.3.1. Adresář lib/: a struktura kódu

Jak již bylo zmíněno, sem si programátor píše vlastní kód. Startovací bod programu je však daný funkcí **main()**, která se konvenčně nachází v souboru **main.dart**, nutností tomu však není.

Kód byl v tomto projektu byl rozdělen do několika souborů a složek/balíčků (package):

lib/:

- **const** – složka s konstantami využívanými v projektu
  - **colors.dart** – soubor s hexadecimálním vyjádřením vlastních barev stránek dekodérů
  - **strings.dart** – soubor se Stringy k nápovědám (tip na hlavní stránce a instrukce)

- **decoderPages** – balíček se soubory reprezentujícími jednotlivé stránky dekodérů (logika + rozložení designu)
  - **alphabet.dart** – soubor se stránkou abecedního dekodéru
  - **binary.dart** – soubor se stránkou dekodéru binární soustavy
  - **braille.dart** – soubor se stránkou dekodéru Braillova písma
  - **morseCode.dart** – soubor se stránkou dekodéru morseovky
  - **polishCross.dart** – soubor se stránkou dekodéru polského kříže
  - **semaphore.dart** – soubor se stránkou dekodéru semaforové abecedy

- **generalBuildHelpers** – balíček s metodami vracející widgety, které využívá více stránek najednou
  - **resultPart.dart** – soubor s metodami, které vrací widgety v části stránky s výsledkem
  - **savedResultPart.dart** – soubor s metodami, které vrací widgety v části stránky s uloženým výsledkem
  - **inputPart.dart** – soubor s metodami, které vrací widgety v části stránky vstupu u morseovky a binární soustavy
  - **general.dart** – soubor s dalšími metodami využívanými několikrát v kódu, které ale nepatří do žádné z předchozích částí

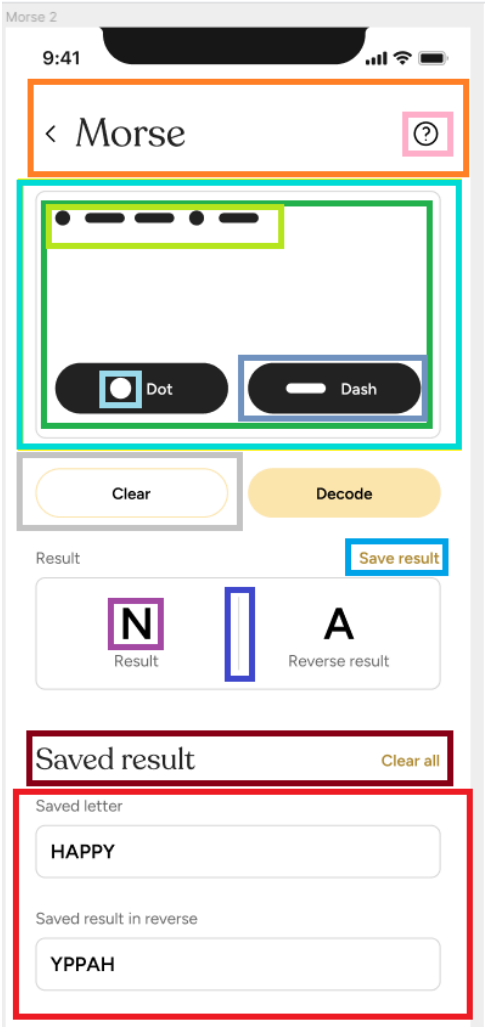
- **customAppbar.dart** – soubor, který se stará o výrobu vlastního Appbaru (horní lišty) a nápovědy k dekodérům
- **homePage.dart** – soubor, který se stará o výrobu hlavní stránky
- **theme_provider.dart** – soubor starající se o denní a noční režim
- **main.dart** – soubor se startovací metodou main()

5.3.4. Widget

Základním prvkem Flutter aplikací jsou „Widgety“, ze kterých pomocí zanořování tvoříme strom. Jedná se o abstraktní immutable třídu, která nám reprezentuje určitou část uživatelského rozhraní. [32] Flutter jich nabízí celou řadu, které si člověk mění podle svých představ a skládá je dohromady.

5.3.4.1.1. Použité widgety

Kromě základních stavebních kamenů Flutter poskytuje spoustu již funkčních widgetů, které si uživatel může dále upravit. Níže je seznam použitých v práci a jejich ukázka vyznačená barevně na obrázku 5.1:



Obrázek 5.1: Ukázka widgetů. Zdroj: autorka práce.

Row

- Widget skládající potomky do řady. Pomocí atr. **mainAxisAlignment** a **crossAxisAlignment** lze měnit rozložení v rámci řady.

Column

- Widget skládající potomky do sloupce. Rozložení viz řada.

VericalDivider

- Tvoří dělící čáru, lze měnit délka a šířka pomocí atr. **Indent** a **width.**

Appbar

- Tvoří horní lištu s názvem a šipkou zpět, vlastnost **actions** bere jako parametr pole widgetů s určitými funkcemi.

IconButton

- Tvoří tlačítko z ikonky, atribut **onPressed** zavolá funkci, co se provede po stisknutí tlačítka.

Icon

- Widget reprezentující ikonku.

Container

- Container je určitý úsek stránky. Atribut **decorations** a třída **BoxDecorations()** mu mohou změnit podobu.

Stack

- Widget, který umožňuje umístit další widgety přes sebe – tečky a čárky a tlačítka jsou položená na žlutém Containeru.

Positioned

- Určuje přesnou pozici vůči rodičovi pomocí atributů **top, left, right, bottom**.

ElevatedButton

- Widget tvořící tlačítko pomyslně nadzvednuté z papíru (3D efekt).

OutlinedButton

- Widget tvořící tlačítko s obrysem. 2D plochý vzhled.

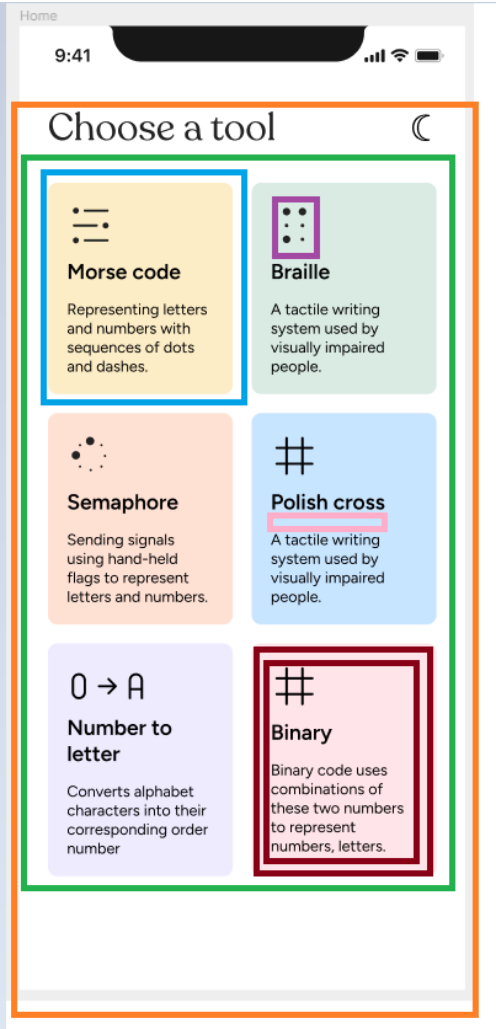
TextButton

- Tlačítko v podobě textu. Atribut **onPressed** spouští metodu pokliknutí.

Text

- Widget reprezentující text. Pomocí atributu **style** a třídy **TextStyle()** se mu mění design.

Níže na obrázku 5.2 je další seznam a ukázka použitých widgetů.



Scaffold

- Widget reprezentující hlavní strukturu uživatelského rozhraní stránky. Atribut **appbar** definuje horní lištu.

GridView

- Třída, jejíž instance se chovají jako widgety. Tvoří flexibilní tabulkové rozložení potomků. Vlastnost **crossAxisCount** určuje maximální počet potomků na jednom řádku.

Card

- Widget tvořící sekci stránky. Oproti Containeru je pomyslně nadzvednutý z papíru (3D efekt) a vlastností **elevation** určuje vrhaný stín.

SvgPicture

- Widget nabízený balíčkem **flutter_svg**, který umožňuje přidání vlastních vektorových obrázků.

SizedBox

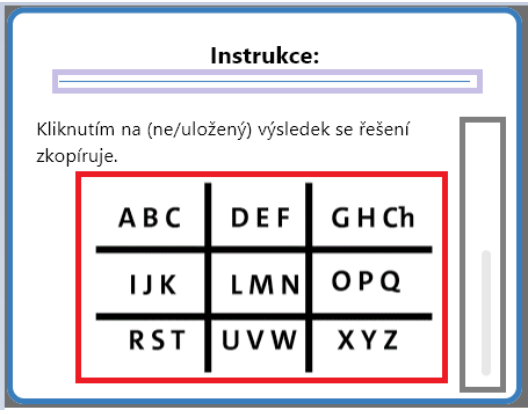
- Widget reprezentující část stránky s danou velikostí. Používá se pro tvorbu mezer mezi Widgety.

Padding

- Widget, který odsadí potomky o zadanou velikost.

Obrázek 5.2: Ukázka widgetů. Zdroj: autorka práce.

Níže na obrázku 5.3 je další seznam a ukázka použitých widgetů.



Image

- Widget zobrazující obrázek. Lze nahrát lokální případně získat ho z URL.

SingleChildScrollView

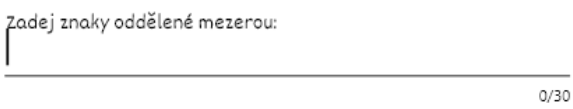
- Umožňuje potomkovi schopnost vertikálního posouvání (scrollování).

Divider

- Funguje stejně jako VerticalDivider, ale je horizontální.

Obrázek 5.3: Ukázka widgetů. Zdroj autorka práce.

Níže na obrázku 5.4 je další seznam a ukázka widgetu TextField.



Obrázek 5.4: Ukázka widgetu TextField.
Zdroj: autorka práce.

TextField

- Widget umožňující uživateli zadat text. Pomocí vlastnosti **onChanged** se při každé změně pole zavolá přidělená funkce. Pomocí **controlleru** se může se zadaným textem pracovat přímo.

Níže na obrázku 5.5 je další seznam a ukázka widgetu TextField.

Obrázek 5.5: Ukázka widgetu SnackBar.
Zdroj: autorka práce.

SnackBar

- Widget reprezentující okno vyskakující ze spodu obrazovky viditelné na pár vteřin (délka se ovládá vlastností **duration**).

MaterialApp

- Slouží jako kořen stromu aplikace. Vlastnost **theme** určuje barevné schéma a **home** přesměrovává na úvodní stránku aplikace.

LayoutBuilder

- Poskytuje možnost dynamicky přizpůsobovat rozložení podle velikosti rodičovského widgetu.

Expanded

- Widget používající se k rozšíření dětského widgetu v rámci flexibilního rodiče jako například Row nebo Column, aby se zabral celý dostupný prostor.

FittedBox

- Widget, který automaticky škáluje svého potomka podle definovaných omezení pomocí atributu **fit**.

AutoSizeText

- Widget nabízený v balíčku **auto_size_text**, který automaticky škáluje text podle hranic rodiče.

Align

- Widget, který zarovnává potomka. Zarovnání se definuje vlastností **alignment**.

Center

- Widget, který zarovnává potomka na střed.

GestureDetector

- Widget, který rozpoznává gesta u potomka, například kliknutí, podržení, vypuštění tlačítka atd.

Podrobnější vysvětlení, použití a spoustu dalších widgetů lze najít ve Flutter dokumentaci na stránkách https://docs.flutter.dev.

5.4. Logika a kód

V této kapitole se představují důležité části kódu s komentáři. Úroveň důležitosti je subjektivní, ale jelikož se mnoho částí fundamentálně opakovalo, nejsou pro čtenáře tolik zajímavé. Proto bylo po domluvě s vedoucím práce rozhodnuto nekomentovat celý kód, ale pouze některé části dle výběru autorky.

5.4.1. Start aplikace

```dart
void main()  {
  //setting minimum and maximum size on desktop
  WidgetsFlutterBinding.ensureInitialized();
  windowManager.ensureInitialized();
  if (Platform.isWindows || Platform.isMacOS || Platform.isLinux) {
    WindowManager.instance.setMinimumSize(const Size(500, 800));
    WindowManager.instance.setMaximumSize(const Size(1200, 1000));
  }
  //start Flutter app
  runApp(
    ChangeNotifierProvider(
      create: (context) => ThemeProvider(),
      child: MyApp()
    )  // ChangeNotifierProvider
  );
}
```

- Dart kód se spustí díky metodě **main()**, ve které se nastavuje maximální a minimální velikost okna na desktopových aplikací pomocí **WindowManageru**.
- Samotná Flutter aplikace se spouští až v metodě **runApp()**.
- Při startu aplikace se vytvoří instance třídy **ThemeProvider()**, která má na starost denní/noční režim aplikace.

5.4.2. Noční režim

```dart
//colour scheme for dark mode
final darkTheme = ThemeData(
  colorScheme: ColorScheme.dark(
    background: Colors.black,
    primary: Colors.white,
    secondary: Colors.grey.shade300,
  ),  // ColorScheme.dark
  brightness: Brightness.dark,
  iconTheme: IconThemeData(color: Colors.white),
);  // ThemeData
```

- Barevné téma se definuje pomocí poskytované třídy **ThemeData()**.
- V denním režimu jsou barvy opačné – pozadí bílá, primární a ikonky černá a sekundární je tmavě šedivá. Denní režim je deklarován stejným způsobem.

```dart
//class that handles themes
class ThemeProvider with ChangeNotifier {
  //sets default theme based off theme of the device
  ThemeData _themeData = PlatformDispatcher.instance.platformBrightness == Brightness.dark ? darkTheme : lightTheme;
  ThemeData get themeData => _themeData;

  set themeData(ThemeData themeData) {
    _themeData = themeData;
    notifyListeners();
  }

  void toggleTheme() {
    if (isLight()) themeData = darkTheme;
    else themeData = lightTheme;
  }

  bool isLight() {
    if (_themeData == lightTheme) return true;
    else return false;
  }
}
```

- Třída starající se o změnu denního a nočního režimu – implicitně se nastaví podle režimu zařízení pomocí vlastnosti **plaformBrightness.**
- Metoda **notifyListeners()** pochází z třídy **ChangeNotifier()** a upozorňuje všechny posluchače v kódu o změnu nastavení _**themeData**. To se stane i v metodě **toggleTheme()** při změně režimu.

5.4.3. Navigace na domovské stránce

```dart
//widget that creates the grid that holds individual menu cards and redirects user when tapped
Widget makeMenuGrid(BuildContext context) {
  return GridView.count(
    physics: NeverScrollableScrollPhysics(), // Each box in the view is not scrollable
    shrinkWrap: true, // Wrap the GridView with fixed height
    crossAxisCount: (MediaQuery.of(context).size.width / 180).floor(), //maximum width for the gridview
    padding: EdgeInsets.all(10.0),
    childAspectRatio: 0.8,
    children: [
      buildMenuButton(context, () { Navigator.push(context, MaterialPageRoute(builder: (context) => AlphabetPage()),); },
          "alphabet", "Abeceda", alphabetTip, alphabetButtonColor),
      buildMenuButton(context, () { Navigator.push(context, MaterialPageRoute(builder: (context) => MorsePage()),); },
          "morse_code", "Morseovka", morseTip, morseButtonColor),
      buildMenuButton(context, () { Navigator.push(context, MaterialPageRoute(builder: (context) => BraillePage()),); },
          "braille", "Braillovo písmo", brailleTip, brailleButtonColor),
      buildMenuButton(context, () {Navigator.push(context, MaterialPageRoute(builder: (context) => SemaphorePage()),);},
          "semaphore", "Semaforová abeceda", semaphoreTip, semaphoreButtonColor),
      buildMenuButton(context, () {Navigator.push(context, MaterialPageRoute(builder: (context) => PolishCrossPage()),);},
          "polish_cross", "Polský kříž", crossTip, crossButtonColor),
      buildMenuButton(context, () {Navigator.push(context, MaterialPageRoute(builder: (context) => BinaryPage()),);},
          "binary", "Binárka", binaryTip, binaryButtonColor),
    ],
  );  // GridView.count
}
```

- Metoda **makeMenuGrid()** vytváří rozložení karet, které jsou individuálně stavěny v metodě **buildMenuButton()**.
- Funkce **Navigator.push(context, MaterialPageRoute(builder : (context) =>))** přesměrovává uživatele po kliknutí na určitou kartu na danou stránku dekodéru.

5.4.4. Dekódování morseovky

```
//decode takes into account czech letter Ch, but it had to be done separately, since
// moreMorseCode.decode does not know that letter
void decodeMorse() {
    if (_input.isNotEmpty) {
      setState(() {
        if (_input == " . . . .") {
          _solution = 'H';
          _reverse = 'Ch';
        }
        else if (_input == " _ _ _ _") {
          _solution = 'Ch';
          _reverse = 'H';
        }
        else {
          MorseCode meroMorseCode = MorseCode();
          String tmp = _input.replaceAll(' ', '');
          try {
            _solution = meroMorseCode.deCode(tmp.replaceAll('_', '-'));
          } catch (e) {
            _solution = '';
          }
          try {
            _reverse = meroMorseCode.deCode(tmp.replaceAll('.', '-').replaceAll('_', '.'));
          } catch (e) {
            _reverse = '';
          }
        }
      });
    }
}
```

- K dekódování vstupu (**_input**) se využívá funkce **deCode(String)**.
- Jelikož ta metoda nezná české písmenko „CH," se kterým se v dekodéru počítá, je třeba nejdříve zjistit, zda nebyl vstup rovný „CH" nebo „H" (to tvoří opačné řešení „CH").
- Opačné řešení totiž spočívá v tom vyměnit tečky a čárky, ne pořadí znaků ve vstupu.
- Do vstupu se čárky reprezentují podtržítkem (_) kvůli stejné výšce, jako má tečka (.), ale metoda **deCode()** vyžaduje čárku jako spojovník (-) , proto se nejdřív musí ve vstupu všechny podtržítka nahradit metodou **replaceAll('_', '-')**.
- Funkce **setState()** zaručí, že se změny propíšou na stránku (přerenderuje se).

5.4.5. Dekódování binární soustavy

```
String reverseBinary(String str) {
  return str.replaceAll('0', '2').replaceAll('1', '0').replaceAll('2', '1');
}

void decodeBinary() {
  setState(() {
    if (_input.isNotEmpty) {
      _solution = int.parse(_input, radix: 2).toString();
      _reverse = int.parse(reverseBinary(_input), radix: 2).toString();
    }
  });
}
```

- Funkce **int.parse(_input, radix: 2)**, která převádí řetězec String na celé číslo (integer).
- Specifikace radix: 2 určuje, že se jedná o řetězec v binární soustavě (základ 2).
- Opačná forma se tvoří záměnou jedniček a nul v metodě **reverseBinary().**

5.4.6. Dekódování abecedy

```
//index starts at 1
void decodeNumber(String input) {
  int inputValue = int.tryParse(input) ?? 0;
  if (inputValue > 0) {
    String letter = String.fromCharCode((inputValue - 1) % 26 + 'A'.codeUnitAt(0));
    setState(() {
      _solution += " " + letter;
    });
  }
}

void decodeLetter(String input) {
  if (input.length == 1 && isAlpha(input.toUpperCase())) {
    int result = (input.toUpperCase().codeUnitAt(0) - 'A'.codeUnitAt(0)) % 26 + 1;
    setState(() {
      _solution += " " + result.toString();
    });
  }
}

//looks at a character and decides if it's a number or letter and redirects to a specialised decoder
void decodeChar(String input) {
  if (input.isNotEmpty) {
    if (RegExp(r'^[a-zA-Z]$').hasMatch(input)) {
      // Input is a single letter
      _inputValue += input.toUpperCase();
      decodeLetter(input);
    } else if (RegExp(r'^\d+$').hasMatch(input)) {
      // Input is a number
      _inputValue += input;
      decodeNumber(input);
    }
  }
}
```

```
//takes the input and splits it into list based of spaces (dividing character) and then decodes char separately
void processInput(String value) {
  setState(() {
    if (value.isNotEmpty) {
      List<String> inputs = value.trim().split(' ');
      _inputValue = '';
      _solution = '';
      for (String input in inputs) {
        decodeChar(input);
      }
    }
  });
}
```

- V dekodéru abecedy se dekóduje víc vstupů najednou a můžou to být čísla i písmena dohromady, pokud jsou odděleny mezerou. V metodě **processInput(String)** se vstup rozdělí do pole jednotlivých substringů.
- Substrngy se pak posílaj dometody **decodeChar(String)**, která díky regulárnímu výrazu pozná, zda se jedná o číslo nebo písmeno. Podle toho se přesměruje na konkrétní dešifrování.
- Jelikož indexujeme od 1 (A = 1), musí se s tím počítat při modulárním výpočtu v **decodeLetter(String)** i v **decodeNumber(String).**

5.4.7. Dekódování semaforové abecedy

```
List<bool> _isSelected = List.generate(8, (index) => false);
List<List<String>> _semaphoreTable = List.generate(8, (index) => List<String>.filled(8, ''));
```

- List **_isSelected** drží informaci o statusu každého z 8 kruhů.
- 2D List **_semaphoreTable** drží tabulkové rozložení písmenek (viz obrázek 2.13)

```
void makeSemaphoreTableCol(int row, int startIdx, List<String> letters) {
  _semaphoreTable[row].setAll(startIdx, letters);
}

makeSemaphoreTableCol(0, 1, ['A', 'B', 'C', 'D', 'E', 'F', 'G']);
makeSemaphoreTableCol(1, 2, ['H', 'I', 'K', 'L', 'M', 'N']);
makeSemaphoreTableCol(2, 3, ['O', 'P', 'Q', 'R', 'S']);
makeSemaphoreTableCol(3, 4, ['T', 'U', 'Y']);
makeSemaphoreTableCol(4, 6, ['J', 'V']);
makeSemaphoreTableCol(5, 6, ['W', 'X']);
makeSemaphoreTableCol(6, 7, ['Z']);
```

- Tabulka je souměrná po diagonále, stačí ji tedy naplnit pouze polovičně.

```
//only half of the table is filled, so if the coords lead to empty space, they swap and try to find solution
void decodeSemaphore() {
  int first = -1, second = -1;
  for (int i = 0; i < _isSelected.length; i++) {
    if (_isSelected[i] == true) {
      first == -1 ? first = i : second = i;
    }
  }
  if (_semaphoreTable[first][second] == '') {
    _solution = _semaphoreTable[second][first];
  }
  _solution = _semaphoreTable[first][second];
}

//handles toggles, only two circles can be selected at the same time
void togglePart(int index) {
  setState(() {
    if (_numSelected < 2 && _isSelected[index] == false) {
      _isSelected[index] = !_isSelected[index];
      _numSelected++;
      if (_numSelected == 2) decodeSemaphore();
    } else if (_isSelected[index] == true) {
      _isSelected[index] = !_isSelected[index];
      _numSelected--;
      _solution = "";
    }
  });
}
```

- Zmáčknutím individuálního kroužku se zavolá metoda **togglePart()**, která kontroluje, zda již není konkrétní kroužek vybrán a pokud ne, zda nejsou dva jiné vybrány (dva je maximum).
- Při dekódování se zjistí, která dva kroužky jsou vybrán a podle jejich indexu se hledá v tabulce semaforové abecedy. Indexy totiž reprezentují řádky/sloupce 2D pole **_semaphoreTable**.
- Jelikož je tabulka zaplněná pouze z jedné půlky, nemusí se hned vždy vrátit výsledek. Pokud je tomu tak, vyměněná čísla řádku a sloupce řešení najdou.

5.4.8. Dekódování Braillova písma

```
List<bool> _isSelected = List.generate(6, (index) => false);
List<List<String>> brailleTable = List.generate(8, (index) => List<String>.filled(8, ''));
```

- List **_isSelected** drží informaci o statusu každého z 6 kruhů.
- 2D List **brailleTable** drží tabulkové rozložení písmenek (viz obrázek 2.10)

```
void makeBrailleTableRow(int row, List <String> chars) {
  for (int i = 0; i < chars.length; i++) {
    brailleTable[row][i] = chars[i];
  }
}

makeBrailleTableRow(1, ['A', 'C', 'E', 'D', 'Á', 'Č', 'Š', 'Ď']);
makeBrailleTableRow(2, [',', 'I', ':', 'J','?', 'Ó', '+', 'Ř']);
makeBrailleTableRow(3, ['B', 'F', 'H', 'G', 'Ě', 'Ň', 'Ť', '/']);
makeBrailleTableRow(4, ['.', 'Í', '*', 'É', '-', 'Ú', ')', '']);
makeBrailleTableRow(5, ['K', 'M', 'O', 'N', 'U', 'X', 'Z', 'Y']);
makeBrailleTableRow(6, [';', 'S', '!', 'T', '(', 'Ž', '"', 'Ů']);
makeBrailleTableRow(7, ['L', 'P', 'R', 'Q', 'V', 'Ý', 'W', '']);

//table is set up in a binary fashion -> counting all true and false gives us a binary code which
// we can decipher and know the row/column number
void decodeBraille() {
  setState(() {
    int row = 0, col = 0;
    String rowBin = '', colBin = '';
    for (int i  = 0; i < _isSelected.length; i++) {
      if (i % 2 == 0) {
        int a = _isSelected[i] ? 1 : 0;
        rowBin = a.toString() + rowBin;
      }
      else if (i % 2 == 1) {
        int a = _isSelected[i] ? 1 : 0;
        colBin = a.toString() + colBin;
      }
    }
    row = int.parse(rowBin, radix: 2);
    col = int.parse(colBin, radix: 2);
    _solution = brailleTable[row][col];

    row = int.parse(reverseBinary(rowBin), radix: 2);
    col = int.parse(reverseBinary(colBin), radix: 2);
    _reverse = brailleTable[row][col];
  });
}
```

- **brailleTable** funguje binárně – když zjistíme, jaké všechny kroužky jsou ne/vybrané, získáme tím číslo v binární soustavě pro sloupec (lichý index) a pro řádek (sudý index).
- Dekódováním daného binárního čísla zjistíme pořadí sloupce a řádku, což vrátí jednoznačně výsledek.
- Opačný výsledek funguje způsobem, že se statusy vymění, neboli stačí vyměnit jedničky a nuly v rámci binárního čísla a spočítat nový řádek a sloupec, což vrátí opačný jednoznačný výsledek.

5.4.9. Dekódování polského kříže

```
int _numSelected = 0, _sideSelected = 0, _groundSelected = 0, _previousIndex = -1;
```

- U polského kříže je nutné, aby byla vždy vybrána přesně jedna pozice (**_numSelected**), alespoň jedna boční strana (**_sideSelected**) a jedna spodní/horní strana (**_groundSelected**)

```
List<bool> _isSelected = List.generate(7, (index) => false);
List<List<List<String>>> _crossTable = [[['A', 'B', 'C'], ['D', 'E', 'F'], ['G', 'H', 'Ch']], [['I', 'J', 'K'],
  ['L', 'M', 'N'], ['O', 'P', 'Q']], [['R', 'S', 'T'], ['U', 'V', 'W'], ['X', 'Y', 'Z']]];
```

- List **_isSelected** drží informaci o statusu každé ze 7 částí.
- 3D List **_crossTable** drží tabulkové rozložení písmenek (viz obrázek 2.14)

```
//finds out which of the three positions is selected and which letter is the solution
void decodeCrossValue(int row, int col) {
  if (_isSelected[2]) {
    _solution = _crossTable[row][col][0];
  }
  else if (_isSelected[3]) {
    _solution = _crossTable[row][col][1];
  }
  else if (_isSelected[4]) {
    _solution = _crossTable[row][col][2];
  }
}

//finds which row/column in the 3x3 grid is active
int decodeCrossCell(bool first, bool second) {
  if (first) {
    if (second) {
      return 1;
    }
    else return 2;
  }
  return 0;
}
```

```
//decodes polish cross -> at least one side and one ground/ceiling has to be active
// and exactly one circle position in the middle has to be active
void decodeCross() {
  setState(() {
    if (_numSelected != 1 || _sideSelected < 1 || _groundSelected < 1) {
      _solution = "";
    }
    else {
      int row = decodeCrossCell(_isSelected[0], _isSelected[6]);    //finds out in which row is the active cell
      int col = decodeCrossCell(_isSelected[1], _isSelected[5]);    //finds ou in which column is the active cell
      decodeCrossValue(row, col);   //finds out which is the active position in a concrete cell
    }
  });
}
```

- Metoda **decodeCross()** nejprve zjistí, zda je vstup v pozici, že by mohl existovat výsledek.
- Pomocí **decodeCrossCell(bool, bool)** se zjistí, které stěna (horní, dolní, případně obě) je aktivní. Tím se zjistí řádek **_crossTable.** To samé se pak hledá i pro boční stěny, které naleznou sloupec. Tím se najde aktivní buňka.
- **decodeCrossValue(int, int)** hledá, která ze 3 pozic v buňce je aktivní. Kombinací buňka + pozice se pak najde jednoznačný výsledek.

5.4.10. Nastavení orientace obrazovky

```dart
//sets up portrait (vertical) orientation on all devices
void setOrientation() {
  // Set preferred orientation to portrait
  SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp,
    DeviceOrientation.portraitDown,
  ]);
}
```

- Je žádoucí, aby aplikace byla v portrétním (vertikálním) módu na všech zařízeních, jelikož je na to vypracovaný design.

5.4.11. Kopírování textu do schránky

```dart
//creates snackbar that tell us we copied text into clipboard
SnackBar makeSnackbar(String value, Color snackColor) {
  return SnackBar(
    content: Text('Zkopírováno do schránky: $value', style: TextStyle(color: Colors.black, fontFamily: "Figtree")),
    duration: Duration(milliseconds: 800),
    backgroundColor: snackColor,
  );  // SnackBar
}
```

```dart
Expanded(
  child: GestureDetector(
    onTap: () {
      Clipboard.setData(ClipboardData(text: result));
      ScaffoldMessenger.of(context).showSnackBar(
      makeSnackbar(result, snackColor),
      );
    },
    child: AutoSizeText(result, style: TextStyle(fontWeight: FontWeight.w800, fontSize: 2.5 * fontSize, fontFamily: "Dekko"),),
  ),  // GestureDetector
),  // Expanded
```

- Při zmáčknutí na widget **Text**, který je obalem **GestureDetectorem**, se zavolá funkce **Clipborad.setData()** a zkopíruje se daný text do schránky. Funguje na každém zařízení.

5.5. Shrnutí kapitoly implementace

Tato kapitola se zaměřila na implementaci aplikace. Nejprve se porovnávaly výhody a nevýhody dvou přístupů vývoje: nativního a cross-platform. Poté se představila vybraná cross-platform technologie Flutter a její klíčové prvky pro tvorbu front-endu jménem widgety. Na závěr byly uvedeny důležité části kódu s komentáři, jak jednotlivé útržky metod fungovaly.

6. Testování aplikace

Testování software je klíčovým prvkem ve vývoji kvalitního software. Pomáhá nejen identifikovat a opravit chyby, ale také zajišťuje, že software je bezpečný a efektivní. [33] Tato kapitola se věnuje zpětné vazbě ohledně dostupných funkcí a vzhledu. Aplikace byla otestována ve dvou rovinách – automaticky pomocí naprogramovaných testů a uživatelsky, kde se jednalo převážně o „lidský přístup“. Aplikace by měla být přívětivá, proto jsou názory uživatelů klíčovým krokem k dokončení projektu.

6.1. Unit testy

Automatizované testování proběhlo takzvanými unit testy neboli testováním samostatných jednotek softwaru. Jednotky bývají především jednotlivé metody, kde se testuje, zdali fungují správně. [34]

V aplikaci se testovala především správnost dekódování šifer a správnost funkcí napojených na tlačítka. Samotný proces klikání a testování, zda se zavolala správná metoda v tomto přístupu provádět nelze, to se lépe zjišťuje uživatelsky.

6.1.1. Struktura

Testy byly rozděleny do 7 souborů, podle toho, jakou třídu zkoušely (6 dekodérů + ThemeProvider) a v rámci souboru do skupin dle cíle testu. Tyto skupiny jsou tři: testy vstupu, dekódování řešení a manipulace s uloženými výsledky. ThemeProvider se tomu vymyká, tam se zkouší správnost změny nočního a denního režimu.

Celkově bylo napsáno 50 unit testů, jež pokryly všechny interakce systému s uživatelem. Testy jsou napsány trojfázovým způsobem „SETUP, DO, TEST,“ kde se v první části připravují nutné prostředky k testování, následně proběhnou metody programu a na závěr se výsledky porovnávají s očekávanou hodnotou.

6.1.2. Příklad z kódu

```
class TestHelper {
  static Future<BraillePageState> setup(WidgetTester tester) async {
    await tester.pumpWidget(MaterialApp(home: BraillePage()));
    return tester.state<BraillePageState>(find.byType(BraillePage));
  }
}
```

- Metoda **setup()** určuje podmínky testovacího prostředí
- Najde se widget typu **BraillePage()** a vrací jeho state, neboli **BrailleStatePage**, tím se určuje, jaká třída se bude testovat, v tomto případě dekodér Braillova písma.

```
group("saved result tests", () {
  testWidgets("Test saving result", (WidgetTester tester) async {
    // SETUP
    final braillePageState = await TestHelper.setup(tester);
    //DO
    braillePageState.togglePart(0);
    braillePageState.togglePart(3);
    braillePageState.decodeBraille();
    braillePageState.addToSaved();
    //TEST
    expect(braillePageState.savedSolution, "E");
    expect(braillePageState.savedReverse, "Ž");
  });
```

- Testování, zda se řešení přidá do uloženého řetězce.
- Zmáčknutá tlačítka určují výsledek, který se dekóduje (správnost metod **togglePart(int)** a **decodeBraille()** byla ověřena v předchozích testech.
- Po provedení metody **addToSaved()** se testuje, zda řetězce **savedSolution** a **savedReverse** nejsou prázdné a jaký mají v sobě znak.

6.1.3. Shrnutí a výsledky implementačních testů

Testování odhalilo 2 identické chyby v kódu, a to ve funkci mazání posledního přidaného uloženého výsledku u dekodéru abecedy a binární soustavy.

```
testWidgets('Test delete last saved', (WidgetTester tester) async {
  // SETUP
  final alphabetPageState = await TestHelper.setup(tester);
  // DO
  alphabetPageState.processInput("1 A 17 m PS");
  alphabetPageState.addToSaved();
  alphabetPageState.addToSaved();
  alphabetPageState.clearLastSaved();
  //TEST
  expect(alphabetPageState.savedSolution, " A 1 Q 13");
});
```

- Výsledky u abecedy se můžou skládat z vícemístných číslic, proto jsou všechny mezivýsledky odděleny mezerou pro přehlednost.
- Metoda **addToSaved()** vezme celý řetězec výsledků a ten uloží. Druhým zavoláním se prodlouží o další řetězec.
- Metoda **clearLastSaved()** nesmaže poslední znak, nýbrž smaže poslední uložený řetězec.
- Chyba se vyskytovala v přidávání dalšího řetězce, jelikož v prvním případě se mezi oba přidávala mezera navíc, která se však nemazala a zůstávala tam.
- U opakovaného mazání to vedlo k chybným výsledkům, jelikož byly indexy posunuty o nadbytečné mezery.

Opravou tohoto problému se povedlo mít bezchybné interakce, jelikož všechny ostatní metody fungovaly správně.

6.2. Uživatelské testování

Kromě automatizovaných testů bylo třeba program vyzkoušet vybranými lidmi, zda jim aplikace funguje a jaký mají názor na její použitelnost a vizuál. Na jeden test stačí získat 5 zkoušejících, kteří statisticky dokážou najít 70-80 % chyb. [35]

6.2.1. Způsob testování

Konvenčně je pro testery připraven nějaký scénář, kterým si mají projít, jelikož otestuje většinu (nebo všechny) nabízených funkcí. Toto však nebyl přístup, kterým jsem chtěla zjistit účinnost aplikace. Chtěla jsem, aby aplikace byla intuitivní pro kohokoliv, neboli i pro nováčky. Navádění respondentů správným směrem by nebylo proto efektivní. Aplikace zároveň není tak rozsáhlá, aby si testeři nevšimli nějakých funkcí, které by nevyzkoušeli. Byly osloveny dvě skupiny vybraných lidí – lidé se zkušenostmi se šifrovačkami (3 osoby) a úplní nováčci (6 osob). Diverzita zkoušejících pokryla všechny potřebné aspekty. Zpětná vazba byla sbírána hned na místě, jelikož aplikace byla zkoušena pod mým dohledem a díky tomu jsem mohla vnímat i neverbální reakce testerů.

6.2.2. První iterace testů

Testování proběhlo ve více fázích – po každé fázi byly provedeny změny, které byly vyzkoušeny dále až do fáze, kdy jsem byla spokojena. Do výčtového seznamu níže jsou vypsány časté připomínky testovacích skupin a moje vlastní pozorování.

6.2.2.1. Zpětná vazba po první iteraci

Veteráni

- Chyběl jim tmavý režim.
- Design aplikace je hezký a přehledný pro potřeby šifrovaček.
- Přepínání mezi způsobem zadávání u abecedy (písmena x číslice) jim připadalo „otravné“.

Nováčci

- Neklikali na symbol nápovědy.
- Nepochopili princip některých šifer a způsob zadávání i po přečtení nápověd.
- Nepochopili princip opačného výsledku.
- Chybělo jim přepínání do jiných jazyků.

Obě skupiny

- Nefungovalo škálování – na malých telefonech přetekly části obrazovky.
- Manuálně mačkat tlačítko „dekódovat“ bylo uživatelsky nepříjemné.
- Po uložení se řešení automaticky nenulovalo.

6.2.2.2. Stav po první iteraci

Provedené změny

- Zdokonalilo se škálování, aby se velikost widgetů odvíjela od velikosti zařízení.
- Zavedlo se dynamické dekódování a zrušilo se tlačítko „Dekódovat“.
- Vstup abecedy byl změněn. Nyní může být zadáno písmeno nebo číslo a víc vstupů najednou, aplikace sama pozná, jaký dekodér má zavolat.
- Přepsal se text nápověd pro detailnější vysvětlení šifer, způsobu zadávání i významu opačných řešení.

Neprovedené změny

- Řešení se automaticky nenuluje záměrně – pro případ opakujícího se uložení.
- Není v plánu aplikaci rozšiřovat do dalších jazyků, jelikož je určená pro českou a slovenskou komunitu.

6.2.3. Druhá iterace testů

Druhé testování proběhlo po zakomponování zpětné vazby z předchozí iterace testů. Do výčtového seznamu níže jsou vypsány časté připomínky testovacích skupin a moje vlastní pozorování.

6.2.3.1. Zpětná vazba po druhé iteraci

Veteráni

- Ocenili by zadávání více písmenek morseovky najednou.
- Manuální přepínání pozice u polského kříže jim připadalo „otravné“.

Nováčci

- Neklikali na symbol nápovědy.
- Z důvodu dlouhého textu nápovědy ztratili zájem ji pročítat.

Obě skupiny

- Chybělo jim mazání posledního přidaného znaku vstupu u morseovky a binární soustavy.
- Chybělo jim mazání posledního uloženého řešení.
- Nelíbilo se jim písmo (Charm) u jména dekodéru na horní liště stránek.

6.2.3.2. Stav po druhé iteraci

Provedené změny

- Přidání obrázkových nápověd k potřebným vysvětlivkám pro lepší čitelnost nápověd.
- U polského kříže, pokud je již vybraná jiná pozice, původní pozice se resetuje a vybere se pouze nová.
- Přidání mazání posledního vstupu i uložených výsledků.
- Změna písma jména dekodéru v horní liště na písmo Figtree.

Neprovedené změny

- Není možnosti zadávat více písmenek v rámci jednoho vstupu morseovky, jelikož by se na menších obrazovkách vstup nevešel a přeteklo by to. Zmenšené by to nebylo dobře k přečtení.

6.2.4. Návrhy do budoucna

Po druhé iteraci byla položená testerům otázka, co dalšího by si od aplikace představovali.

Veteráni

- Více šifer – přidání převodníku dalších číslicových soustav a posunu (Césarovy šifry).

Nováčci

- Reverzní zadávání – například místo znaků morseovky zadat písmenko a kombinace teček a čárek by bylo řešení.
- Možnost vyfotit papír, aby se vstupy samy automaticky načetly.

6.3. Shrnutí kapitoly testů

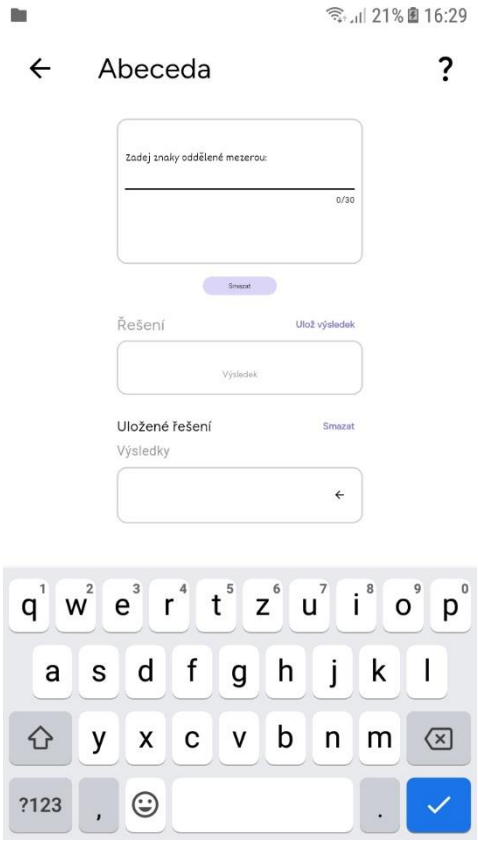
V této kapitole byla provedena důkladná kontrola funkčnosti vyvinuté aplikace. Testovací proces zahrnoval jak automatizované unit testy, tak uživatelské testování. Automatizované unit testy byly použity k ověření správnosti jednotlivých funkcí aplikace. Tyto testy se zaměřovaly na kontrolu, zda každá část kódu funguje a vrací očekávané výsledky. Následovalo uživatelské testování, které se zaměřovalo na několik klíčových aspektů: design aplikace a uživatelskou přívětivost. Testeři poskytovali zpětnou vazbu ohledně vzhledu a uspořádání aplikace, hodnotili estetiku, intuitivnost, celkový dojem a přátelské použití.

7. Vyhodnocení práce

7.1. Aktuální stav aplikace

Aktuální verze aplikace funguje bez problému, podle mě má však 2 nedostatky. Prvním nedostatkem je poloha šipek, které odstraňují poslední uložené (i opačné) řešení (viz obrázek 7.1 níže). Poloha těchto šipek je na pravé části *Containeru* (viz *Použité widgety*) s uloženým (opačným) řešením, bohužel však na malých obrazovkách ujíždí mimo své místo. Druhý nedostatek je při kliknutí na zadávání vstupu u dekodéru abecedy – zobrazí se systémová klávesnice a celá stránka se zmenší, což vede k horší čitelnosti (viz obrázek 7.2).

Obrázek 7.1 Screenshot finální aplikace. Zdroj: autorka práce.



Obrázek 7.2 Screenshot aplikace. Zdroj: autorka práce.

7.2. Budoucí stav

Na vývoji aplikace určitě budu pracovat i nadále. Je to nástroj, který jsem již dlouho postrádala. Kromě vyřešení dvou předchozích nedostatků lze aplikaci vylepšit přidáním několika dalších užitečných dekodérů, například Césarovu šifru (posun), 7segmentový display, T9 klávesnice (klávesnice na tlačítkových telefonech) a dalších.

Funkce focení papíru pro automatické načítání vstupu je vylepšením, které by vyžadovalo znalost strojového vidění a rozpoznávání textu. To by mohlo být potenciálně téma pro diplomovou práci do budoucna.

8. Závěr práce

Cílem této bakalářské práce bylo vytvořit mobilní aplikaci, která by účastníkům logických her zvaných „šifrovačka“ zjednodušila manuální práci při dešifrování úkolů. Na základě podrobné analýzy šifrovacích soutěží, šifer a mých osobních zkušeností jsem navrhla a implementovala aplikaci, která poskytuje intuitivní a uživatelsky přívětivé prostředí. Aplikace byla vyvinuta ve frameworku Flutter, což umožňuje její nasazení na platformách Android, iOS a také jako desktopovou aplikaci.

Práce vychází z již existujících metod podpory pro řešení šifer, zejména z populárních šifrovacích tabulek. Tyto tabulky, které jsou široce používány účastníky šifrovacích soutěží, poskytly základní strukturu a funkcionalitu, kterou aplikace dále rozvíjí a zlepšuje. Cílem bylo nejen digitalizovat tyto tabulky, ale také přidat další užitečné funkce, které zrychlí a zjednoduší proces dešifrování.

Práce začala definicí klíčových pojmů spojených se šifrovačkami a analýzou potřeb uživatelů. Na základě těchto poznatků byla vytvořena aplikace, která splňuje definované funkční a nefunkční požadavky. Byla několikrát testována jak automatizovanými unit testy, tak uživateli. Testování prokázalo, že aplikace funguje správně a je uživatelsky přívětivá.

Všechny cíle stanovené na začátku práce byly úspěšně splněny. Aplikace nabízí účinný nástroj pro účastníky šifrovacích soutěží, který zrychluje a usnadňuje dešifrování úkolů.

8.1. Osobní reflexe

Během vypracování této bakalářské práce jsem získala cenné zkušenosti a dovednosti, které mě v programování obohatily, jelikož to byl můj první vývoj mobilní aplikace. Díky ní jsem si prohloubila znalosti o metodách cross-platform programování, o kterých jsem dříve jen něco málo slyšela. Naučila jsem se sama programovat v novém jazyce pouze s pomocí série videí *Widget of the Week*[9]a nástroje ChatGPT, kterého jsem se mohla ptát na programovací konvence Flutteru. Aplikaci se mi povedlo vyrobit přesně podle mých představ a budu ji na soutěžích využívat. A doufám, že bude sloužit i dalším zájemcům.

[9] Playlist s videi na adrese:
https://www.youtube.com/watch?v=PPwJ75vqP_s&list=PLjxrf2q8roU23XGwz3Km7sQZFTdB996iG

Zdroje

[1] Kódování textu a šifrování. *Umimeinformatiku.cz* [online]. [cit. 2024-05-18]. Dostupné z: https://www.umimeinformatiku.cz/book/cviceni-sifry#parent-kc-72

[2] *Rovnice úspěchu: Odbornost, soft skills a sebevědomí* [online]. 2015 [cit. 2024-01-14]. Dostupné z: https://www.topvision.cz/blog/rovnice-uspechu-odbornost-soft-skills-a-sebevedomi/?nezajem=1

[3] SLAVĚTÍNSKÝ, Radek. *Schémata symetrického šifrování*. Praha, 2014. Bakalářská práce. Bankovní institut vysoká škola Praha, Katedra informatiky a kvantitativních metod.

[4] SOUKUPOVÁ, Jana. *Softwarové zabezpečení dat*. Jindřichův Hradec, 2007. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta managementu v Jindřichově Hradci.

[5] *Šifrování dat*. Online. 2022. Dostupné z: https://www.sprava-site.eu/sifrovani-dat/. [cit. 2024-01-10].

[6] LAUŠ, Jan. *Moderní kryptoanalytické metody*. Brno, 2007. Diplomová práce. Masarykova Univerzita, Fakulta informatiky.

[7] VONDRUŠKA, Pavel. *Kryptologie, šifrování a tajná písma*. Praha: Albatros, 2006. ISBN 80-00-01888-8.

[8] KOCIÁNOVÁ, Bc. Helena. *Digitální steganografie*. České Budějovice, 2009. Diplomová práce. Jihočeská univerzita v Českých Budějovicích, Pedagogická fakulta.

[9] Řešení 8 – Značkostěn. In: *Kokorinskykomar.cz* [online]. 2023 [cit. 2024-01-14]. Dostupné z: https://kokorinskykomar.cz/data/2023/reseni-sifer/sifra08_znackosten_reseni.pdf

[10] Reseni. In: *Vánoční tápání 2023* [online]. 2023 [cit. 2024-01-14]. Dostupné z: https://tapani.cz/sifra/slova/reseni?file_name=

[11] BENDOVÁ, Petra. *Základy speciální pedagogiky nejen pro speciální pedagogy*. Hradec Králové: Hradec Králové: Gaudeamus, 2015. ISBN 978-80-7435-422-9.

[12] Shaman.cz | Braillovo písmo. In: *Shaman.cz* [online]. 2021 [cit. 2024-01-14]. Dostupné z: http://www.shaman.cz/img/braillovo-pismo-abeceda.gif

[13] Šifrovací pomůcky – Chlýftým. In: *Šifrovací pomůcky – Chlýftým* [online]. c2005-2024 [cit. 2024-01-14]. Dostupné z: https://chlyftym.cz/wp-content/uploads/pomucky_mini_2-1-3.pdf

[14] ROUŠOVÁ, Kateřina. *Číselné soustavy včera a dnes*. Brno, 2018. Bakalářská práce. Masarykova Univerzita, Pedagogická fakulta.

[15] *Semaforová abeceda* [online]. 2015 [cit. 2024-01-14]. Dostupné z: https://nastobavi.webnode.cz/news/semaforova-abeceda/

[16] Semaforová abeceda. In: *Dakota.skautkostelec.cz* [online]. [cit. 2024-01-14]. Dostupné z: http://dakota.skautkostelec.cz/skautska_stezka/praxe/semafor_soubory/image001.jpg

[17] Kriz.gif. In: *Velký polský kříž - 1. dívčí a 1. skautský oddíl z Litoměřic* [online]. 2024 [cit. 2024-01-14]. Dostupné z: https://1oddil-ltm.skauting.cz/wpcontent/uploads/2023/05/kriz.gif

[18] Šifrovací pomůcky – Chlýftým. In: *Šifrovací pomůcky – Chlýftým* [online]. c2005-2024 [cit. 2024-01-14]. Dostupné z: https://chlyftym.cz/wp-content/uploads/pomucky_mini_2-1-3.pdf

[19] *Manual-mini.png* [online]. [cit. 2024-01-14]. Dostupné z: https://chlyftym.cz/wp-content/uploads/manual_mini.png

[20] GOOGLE PLAY. *Šifrovací pomůcky Absolutno* [online]. [cit. 2024-05-18]. Dostupné z: https://play.google.com/store/apps/details?id=cz.absolutno.sifry&hl=cs

[21] IEEE STANDARDS BOARD. *IEEE Standard Glossary of Software Engineering Terminology*. 1990. ISBN 1-55937467-X.

[22] Nefunkční požadavky. *Pmconsulting.cz* [online]. [cit. 2024-05-18]. Dostupné z: https://www.pmconsulting.cz/slovnikovy-pojem/nefunkcni-pozadavky/

[23] URBAN, Vít. *E-learningové materiály pro výuku jazyka UML*. Brno, 2013. Bakalářská práce. Masarykova Univerzita, Fakulta informatiky.

[24] UML 2 Tutorial - Activity Diagram. *Sparxsytems.com* [online]. [cit. 2024-05-24]. Dostupné z: https://sparxsystems.com/resources/tutorials/uml2/activity-diagram.html

[25] Felix Bachmann. Volume ii: Technical concepts of component-based software engineering, 2nd edition. Technical report, CMU/SEI - Carnegie Mellon University/Software Engineering Institute, 2000.

[26] KREUZMAN, Michal. *UML modelování*. České Budějovice, 2007. Bakalářská práce. Jihočeská univerzita v Českých Budějovicích, Pedagogická fakulta.

[27] MUSIL, Bronislav. Bakalářská práce. Brno: Masarykova Univerzita, Fakulta informatiky, 2018.

[28] ŘEZÁČ, Jan. Web ostrý jako břitva. 1. vyd. Jihlava: BAROOUE PARTNERS s.r.o., 2014. ISBN: 978-80-87923-01-6

[29] Barvy. *Dobre-svetlo.cz* [online]. [cit. 2024-05-24]. Dostupné z: http://www.dobre-svetlo.cz/barvy.php

[30] Love dark mode? Here's why you may still want to avoid it. *Androidauthority.com* [online]. [cit. 2024-05-24]. Dostupné z: https://www.androidauthority.com/dark-mode-1046425/

[31] AUGUSTA, Lukáš. Šperkujte své UI designy. *Designui.cz* [online]. [cit. 2024-05-17]. Dostupné z: https://www.designui.cz/lekce/sperkujte-sve-ui-designy

[32] PŠENIČKA, Jakub. *Návrh mobilní aplikace pro rezervaci zdrojů*. Praha, 2022. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická.

[33] Co je to testování software? *Tredgate.cz* [online]. [cit. 2024-05-17]. Dostupné z: https://tredgate.cz/2023/09/30/co-je-to-testovani-software/

[34] Unit testování. *Peach-dev.cz* [online]. [cit. 2024-05-17]. Dostupné z: https://peach-dev.cz/wiki/unit-testovani/

[35] VOJÁK, Michal. Jak dělat uživatelské testování. *Designdev.cz* [online]. [cit. 2024-05-18]. Dostupné z: https://designdev.cz/jak-delat-uzivatelske-testovani

Přílohy

 statistikyŠifrovačky.xlsx	Nasbíraná data o šifrovačkách z let 2019–2023	https://1drv.ms/x/s!Ao4Ag123rJ9lm8c88l_JM3DeHD0nNA?e=sFyr6i
 grafyŠifrovačky.pbix	Soubor s grafy vycházející z nasbíraných dat	https://1drv.ms/u/s!Ao4Ag123rJ9lm8dFtAo6Tkhf5xqspg?e=dkGCS8
 aplikaceSifrovackyDiagramy.eapx	Soubor s diagramy	https://1drv.ms/u/s!Ao4Ag123rJ9lnaMmgjmX14-SdlWbww?e=Xue46h
 prototypy.fig	Soubor s prototypy stránek	https://1drv.ms/u/s!Ao4Ag123rJ9lnshVjRpmZcF28SENTg?e=cgnzFQ
 sifrovaciNastrojeCode.zip	Adresář s kódem	https://gitlab.fel.cvut.cz/foltyka1/bakalarka
 app-release.apk	Soubor s nainstalovatelnou a spustitelnou aplikací pro systém Android	https://1drv.ms/u/s!Ao4Ag123rJ9lnshW_yKZnVrphWvrxg?e=nur2Ua