

Střední průmyslová škola Třebíč

Maturitní práce

webové rozhraní pro podporu penetračního testování

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITB4

Školní rok: 2023/2024 Radim Pokorný

Zadání práce

Hlavním cílem maturitní práce je vyvinout webové uživatelské rozhraní v podobě responzivní SPA (Single-page application) pro podporu manuálního penetračního testování. Pomocí webového rozhraní bude možné provádět kódování a dekódování Base64, URL, ASCII, HTML, Hex, decimální a binární soustavy, hashování MD, SHA, LM a NT, CRC, Bcrypt a Argon2 a šifrování nebo dešifrování pomocí AES, RC, RSA a DES. Výsledná aplikace umožní také načtení nebo uložení dat ze/do souboru. Součástí řešení bude také integrace implementovaných komponent do platformy Penterep.

ABSTRAKT

Cílem tohoto projektu je podpořit penetrační testování webovou aplikací, na které se vkládá vstupní řetězec, který se může převést na jinou formu. Operace ve většině je dvoucestná, a tak je možné pracovat s výstupem jako se vstupem. Teoretická část pojednává o důležitosti penetračního testování a o tom, jak mu tento projekt může pomoci, vysvětlení metod a seznámení s technologiemi, které jsou použity. V praktické části je ukázán postup vývoje, návrh vzhledu, optimalizace kódu a náročnost testování.

KLÍČOVÁ SLOVA

Penetrační testování, kódování, šifrování, hashování, Vue.js, PrimeVue, TypeScript, SCSS

ABSTRACT

The project aims to support penetration testing with a web application that embeds an input string that can be converted to another form. The operation in most cases is two-way, so it is possible to treat the output as input. The theoretical part discusses the importance of penetration testing and how this project can help it, explaining the methods and introducing the technologies used. The practical part shows the development process, the design of the layout, the optimization of the code, and the difficulty of testing.

KEYWORDS

Penetration testing, encoding, encryption, hashing, Vue.js, PrimeVue, TypeScript, SCSS

PODĚKOVÁNÍ

Děkuji Bc. Willimu Lazarovi za cenné připomínky a rady, které mi poskytl při vypracování maturitní práce.

V Třebíči dne 10. března 2024 podpis autora

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně a uvedl/a v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil/a.

V Třebíči dne 10. března 2024

podpis autora

Obsah

[Úvod 7](#_Toc160995104)

[1 Penetrační testování 8](#_Toc160995105)

[2 Visual Studio Code 9](#_Toc160995106)

[3 TypeScript 10](#_Toc160995107)

[4 SCSS 11](#_Toc160995108)

[5 Vue.js 12](#_Toc160995109)

[6 PrimeVue 13](#_Toc160995110)

[7 GitHub 14](#_Toc160995111)

[8 Kódovací metody 15](#_Toc160995112)

[8.1 Base64 15](#_Toc160995113)

[8.2 URL 16](#_Toc160995114)

[8.3 ASCII 17](#_Toc160995115)

[8.4 HTML 18](#_Toc160995116)

[8.5 Hexadecimální soustava 19](#_Toc160995117)

[8.6 Decimální soustava 20](#_Toc160995118)

[8.7 Binární soustava 20](#_Toc160995119)

[9 Hashovací metody 22](#_Toc160995120)

[9.1 MD5 22](#_Toc160995121)

[9.2 SHA 23](#_Toc160995122)

[9.3 LM 24](#_Toc160995123)

[9.4 NT 24](#_Toc160995124)

[9.5 CRC 24](#_Toc160995125)

[9.6 Bcrypt 25](#_Toc160995126)

[9.7 Argon2 25](#_Toc160995127)

[10 Šifrovací metody 26](#_Toc160995128)

[10.1 AES 27](#_Toc160995129)

[10.2 RC 27](#_Toc160995130)

[10.3 RSA 28](#_Toc160995131)

[10.4 DES 28](#_Toc160995132)

[11 Praktická část 29](#_Toc160995133)

[11.1 Vue projekt 29](#_Toc160995134)

[11.2 PrimeVue 29](#_Toc160995135)

[11.3 Základní funkce 29](#_Toc160995136)

[11.3.1 Base64 30](#_Toc160995137)

[11.3.2 URL 30](#_Toc160995138)

[11.3.3 ASCII 30](#_Toc160995139)

[11.3.4 BIN 31](#_Toc160995140)

[11.4 Hashovací metody 32](#_Toc160995141)

[11.4.1 LM a NT 32](#_Toc160995142)

[11.4.2 Bcrypt 33](#_Toc160995143)

[11.4.3 Argon2 34](#_Toc160995144)

[11.5 Šifrování 36](#_Toc160995145)

[11.5.1 RC 36](#_Toc160995146)

[11.5.2 DES 38](#_Toc160995147)

[11.6 Stylování 38](#_Toc160995148)

[11.7 Tlačítka 38](#_Toc160995149)

[11.8 Responzivita 39](#_Toc160995150)

[11.9 Import a Export souboru 40](#_Toc160995151)

[Závěr 42](#_Toc160995152)

[Seznam použitých zdrojů 43](#_Toc160995153)

[Seznam obrázků 47](#_Toc160995154)

[Seznam příloh 49](#_Toc160995155)

Úvod

Cílem této práce je vytvořit nejaktuálnější funkční webovou aplikaci, kde je možné vstupní textový řetězec převést do jiné podoby pomocí kódování, šifrování a hashování. Webová stránka má v sobě spoustu možností, jak pracovat s textem. Například ukládání a čtení souboru, kopírování, automatické ustřihnutí textu, prohazování vstupu a výstupu, resetování obou textových polí a přehledné možnosti si vybrat metodu na převedení textu. Důvod vybrání tohoto projektu bylo se naučit vzrůstající se framework Vue.js, obohatit znalosti v oblasti kódování, šifrování a hashování i testování webových aplikaci.

Dnes je penetrační testování relevantnější než kdy dřív, a proto tato aplikace obsahuje nejznámější funkce pro převod textu, kde se snaží uživateli co nejvíc usnadnit práci. V Rámci bezpečnosti jsou hesla z drtivé většiny šifrovány v databázích různých společností. Uživatel či (etický) hacker může testovat bezpečnost aplikací, kde mu tato stránka značně urychlí celý proces.  
 Při tomto projektu byly použity technologie Vue.js, díky které je vytvořené jádro aplikace. Dále byl použit TypeScript pro optimalizaci skriptovacího kódu. Nakonec je použito PrimeVue, které je dobré jak pro vzhled komponent, tak i práci s nimi.

# Teoretická část

## Penetrační testování

Každý den se v internetovém světě vyvíjí a publikují nové webové aplikace, technologie a služby jim podobné. S nimi přichází i nové bezpečností rizika, které mohou být zneužity nebo nahlášeny. Z etického hlediska je penetrační testování v našem světě relevantní čím dál tím víc. Soukromí běžných uživatelů se dá kompromitovat mnoha způsoby, jako je např. sociální inženýrství. Příčinou je globální digitalizace všech odvětví, čím roste taky poptávka pro specialisty v kybernetické bezpečnosti, kteří musí mít velmi rozšířené znalosti a mnoho nástrojů, pro rychlost a efektivitu v práci.

## Visual Studio Code

VS code je kódovací prostředí, kde je možné vidět složky, s kterými lze pracovat včetně jejich souborů. V tomto programu lze také použít mnoho rozšíření s mnoha funkcemi pro pohodlnější vývoj. Studio podporuje všechny možné typy souborů, jako je např. php, obrázek jpg, textový soubor apod. Aplikace také spolupracuje s Git, což podporuje přehlednost i rychlost progresu.

## TypeScript

TypeScript je lehce odlišná verze JavaScriptu, který je používán nejčastěji ve webových aplikacích. Je jeden z nejrozšířenějších multiplatformových jazyků vůbec. Má objektově založený programování, které může být v různých případech složité v průběhu vývoje kvůli přehlednosti typu proměnných. TypeScript nabízí proměnné s typy proměnných. Jeho kompilování je i není potřeba provádět na serveru stejně jako JavaScript. Pokud uživatel potřebuje převést JavaScript kód do TypeScript souboru, tak se kód provede bez problémů, ovšem bezpečnost pak může být narušena. [1]

## SCSS

Dá se říct, že SCSS je rozšíření kaskádových stylů s užitečnými doplňky. Samotný soubor nelze spustit a aplikovat pro webovou stránku, protože je třeba ho prvně zkompilovat přes kompilátor SASS. Ten ho jednoduše řečeno převede do klasického CSS souboru, který pak server nebo webový prohlížeč přijme.   
SCSS umožňuje vývojáři používat proměnné, obsahující barvy, názvy fontů apod. přímo v souboru. Dále je velmi užitečná funkce hnízdění, kde je možné pro libovolnou třídu nastylovat její podřadný prvek přímo v ní. Stylování může vypadat pak takto:  
*div*

*{*

*background: white;*

*p*

*{*

*color: gray;*

*}*

*}*

Je možné používat i funkce. Z největší části pracuje s proměnnými a patří mezi ně klíčová slova neboli příkazy: *@function*, *@return* a podobně. Dále SCSS poskytuje funkci výpočtů různých velikostí. Pokud uživatel chce vypočítat poměrovou velikost, použije např. tento příkaz: *width: (100% / 3);*. Propočty se vkládají i do podmínek, kde se opět pracuje nejčastěji s proměnnými. Syntaxe je opět jednoduchá, stačí přidat před výraz “@”. Poslední důležitou částí je příkaz “@import”, díky kterému se připojí další SCSS soubor s kódem. [2]

## Vue.js

Použitý framework byl vynalezen v roce 2014 stejnojmennou společností. Je v komunitním vývoji, což znamení programování v různých “branchích”. Jelikož je vyvíjen komunitou, tak se dá říct, že je jeden z nejpokročilejších a nejmodernějších frameworků. Využívá mnoho komponent pro usnadnění v částí programování. Absolutně vyniká v SPA (Single-page application), kde jsou pak dané aplikace vyvíjeny mnohem rychleji a efektivněji.   
Kód se kompiluje na straně serveru, díky čemu je posílena odolnost proti velkému provozu na webové stránce. Dále je obsah vykreslování nativně, což znamená absence virtuální Document Object Model. Díky této možnosti je zrychlený vývoj mobilních aplikací se zvolenou verzí Vue.js. [3]

## PrimeVue

Knihovna PrimeVue obsahuje responzivní vzhled pro Vue.js komponenty a usnadňuje jednoduchou editaci vzhledu jak přímo v kódování, tak i v předpřipravených komponentách na svých webových stránkách, kde je možné stáhnout námi upravený vzhled zcela bezplatně. Obsahuje také svoji knihovnu ikon, takže je tedy velmi podobná bootstrapu, akorát pro velké procento lidí může připadat více moderně a přívětivě. Pro nainstalování knihovny je potřeba mít nainstalováno NodeJS, díky kterému se nainstaluje balíček v příkazové řádce následovně: *npm install primevue*. Poté se předpokládá, že uživatel využívá programovací jazyk TypeScript, kde se importuje např. komponenta button: *import Button from 'primevue/button';*

## GitHub

Internetová stránka GitHub poskytuje programátorům prostředí, kde je možné ukládat jakékoliv projekty, u kterých je dostupná celá historie úprav. Repozitář si může zobrazit každý, pokud viditelnost je nastavena na veřejnou. V opačném případě má přístup jen tvůrce. GitHub má sadu svých nástrojů, jako je git. S ním se pracuje v příkazové řádce pro vytvoření repozitáře, editaci a podobně.

Dále poskytuje aplikaci GitDesktop. Toto uživatelské rozhraní umožňuje provádět pull, push a fetch origin “commitů”. Tedy zjednodušeně řečeno lehké aktualizování i stahování svých projektů. Přes tuto přehlednou aplikaci jsou možné vidět i konkrétní úpravy v kódu, otevřít kód v mnoha dostupných editorech a průzkumníku souborů.

## Kódovací metody

Hlavní vlastností kódování je jeho obousměrnost. U šifrování a hashování se v některých případech potřebujeme další proměnné pro získání naší hodnoty a pro provedení nemůžeme bez nich vstup získat zpět. Kódovací metody  se používají pro základní bezpečnost a funkcionalitu v přenosu dat, kryptografií i dalších oblastech. Hlavním účelem je bezchybný přenos dat s nejvyšší přesností, kdy počítač lépe pracuje s výstupem po operaci.

### Base64

Po síti komunikujeme přes určité kanály, které mají maximální bitovou kapacitu. Jsou dva známe typy BLOB a CLOB. BLOB je velký objektový soubor jako je třeba video, mp3 zvuk nebo obrázek. CLOB je znakový řetězcový soubor jako je XML a podobně. K tomuto typu používáme kódování Base64.

U této metody kódování můžeme převést znaky z ASCII tabulky. Ve funkci JavaScript diakritiku nelze převést, ale existují rozšířené metody pro např. e-mailové zprávy, kde háčky a čárky převést lze.  
 Z počátku se vypočte délka vstupního textového řetězce. Když není násobek tří, tak se dává nakonec např. znak “=”. Poté řetězec se přetvoří do ASCII hodnoty, kde z každé hodnoty, která má 8 bitů se vezme jen 6. Máme tedy číselnou posloupnost vytvořenou z 6bitových čísel. Nakonec, pomocí Base64 tabulky, se čísla přemění na konečnou podobu.  
 Zde je příklad praktické části:

Převede se textový řetězec “THS”. Každý znak převedeme do ASCII hodnoty. Vyjde nám tedy “84 72 83”. Nyní se převedou hodnoty ASCII do 8bitového binárního čísla:

*01010100-01001000-01010011*

Celkový počet cifer, což je 24, je dělitelný třemi. Nebudeme muset tak použít na konci znak “=”. 24 bitů si rozdělíme na 4 čísla a každé z nich má 6:

*010101-000100-100001-010011*

Hodnoty představují určitá čísla, která každé zvlášť následovně převedeme na hodnoty z Base64 tabulky:

*V-E-h-T*

Jestliže vstup není dělitelný třemi, tak převedení poslední hodnoty by vypadalo takhle:

*000100= -> 4= -> E=*

[4]

### URL

URL adresa se skládá z mnoha částí. První část je protokol, který je buďto HTTP nebo HTTPS. Hypertext transfer protocol (secure). Zabezpečená verze zajišťuje integritu a bezpečnost dat, které jsou přenesené po síti. Další následuje nepovinná subdoména, což je menší část hlavní domény, která je hned před ní. Poté je za ní doména nejvyššího řádu. Ta ukazuje většinou geolokaci stránky, ale může být učiněno i jinak. Např. “com”, “net”, ”org” a podobně. Když je zvolena jedna z geologických domén, tak některé weby mají i za tady tou doménou geologickou lokaci. Tuto část můžeme definovat jako část 1.

Druhá část se skládá z cesty k souboru, který je uložen na serveru na určité doméně. Ve vyhledávači se z větší části ukazují i přípony souborů. Buďto soubor programovacího jazyku jako je “.php”, či jiný čitelný soubor např. “.jpg”, “.txt”. Pokud na webu chceme nějak interagovat, tak je velká pravděpodobnost užití query dotazu. V doméně má svůj symbol “?”. Za ním je většinou formulovaný dotaz s logickými operátory. Poslední používaný prvek je kotevní identifikátor “<a>”, který v adrese začíná znakem “#” a za ním je parametr. Tento prvek v adrese se objeví většinou po kliknutí uživatel na hypertext odkazující na jinou část stránky s jistým ID. [5]

Některé znaky nelze použít v URL adrese, a proto pro ně používáme kódování. Dále jsou zde i znaky, které provádí určité operace. Pokud chceme použít jako textový řetězec, tak je musíme kódovat též. Znaky, které musíme kódovat jsou velká a malá písmena základní anglické abecedy, čísla, nezarezervované znaky   ”- \_ . ~” a rezervované znaky ”! \* ' ( ) ; : @ & = + $ , / ? % # [ ]”.  Rezervované se tak nazývají, jelikož mají nějakou úlohu, pokud jsou napsány v řetězci. Nerezervované nemají důležitý úkol. Slouží spíše jen pro přehled. [6]

Každý znak ve Windows 1252, který není ani v jedné kategorii výše se přemění v URL adrese na třímístný textový řetězec počínaje znakem “%”. Po něm je hodnota znaku v tabulce URL, která má podobu hexadecimální čísla. Pokud používáme utf-8 kódování, což používá technologie HTML5, tak může řetězec být i delší. Znak “<” může vypadat takto:

*%E2%80%B9* [4]

Pokud chceme kódovat znaky, které nejsou ve výchozí ASCII tabulce, tak textový řetězec ”上海+中國” vypadá takto:

*%E4%B8%8A%E6%B5%B7%2B%E4%B8%AD%E5%9C%8B* [7]

### ASCII

Základní ASCII tabulka obsahuje 128 znaků. Každý z nich se skládá ze 7 bitů. 95 z nich jsou grafické znaky, např. latinská abeceda (malé a velká písmena), arabské číslice a další interpunkční znaménka. Ostatní znaky jsou používány pro kontrolu nad procesy a poskytují ASCII kódovaná metadata. Každá ASCII hodnota má svojí jedinečnou hodnotu. Většinou je hodnota uvedena v decimálním, čí hexadecimálním čísle.

Postupem času vzrůstající (8 bitová, 16 bitová, ..) architektura podpořila rozšíření variant ASCII tabulky. Programovací jazyky mají obvykle nejmenší datový typ o velikosti 8 bitů. Technologie použila poslední 8. bit pro rozšíření tabulky. Jsou dvě populární znakové sady. Každá z nich používá 8. bit trošku jinak. Windows 1252 rozšiřuje tabulku o 123 znaků, 5 zbylých jsou prázdné. Potom je sada UTF-8, která se bere jako standard. Používá všech dalších 128 slotů, ve kterých jsou další obecné znaky. Pokud bychom měli nějaký méně známý a nepoužívaný znak, tak bychom potřebovali na kódování UTF-16 nebo třeba i UTF-32. [8]

### HTML

Hypertext markup language je jedna z nejpoužívanějších webových technologií díky své jednoduchosti a rozsáhlé podpoře. Má mnoho svých tagů, do kterých lze vkládat další i ručně definovaný. V hlavičce “<head>” má dokument své metadata definující např. název, popis, jazyk, skripty, styly CSS a mnoho dalších. Pak se obvykle používá tag “<body>” pro samotný vzhled. V něm už existuje spoustu možností, jak vizuální rozpoložení stránky provést. Jeden z prvků, co můžeme použít, je “<p>” tag vyhrazený pro paragraf. Pokud bychom chtěli do plaintextu zařadit znaménko “<”, tak HTML vyhodnotí znak jako začátek tagu. Právě proto se používají HTML entity pro tyto znaky, kterých je mnoho, a spoustu jiných dalších funkcí.

Znaménko “&” HTML řekne, že jde o entitu. Následuje hodnota, jež vložíme. Můžeme si vybra mezi napsáním číselné hodnoty nebo samotným názvem. Již uvedený znak “<” bychom pomocí písmen zakódovali takto:

*&lt;*

V Číselné podobě by entita vypadala takto:  
  
 *&#60;*

Co se týče přehlednosti kódu je lepší používat slovní označení. [6] Pokud chceme psát plaintext pouze v latinské abecedě a použít HTML entity pro háčky a čárky, tak HTML entity umisťujeme hned za písmenem, nad kterým má čárka být. Zde je příklad pro písmeno “á”:

*a&#769;*

Tato entita není v nejpoužívanějších znacích, a proto nemá svůj název. Použijeme tedy číselnou hodnotu jak v příkladu výše. [9]

Kódování HTML entit je důležité kvůli syntaxi kódu a bezpečnosti webové aplikace. Při nedostatku zabezpečení je webová stránka vystavena riziku útoku XSS (Cross-site scripting). To znamená, že útočník může vložit škodlivý skript do kódu ověřené stránky. Uživatel, který chce vložit vstupní textový řetězec může v realitě vkládat skript. Kód, který pracuje se vstupem by měl obsahovat knihovnu na kódování HTML entit a další bezpečnostní prvky při jakémkoliv případě. Už jen kvůli tomuto důvodu je dobré používat HTML entity nejvíc, jak jen to jde. [10]

### Hexadecimální soustava

Hexadecimální formát čísla obsahuje 16 hodnot. Prvních 9 jsou cifry od 0 do 9. Poté následují písmenka “A” až “F”. “A” v podstatě představuje číslo 10 a “F” se rovná 16. V praxi nezáleží na tom, jestli je písmeno velké nebo malé. Struktura pozicování čísel spočívá ve velikosti a váze daného čísla. Cifra nejvíce vpravo má nejmenší hodnotu z celého čísla. Každý další cifra vlevo se násobí šestnácti závisle na své pozici v čísle. Pro

Důvod užívání v praxi je jejich délka, pokud jde o číslo, reprezentující mnoho bitů. Navíc čísla jsou lehce srozumitelná celkově i v mikroprocesorové technice a uživatel od oka může rozlišit, o jak velké číslo jde. Dále se používá v datové vědě a analytice, strojovém učení umělé inteligence a vyjadřují se pomocí nich i hodnota barvy. Před hodnotou se zadává, že jde o hexadecimální číslo. U hexadecimálního formátu je zápis uváděn takto:

*0xB5,* kde číslo B5 je příklad.

Pro příklad převedení čísla z decimální soustavy “512” bude postup vypadat následovně:

*512 = 2x162 + 0x161 + 0x160 = 200*

Pokud bychom chtěli převést binární hodnotu “1011010101100001”, tak postup by probíhal takhle:

Rozdělíme číslo na 4 části po 4 bitech. Kdyby nebylo dělitelné čtyřma, tak bychom do bloku plně nalevo doplnili nulu:

*1011 0101 0110 0001*

Poté za každý blok popořadě dosadíme určitou hexadecimální hodnotu a vyjde nám výsledek

*B561* [11]

### Decimální soustava

Uživatelé používají nejčastěji tento způsob zápisu čísla při psaní vstupu. Pro paměť počítače jsou hodnoty dost nepraktické v rámci optimalizace. Jsou tedy vždy převedeny do hexadecimálního nebo binárního tvaru.

V programovacím prostředí se používá typ proměnný “decimal”, který má obrovskou maximální kapacitu. Tento typ je velmi relevantní pro vývoj bankovních aplikací, kde se pohybují obrovské částky i s hodně desetinnými čísly:

*1927.31895295*

### Binární soustava

Binární soustava je ta nejzákladnější a nejpoužívanější forma čísla v oblasti počítačů. Dnes existuje mnoho programovacích jazyků, které se vždy nakonec zkompilují do binárního tvaru.

Tato soustava používá pouze cifry 1 a 0, tedy ano/ne. Každá z těchto cifer představuje určitý bit, který má svůj účel. Pokud binární číslo chceme použít čistě pro převedení do jiné podoby např. decimální, tak číslo celé zpaměti lze číst zprava.

Např. číslo 0101. Bit úplně vpravo představuje hodnotu 1. V podstatě prozrazuje, zda je číslo sudé nebo ne. Druhý bit od prava je dvojnásobek toho předešlého. Čtvrtý bit představuje hodnotu 8. V přítomnosti jedničky bit započítáváme. Příklad lehkého výpočtu:

*0101 = 0.23 x 1.22 x 0.21 x 0.20 = 0 + 4 + 1 = 5*

Pokud jde o záporné číslo, existuje mnoho způsobů, jak vyřešit jeho zápis. Nejčastějším z nich je si vzít binární číslo v absolutní hodnotě a zjistit délku. Když je číslo dělitelné osmi, přidáváme další 4 bity a na první bit vlevo umístíme hodnotu 1. V opačném případě (délce např. 6 bitů) se přičtou 2 do osmi, tedy dělitelné čtyřmi. Následně se provede stejná operace, jako v prvním případě. Ve všech případech není třeba nutně číslo prodlužovat o 4 bity, ale jde o nejpřehlednější metodu. Druhý způsob, prováděný v inverzním kódu, prohodí všechny hodnoty, kromě té nejvíc vlevo. Proměna:

*0000 0101 => 1111 1010*

Náročnější operace je kódování reálných čísel s desetinnými čísly. Reálné číslo např. 63,72 si prvně převedeme na 6372 do binární podoby:

*0001 1000 1110 0100*

Tomuto základu říkáme Mantisa. Napravo od něj se nachází Exponent, což je blok, který značí o kolik se posouvá desetinná čárka. [12] V tomto případě má velikost 8 bitů. Jako poslední úplně vlevo je bit určující znaménko exponentu, zda bude kladný. Jestliže ano, tak desetinná čáka se posune doprava, v opačném případě doleva. Čím více bitů zabereme, tím přesnější převod bude:

*111111.10111000010100011111*

Pokud je potřeba zakódovat určitý znak, nepatřící do číslic, tak by byla převedena jeho ASCII hodnota. Hodnota znaku “\*”:

*\* = 42 => 0010 1010* [9]

## Hashovací metody

Podstatou hashovacích metod je jejich jednosměrnost. Se znalostí určitého výstupu v podobě hashe nelze zjistit, o jaký šlo vstup. Mezi obecné vlastnosti hashování patří délka, která je vždy stejně velká bez ohledu na délku vstupního řetězce, každá počáteční hodnota vždy dostane stejný hash nehledě na počet pokusů pro převedení. Existuje mizivá šance, že dva různé vstupy mají stejný výstup. Jestliže se tak stane kolize, tak se dá usoudit nízká efektivita a bezpečnost dané hashovací funkce. Uplatnění metod je nejvíce uplatňováno v databázích a ověřování integrity dat. V databázích se používá nejčastěji pro zabezpečení hesel před útočníky. [13]

### MD5

Mezi nejznámější hashovací metody patří MD5, která byla vynalezena v roce 1991 Ronaldem Rivestem. Výhody oproti minulé verzi MD4 je bezpečnost.  Je možné se s ním setkat u ověřování souborů a uživatelských hesel. Jeho pevná délka výstupu je 128 bitů. Mezi výhody patří jeho výkonová optimalizace a srozumitelnost. Vygenerovat zprávu libovolné velikost nezabere tolik času ani paměti, a proto je mezi programátory velmi populární. Na druhou stranu, jelikož je to poměrně starý algoritmus, má pár nevýhod. Pro některé uživatele se může zdát nejistý a špatně zabezpečený přes SHA1. Lze ho tedy nahradit za metodu SHA256. U prvního kroku pro převedení potřebujeme určitý násobek 512, od kterého se poté odečte 64.

*Zpráva má 1000 bitů, počítá se do nejbližšího násobku (512\*3 = 1536), 1536 - 64 = 1472.*

Poté se rozdělí na 4 vyrovnávací paměti o velikost 32 bitů.

*J = 0x67425301, K = 0xEDFCBA45, L = 0x98CBADFE, M = 0x13DCE476*

Nejdůležitějším krokem je zpracování každého bloku. Ve 4 kolech se provede 64 logických operací s operátory AND, OR, XOR a NOT. V prvním kole funkce F, v druhém G, v třetím H, ve čtvrtém I. Operace tak budou vypadat takto:

*F(K,L,M) = (K AND L) OR (NOT K  AND M)*

*G(K,L,M) = (K AND L) OR (L AND NOT M)*

*H(K,L,M) = K XOR L XOR M*

*I(K,L,M) = L XOR (K OR NOT M)*

Nyní je třeba provést 4 operace:

*Přidat modulo 232*

*M[i] – 32bitová zpráva.*

*K[i] – 32bitová konstanta.*

*<<<n – Posun doleva o n bitů.*

Paměť K bude přivedena do L, L do M a M do J. V prvním kroku budou převzaty výstupy K, L a M a poté se na ně aplikuje funkce F. Přidáme modulo 232bitů pro výstup s J. Ve druhém kroku se přidá bitová zpráva M[i] s výstupem prvního kroku. Poté bude přidána 32 bitová konstanta, tj. K[i] na výstup druhého kroku. Nakonec se provede operace posunu doleva o n (může být jakákoliv hodnota n) a sčítání modulo o 232. Po všech těchto krocích bude výsledek J uložen do K. Stejný kroky budou tak provedeny i u pamětí G, H a I. Po úspěšně provedených 64 operacích bude dostupný souhrn. Vyrovnávací paměti J, K, L a obsahují výstup. Výsledný hash výstupu např. ’’This is a message sent by a computer user.’’ bude vypadat takto:  
*922547e866c89b8f677312df0ccec8ee*. [14]

### SHA

Secure hash algorithm je skupina několik metod rozdělených podle hodnot konečné hodnoty v bitech. Momentálně je jedna z nejpoužívanějších SHA 256. Pro veřejnost vyšlo toto hashování v roce 2001 a společnosti NSA a NIST chtěli představit nového nástupce zastaralých metod SHA1, SHA2, atd. Jeho použití v praxi je nejvíc v digitálních podpisech, kontroly integrity a SSL handshake, který hlídá bezpečné připojení mezi prohlížečem a webovým serverem.   
Máme počáteční prostý text, který musí být upraven tak, aby byl násobkem čísla 512. Pro začátek se musí vyplnit 8 vyrovnávacích pamětí. Jedna z nich může vypadat následovně: a = 0x6a09e667. Poté uložit 64 různých klíčů do pole [0-63]. Klíče budou mít stejný formát jako hodnoty ve vyrovnávací paměti. Jakmile mám inicializované pole, tak celý text bude rozdělen do několika bloků po 512 bitech. Každý blok je prováděn 64 cykly. Výstupy po cyklech pak slouží pro vstupy pro následující blok. Celá operace končí, až se provede poslední činnost v 512. bloku. Po tomto posledním cyklu máme konečný výstup. [15]

### LM

Tento způsob hashování se používá spíše ve starších operačních systémech a celkově zastaralých prostředí. Mohou být znovu aktivovány ve Windows pomocí GPO (Group Policy Objects). Kroky pro převedení vstupu jsou prosté a jednoduché pro pochopení. Operace by začala převedením všech malých znaků na velké. Pro příklad bude náš řetězec mít délku 14 znaků. Tento text rozdělíme na 7 a 7 charakterů. Ke každému z nich vytvoříme DES klíč. DES zašifruje řetězec "KGS!@#$%" těmito dvěma řetězci. Jejich výstupy spojíme a dostaneme výsledný hash.  
Pokud by byla potřeba hash prolomit, tak je možno použít program John The Ripper, který je určen pro penetrační testování. Použijí se tyto dva příkazy: john *--format=lm hash.txt* a *hashcat -m 3000 -a 3 hash.txt.*[16]

### NT

NT neboli NTLM (New Technology LAN Manager) je novější verze typu LM. Tyto dva způsoby se liší tím, že NTLM dokáže zahashovat mnohem delší hesla s téměř neomezenou délkou.

### CRC

Cyklická redundantní kontrola slouží pro kontrolu chyb v různých datech. Proto díky její činnosti se dá považovat za plnohodnotnou hashovací funkci. Detekuje změny nezpracovaných dat používaných v sítích a paměťových zařízeních, jako jsou pevné disky. Je velmi snadno implementovatelná do hardwaru v počítači. Její matematická analýza je taktéž jednoduchá a to jí dělá jednou z nejlepších hashovacích funkcí.  
Základ této metody vynalezl W. Wesley Peterson v roce 1961. Poté se vývoje chopila společnost CCITT (Comité Consultatif International Télégraphique et Téléphonique).  
CRC je založena na binárním dělení a nazývá se i "polynomický kódový kontrolní součet". Vždy když je třeba poslat zprávů, tak se k ní připojí určitý kontrolní počet bitů. Příjemci vyhodnocují, zda mají chyby či ne. Vždy pokud jsou data jakkoliv narušena, vysílá se žádost o nové doručení. Když počítač přečte poškozený nebo nekompletní soubor, spustí se cyklická redundance. CRC může pocházet z jiného zařízení nebo třeba z CD disků. Mezi nejčastější chyby patří selhání systému nebo soubory s velkým množstvím chyb. Při navrhování CRC záleží na implementaci prostředků, hlavně na jejich intenzitě, které mohou ovlivnit výkon. Celkově je tento způsob velmi účinný a má dobrou strategii pro integritu dat. [17]

### Bcrypt

Patří k nejsilnějším hashům, který používá metodu šifrování Blowfish. Díky svému proměnlivému počtu kol se používá pro spoustu systémů (zejména BSD). Jedinou její slabinou je hashování pouze prvních 72 znaků hesla, zbytek je ve výstupu plaintext. Pro vymazání tohoto problému používá mnoho aplikací předem hash SHA. Pokud používáme knihovnu Passlib a chceme převést textový řetězec znaků mimo rozsah [./A-Za-z0-9], tak nám metoda vrací ValueError. Již zmíněná Passlib obsahuje dobře funkční funkce pro hashování a pokud je potřeba s ní pracovat, tak musíme určitým způsobem zakódovat vstup s formátem UTF-8. Bcrypt si už prošel moho verzemi. Každý se identifikuje různým prefixem jako např. $2x$, $2y$, $2b$. Každopádně všechny se považují za zastaralé kromě $2b$.

Příklad výsledného výstupu slova “password” za použití knihovny Passlib může vypadat takto: *$2b$12$GhvMmNVjRW29ulnudl.LbuAnUtN/LRfe1JsBm1Xu6LE3059z5Tr8m*. Tento hash se skládá z 5 částí, kterou jdou po sobě v tomto pořadí: typ ($2b$), počet kol, který představuje parametr 12 v již zmíněném hashy. Tyto dvě čísla reprezentují 2 desetinná čísla doplněné nulou. Následuje takzvaný “salt”, který představuje znaky v zmíněném hashy: *GhvMmNVjRW29ulnudl.Lbu*. Poslední znak by pokaždé měl pocházet z této charakterové sady: *[.Oeu]*. Pokud je učiněno jinak, tak by v některých implementacích mohlo dojít k negativnímu výsledku při ověřování. Poslední část je kontrolní součet, který obsahuje 31 znaků, neboli zbytek po “saltu”. [18]

### Argon2

Tato hashovací metoda využívá taktéž sůl, který je náhodně vygenerovaný řetězec znaků, který pracuje s hashovací funkcí. Sůl je vždy jiný a má různé délky. Důsledkem toho je, že dva se stejným heslem budou mít dva různé výstupy, což výrazně redukuje nebezpečí. Argon2 je známý svou náročností na hardware, protože pro vygenerování hashe je možnost použít mnoho parametrů, kde se i přidává pamětní velikost. Klíčový parametr je počet iterací, kolikrát se bude hashovat daný vstup. Čím více iterací, tím logicky těžší prolomení pro útočníka. Z určitého většího počtu mohou být šance téměř nulové.   
U hashování je tedy 5 parametry. Je potřeba zvolit vstup, který bude převeden. Dále již zmíněný počet iterací, kolikrát se cyklus operací provede. Poté počet jednotek v paměti. Většina knihoven má tento parametr základní jednotku kilobajt. Může být zvoleno např 65536 kilobajtů, což je 216. Dále je nejdůležitější parametr délka výstupu s běžnou délkou 32+ znaků a jako poslední je třeba zvolit počet vláken procesoru. Vybírá se např. 1. [19] Průběh celé operace vypadá následovně:

Obsah obrázku text, snímek obrazovky, diagram, Paralelní

Popis byl vytvořen automaticky

Obrázek 1 Popis činnosti hashovací funkce Argon2 [20]

## Šifrovací metody

Šifrování nejčastěji probíhá na komunikačním protokolu HTTP Secured nebo při práci s disky. Při každé operaci je vytvořen klíč, bez kterého není možné získat zpátky vstup, čímž se tyto metody hlavně liší od hashování.   
Existují dva typy šifrování. Symetrické, u kterého je při šifrování a dešifrování použit pouze 1 klíč, který komunikující strany musí znát. Druhý a sofistikovaný je asymetrický s veřejným a privátním klíčem. Veřejný nemusí znát příjemce zprávy. Privátní nezná odesílatel zprávy. Sofistikovanost procesu umožňuje nejen důvěrnou komunikaci, ale i ověření odesílatele, k čemu je možnost použít digitální certifikát.

Jelikož je šifrování širokým pojmem i v právním oddělení, tak existuje mnoho zákonů a předpisů, jako je např. GDPR, PCI-DSS, HIPAA, SOX and GLBA. Systémy s určitými funkcemi tak jsou vytvořený velmi důmyslně, aby nenastaly právní potíže. [21]

### AES

Advanced Encryption Standard je jedním z nejdůležitějších algoritmů v oblasti šifrování důležitých dat. Je používán firmami jako např. Apple nebo Microsoft. Jelikož je to jedna z nejnovějších metod současnosti, tak mezi její plusy patří zabezpečení, výkonová optimalizace na hardware i jednoduchost.  
Pracuje s algoritmem blokové šifry, které jsou rozdělovány pomocí velikosti. Existují 3 typy. Mezi ně patří 128bitové, 192bitové a 256bitové. Velikost o 256 bitech je logicky nejbezpečnější. Každý z těchto typů má určitý počet sérií operace. 128b má 10 sérií, 192b má 12 a 256b má 14. Jelikož je to symetrický způsob šifrování, tak se při zašifrování a dešifrování používá stejný klíč. Odesílatel a příjemce tak sdílí stejný řetězec pro tento klíč.  
AES je ve zkratce nástupce DES, který byl vytvořen 20 let přes AES (1999). DES bylo lehce prolomitelné a dle výpočtů je dokonce AES 6 krát efektivnější, než Triple DES. [22]

### RC

Tato šifrovací metoda má za sebou pár verzí, ale tato část bude pojednávat o nejnovějším typu RC4. Její jednoduchost a rychlost jí udělalo jednou z nejpoužívanějších proudových šifer. Je možné ji najít v SSL nebo znamém HTTPS. Dříve bylo užíváno v domácích sítích WIFI pod zabezpečením WEP  
Pro začátek se vytvoří takzvaný S-box o velikost 256 bajtů. Všechny čísla jsou seřazeny do intervalu <0;255>. Této části se říká inicializace. Poté se tento S-box prochází se zadaným uživatelským klíčem (40-128 bit). Během toho dochází k různým permutacím prvků s možností přetečení prvků pole. Ke konci proběhne prohození prvků v poli S-Box. Skončila tedy fáze Key Scheduling Algorithm. Nyní nastává poslední fáze Pseudo-Random Generation Algorithm, kde se předá výsledný S-BOX do PRGA, což je generátor tzv. keystreamu. Tato operace nevyžaduje klíč a ke konci dojde k dalšímu prohození hodnot prvků pole. [23]

### RSA

Tento algoritmus je asymetrický, tudíž využívá privátní a veřejný klíč. Veřejný klíč se skládá z dvou čísel, kdy jedno z nich je násobkem dvou prvočísel. Soukromý klíč též stejnými prvočísly. Pokud je tedy uživatel schopný faktorizovat ono velké číslo, tak privátní klíč je kompromitován. Jako závěr k tomuto se dá usoudit, že síla šifry spočívá na její velikosti. Prioritizováno je použití délky 1024 bitů nebo 2048 bitů. Prolomení 2048 bitového klíče je dodnes zatím téměř nemožné.  
Zjednodušeně v prvním kroce si vyberem určité dvě prvočísla. Jejich součin by měl mít větší délku, než uživatelem šifrovaná zpráva. To bude modul pro RSA. Poté dojde k šifrování s pomocí modulárních umocňovacích algoritmů. Dešifrování používá stejný algoritmus. [24]

### DES

Data Encryption Standard je způsob šifrování, kde jeho algoritmus používá standardně 56 bitový klíč. Pokud má klíč více bitů, tak má stejný počet kol jako AES. Např. při 192 bitovém klíči se provádí 12 kol. Způsob šifrování je symetrický. Je tedy zapotřebí pouze 1 klíč.   
Jeho stvořitelem je firma IBM s datem publikování 1971. Již zmíněno, DES bylo nejaktivnějším šifrováním dlouhých 30 let, než přišlo AES.  
Jako první krok je počáteční permutace, kde je plaintext rozdělen na části o 64 bitech. Zde se implementuje transpozice. 58. bit nahradí 1., 57. bit nahradí 2. atd. Poté dojde k transformaci klíče. 56 bitový klíč se získá odstraněním každého 8. bitu v 64 bitovém klíči. Výsledkem toho vznikne 48 bitový řetězec. Existuje také verze Triple DES používající 3 klíče k permutaci.   
Nyní dochází k samotnému šifrování. Text se rozdělí na levý a pravý. Pravý i levý prochází 16 koly. Oba bloky se poté spojí. Provede se znovu permutace s výsledkem 64bitového šifrovaného textu.   
I když může být DES algoritmus na první pohled efektivní, není tomu tak. Dá se prolomit hrubou silou, má menší kapacitu bitů pro šifrování, zkrátka je lepší použít AES. [25]

1. **Praktická část**

V této části je popsáno, jak probíhal vývoj od vytvoření a inicializace projektu, včetně učení nové technologie Vue.js 3, stylování s použitím knihovny, programování metod pro převedení textu, problematika vkládání externích knihoven, podmínky logiky obousměrnosti a jednosměrnosti funkcí a testování vstupů a výstupů. Webová stránka s komponentami tlačítek, textových vstupů, rozevíracích seznamů bude detailně popsána s veškerou funkčností.

## Vue projekt

Při úplném začátku byl založen Vue.js projekt, v kterém bylo na výběr si vybrat TypeScript. Zvolil jsem ho díky své přehlednosti, jelikož se v řešení nachází spoustu datových typů. Dále bylo zvoleno použití Prettier pro rychlé opravení překlepů v kódu. Ke konci zakládání byly smazány výchozí šablony nahrazené mojí Single Page aplikací.  Po těchto procesech byl projekt publikován na GitHub s inicializačním commitem.

## PrimeVue

Pro responzivní optimalizovaný vzhled byla importována knihovna PrimeVue. Počátkem byly vloženy tlačítka, textový inputy a dropdowny. Projekt má být implementován do platformy Penterep, tudíž byly ihned zvoleny hlavní barvy oranžová s černou.

## Základní funkce

Než dojde k přidání úvodních kódovacích metod, tak byly naprogramovány funkce pro vyměnění vstupu s výstupem a kopírování daných textových polí. Bloky kódů byly vloženy do hlavního souboru App.vue, kde je i HTML kostra stránky i dodatečný SCSS kód, přepisující výchozí hodnoty vzhledové knihovny PrimeVue. Následují jednotlivé kódovací metody, ale jako defaultní hodnota dropdownů, kde jsou veškeré metody se zkratkami, tak je plaintext.

### Base64

TypeScript má ve své základní verzi zabudovanou funkci pro převedení textu na Base64 kódovaný řetězec i naopak. Implementace byla tedy zhotovena rychle.   
Funkce pro zakódování textu je btoa(). Opačnou operaci provádí funkce atob().

### URL

Pro převádění textu do čitelného tvaru URL se postarala základní funkce *encodeURI(input)/decodeURI(input).* Výstupem metody je textový řetězec typu String. práce s textem a celkově s převedením je opět jednoduše pochopitelná.

### ASCII

Kódování ASCII bylo první ručně více napsaný kód, kde při operacích procházím cyklem “for” každý dostupný znak pomocí funkce *.charCodeAt().*  
Pro plnou funkčnost výstupu je rozdělení hodnot řešeno 1 mezerou.

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

Pro automatické přidávání mezer při psaní vstupu je vytvořena metoda AreaCheck obsahující kód s automatickým přidávání mezer.

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

### BIN

Binární kódování bylo jedno z nejdůležitějších pro vývoj, jelikož je třeba rozlišovat čísla a hodnoty znaků v tabulce ASCII.   
Základní funkcí pro převedení řetězce s hodnotami 0 a 1 je BinToPlain()

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

## Hashovací metody

Hashovací metody mají velkou podporu na stránce NPM JS, kde je mnoho přispěvatelů k daným metodám s mnoho verzemi. Stačilo tedy importovat danou knihovnu. např.:

*import { sha256, sha224 } from 'js-sha256';*

Poté stačí zavolat funkci a vypsat kód:  
*value2.value = sha256(plainText);*Tímto způsobem byly implementovány metody sha256, md5 a crc.

### LM a NT

Obsah obrázku text, snímek obrazovky

Popis byl vytvořen automatickyS importem LM a NT byl problém s funkčností, a tak jejich zdrojový kód, ten který byl třeba, byl implementován do hlavního souboru App.vue. Byly použity 4 metody. První vytváří samotný LM hash. Je použit buffer pro generování bitů. V této funkci je použita i DESjs knihovna. Dále je použit kód s mapou pro hexadecimální hodnoty na pole binárních čísel. Dále kód s reverzní funkčností. Následně funkce, která vkládá 0 u každého 7. bitu. Nakonec je použita metoda pro generování NT hashe, která má 4 řádky s použitím bufferu, md4, použití bufferu u nové md4 a vrácení textového řetězce.

### Bcrypt

**Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky**Implementace metody Bcrypt byla jednoduchá jako většina zmíněných způsobů hashování, akorát bylo třeba trochu upravit front-end pro možnosti volby počtu soli, což je povinný parametr pro vygenerování výstupu. PrimeVue má předem nastavený pravidla pro své komponenty, ale základní hodnoty neodpovídají vzhledu aplikace, tudíž musel být přidán jistý selektor do samotného souboru primevue.css, kde se primárně nepřidává kód navíc.

### Argon2

Tato metoda se primárně vykonává na straně serveru, ale jelikož je projekt čistě závislý pouze na prohlížeči, tak musel být vybrán jistý modul a specifická cesta k souboru. Dále zadávání parametrů pro vygenerování řetězce bylo řešeno formou dialogu, který má PrimeVue výborně optimalizovaný a připravený pro uživatele.

Obsah obrázku text, snímek obrazovky, software, design

Popis byl vytvořen automatickyDále byla použita technologie PrimeFlex, která spadá pod PrimeVue, aby textové vstupy i tlačítka byly vycentrované a zobrazovaly se správně ve všech rozlišeních.Obsah obrázku text, snímek obrazovky, software

Popis byl vytvořen automaticky

Pro rychlost testování byla vytvořena funkce, která generuje náhodný klíč, který pak bude aplikován ve výpočtu hashe. Obsah obrázku text, snímek obrazovky, Písmo, software

Popis byl vytvořen automaticky

Celkově největší časové vytížení způsobila kompatibilita se způsobem kompilace na straně prohlížeče. Zbytek stráveného času bylo vymýšlení způsobu zobrazení pro největší srozumitelnost uživatele.

## Šifrování

Šifrování je kódově nejsložitější hlavně díky své dvou směrnosti a počtem parametrů. V různých metod se pracuje s mnoha prvky jako např. veřejný klíč, privátní klíč a vektor. Většina knihoven přijímá vstupy jako Uint8array, což značně komplikuje vývoj metod.

### RC

Způsob RC4 byl implementačně nejjednodušší. Vstupy přijímají stringy a jako jediný parametr mimo vstupní řetězec je zapotřebí jen privátní klíč. Nejlepší dostupnou knihovnou byla zvolená CryptoJS. Implementace její metod byla jednoduchá.

**Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky**

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automatickyPro generování privátního klíče byla ručně napsána metoda, která může vygenerovat, jakkoliv dlouhý klíč, ale primárně se v programu generuje s délkou 16 znaků. Pro specifické požadavky si uživatel může ručně svůj klíč zadat do textového pole.

### DES

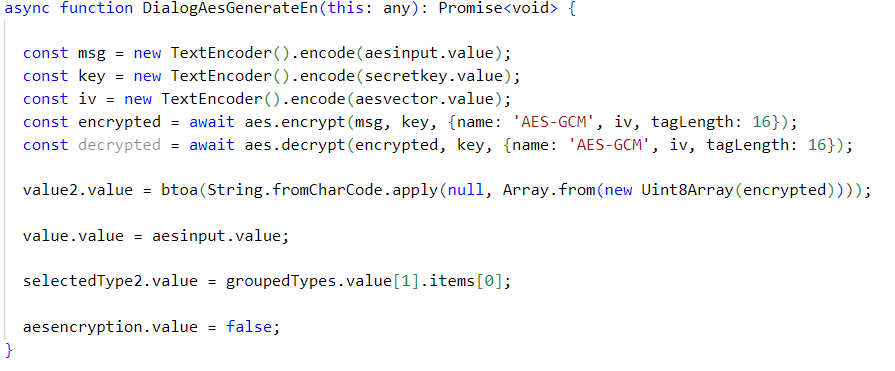
DES pochází ze stejného balíčku Crypto.js společně s RC4, přičemž oba mají téměř identický zápis vyvolávání funkce. Implementace se dá charakterizovat jako duplicitní, kde je pouze místo RC4 dosazeno “DES”.

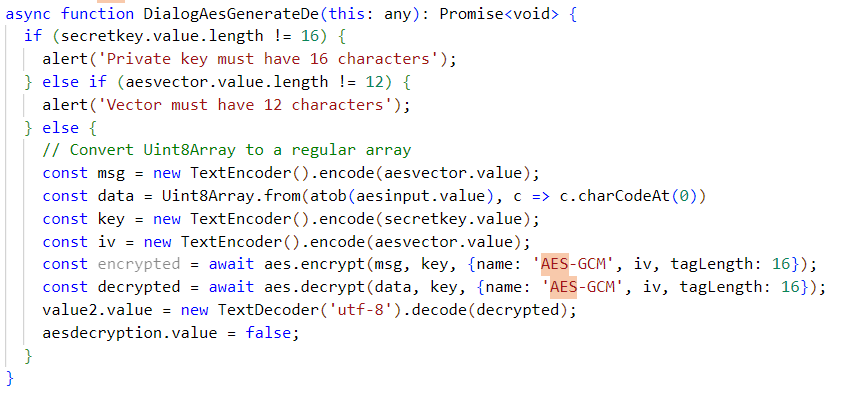
Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

### AES

Šifrování AES má mnoho způsobů, nicméně do aplikace byla vybrána GCM z knihovny JS Crypto AES. GCM pro výstup vyžaduje vstupní vektor a privátní klíč. Oba údaje jsou při šifrování i dešifrování povinné pro vyplnění. Všechny vstupní a výstupní data jsou primárně v datovém typu Uint8array, což pro čitelný výstup uživatele vyžadovalo metodu TextEncoder a TextDecoder.

Při šifrování AES uživateli není poskytnuta žádná metoda pro generování klíče, a tak je potřeba napsat do textového pole manuálně. Všechny funkce jsou volány asynchronně z důvodu požadavků vývojářů knihovny. Pracují tak paralelně. Nakonec pro správné fungování dialogu je zavedena proměnná, která nám po vykonání funkce zavře překrývající okno. Dešifrování je řešeno podobným způsobem, akorát je položena otázka, zda mají určité vstupy validní délku. Jestliže ne, tak vyskočí nahoře alert s příslušnou chybovou hláškou.



## Stylování

Pro vizuální stránku byly použity primární 3 barvy platformy Penterep. Vlastnosti komponent byly v souboru předem definovány, ovšem vůči požadavkům zadání práce se změnily barvy a velikost textboxu, když se očekávají dlouhé výstupy.

## Tlačítka

Veškeré tlačítka na stránce mají absolutní velikost čtverce s délkou strany 59.2 pixelů. Mají oranžové pozadí s bílým obsahem pro dobrou čitelnost. Základní atributy každého tlačítka jsou aria-label pro popis, @click pro provedení dané činnosti a icon pro importování dané ikony z balíčku PrimeVue nebo mdiVue.

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

## Responzivita

Obsah obrázku text, snímek obrazovky, software

Popis byl vytvořen automatickyNa stránce jsou k dispozici prvky např. tlačítka, dropdowny, textové pole apod. Textové pole má zarezervovanou ve výchozím nastavení poměrně velkou oblast. Pokud by uživatel potřeboval z určitých důvodů použít aplikaci na mobilním zařízení, tak se prvky neřadí po sobě do řádku, ale do sloupce.

## Import a Export souboru

Pro vypsání obsahu je zatím dostupný jen textový formát souboru. Pro tuto byl použit HTML prvek input typu file. Ve vlastnostech jsou důležitý body @input, který vyvolá funkci s dialogem a accept, kde jsou napsané podporované přípony.

Obsah obrázku text, Písmo, snímek obrazovky

Popis byl vytvořen automaticky

Kód funkce pracuje s cestou souboru, odkud si vezme určitý zdroj, kde se poté rozdělí název cesty, z které dostane i příponu souboru. Pokud se přípona souboru nerovná txt nebo text, tak se zobrazí alert dialog. V opačném případě se spustí metoda reader a vyplní vstup obsahem vybraného počátečního vstupu.

Obsah obrázku text, snímek obrazovky

Popis byl vytvořen automaticky

Pro export souboru je nachystaná jednoduchá funkce pracující s již naprogramovanými metody saveAs a třídou Blob.

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

Závěr

Vytvořená šablona maturitních prací obsahuje formální požadavky maturitních prací na SPŠT Třebíč. Jedná se zejména o upravené styly v dokumentu, podrobný popis jednotlivých částí maturitní práce a jejího obsahu, snadno editovatelné záhlaví a zápatí s automatickým číslováním stránek a propojení stylů se seznamy a obsahem.

Seznam použitých zdrojů

1. TYPESCRIPT. The TypeScript Handbook. Online. Dostupné z: https://www.typescriptlang.org/docs/handbook/intro.html. [cit. 2024-01-20].
2. BITTNER, Honza. Úvod do CSS preprocesoru Sass. Online. Dostupné z: https://www.itnetwork.cz/html-css/webove-portfolio/tutorial-moderni-webove-portfolio-sass. [cit. 2024-01-20].
3. KOĎOUSKOVÁ, Barbora. VUE JS: VÝHODY, NEVÝHODY A MOŽNOSTI VYUŽITÍ. Online. Dostupné z: https://www.rascasone.com/blog/co-je-framework-vuejs. [cit. 2024-01-19].
4. KUMAR, Akshay. BuiltIn. Online. Dostupné z: https://builtin.com/software-engineering-perspectives/base64-encoding. [cit. 2023-10-27].
5. DEEPIKA92. What is a Web Address? Online. Dostupné z: https://www.geeksforgeeks.org/what-is-a-web-address/. [cit. 2023-11-03].
6. GOOGLE. URL Encoding. Online. Dostupné z: https://developers.google.com/maps/url-encoding. [cit. 2023-11-03].
7. W3SCHOOLS. HTML URL Encoding Reference. Online. Dostupné z: https://www.w3schools.com/tags/ref\_urlencode.ASP. [cit. 2023-11-03].
8. BAELDUNG. *Introduction to ASCII Code*. Online. Dostupné z:<https://www.baeldung.com/cs/ascii-code>. [cit. 2023-11-10].
9. DIAS, Danielle. *What is HTML Entity Encoding (HTML Entity Encode)*. Online. Dostupné z:<https://medium.com/geekculture/what-is-html-entity-encoding-html-entity-encode-be0b9124307c>. [cit. 2023-11-12].
10. W3SCHOOLS. HTML Entities. Online. Dostupné z: https://www.w3schools.com/html/html\_entities.asp. [cit. 2023-11-12].
11. AWATI, Rahul. Hexadecimal. Online. Dostupné z: https://www.techtarget.com/whatis/definition/hexadecimal. [cit. 2023-11-12].
12. UMÍME INFORMATIKU. Binární čísla: záporná a desetinná. Online. Dostupné z: https://www.umimeinformatiku.cz/cviceni-binarni-cisla-zaporna-desetinna. [cit. 2023-11-12].
13. ŠTRÁFELDA, Jan. Co je to hash či hashování. Online. Dostupné z: https://www.strafelda.cz/hash. [cit. 2024-01-05].
14. sachinkumar04. What is the MD5 Algorithm? Online. Dostupné z: https://www.geeksforgeeks.org/what-is-the-md5-algorithm/. [cit. 2024-01-12].
15. KUMAR JENA, Baivab. *A Definitive Guide to Learn The SHA-256 (Secure Hash Algorithms)*. Online. Dostupné z: <https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm>. [cit. 2024-01-12].
16. GOMBOS, Péter. LM, NTLM, Net-NTLMv2, oh my!. Online. Dostupné z: https://medium.com/@petergombos/lm-ntlm-net-ntlmv2-oh-my-a9b235c58ed4. [cit. 2024-01-18].
17. ROUSE, Margaret. Cyclic Redundancy Check. Online. Dostupné z: https://www.techopedia.com/definition/1793/cyclic-redundancy-check-crc. [cit. 2024-01-18].
18. PASSLIB. Passlib.hash.bcrypt - BCrypt. Online. Dostupné z: https://passlib.readthedocs.io/en/stable/lib/passlib.hash.bcrypt.html. [cit. 2024-01-18].
19. NAGENDRA, Saajan. Hashing With Argon2 in Java. Online. Dostupné z: https://www.baeldung.com/java-argon2-hashing. [cit. 2024-01-19].
20. EUM, Siwoo; KIM, Hyunjum a SEO, Hwajeong. Algorithm of Argon2. Online. In: . Dostupné z: https://pub.mdpi-res.com/applsci/applsci-13-09295/article\_deploy/html/images/applsci-13-09295-g004.png?1692175815. [cit. 2024-01-19].
21. ESET. Šifrování. Online. Dostupné z: https://www.eset.com/cz/sifrovani-dat-ve-firme/. [cit. 2024-01-20].
22. SHELBY, Taylor. Kompletní návod na standard pokročilého šifrování (AES). Online. Dostupné z: https://cs.wizcase.com/blog/kompletni-navod-na-standard-pokrocileho-sifrovani-aes/. [cit. 2024-02-08].
23. POLÍVKA, Jaroslav. Lekce 4 - Pod pokličkou šifrovacího algoritmu RC4. Online. Dostupné z: https://www.itnetwork.cz/algoritmy/kryptografie/pod-poklickou-algoritmu-rc4. [cit. 2024-02-08].
24. GEEKSFORGEEKS. RSA Algorithm in Cryptography. Online. Dostupné z: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/. [cit. 2024-02-08].
25. SIMPLILEARN. What Is DES (Data Encryption Standard)? DES Algorithm and Operation. Online. Dostupné z: https://www.simplilearn.com/what-is-des-article. [cit. 2024-02-08].

Seznam obrázků

[Obrázek 1Popis činnosti hashovací funkce Argon2 26](#_Toc160876558)

Seznam příloh

Prázdná šablona maturitní práce