Střední průmyslová škola dopravní, a.s.

Plzeňská 298/217a, Praha 5 - Motol

Profilová maturitní zkouška z odborných předmětů

Praha 2022

Daniel Iliev

Střední průmyslová škola dopravní, a.s.

Plzeňská 298/217a, Praha 5 – Motol

CoronaTracker

Autor : Daniel Iliev

Obor vzdělání : 18-20-M/01 Informační technologie

Třída : 4. A

Školní rok : 2021/2022

Vedoucí práce : Ing. Bc. Kaněrová Soňa, DiS.

Čestné prohlášení autora

Prohlašuji, že jsem závěrečnou práci včetně příloh vypracoval(a) samostatně a uvedl(a) jsem všechny použité prameny a literaturu.
Souhlasím, aby moje práce byla k dispozici k prezenčnímu studiu na SPŠD, a.s.

V Praze dne……. ……
podpis autora (jméno a příjmení)

Abstrakt

CoronaTracker je aplikace, která slouží pro medicínské účely. Jejím hlavním účelem je monitorování pacientů, a to zejména jejich prodělání Covid-19, nebo jejich očkování. Dále aplikace získává statistiky z celého světa z jiných stránek pomocí metod Rest API. Aplikace je napojena na vlastní databázový server konkrétněji MySQL server. V něm můžeme nalézt data kupříkladu: očkování pacientů; pacienti a jejich záznamy očkování a prodělání Covid-19, typy vakcín; zaměstnanci; a tak dále... Aplikace má mnoho funkcí, které jsou shrnuty v tomto dokumentu.

Abstrakt

CoronaTracker is an application focused on the medical sector, and its primary purpose is to monitor patients, especially theirs's Covid-19 illnesses and theirs's vaccination. The application also collects statistics about Covid-19 worldwide from external data sources via API gateways. CoronaTracker is attached to an internal relational database server where it stores all sorts of information, such as patients, patient vaccination, types of vaccinations, employee's data and much more. CoronaTracker has many exciting features, which are summarized in this document.

Obsah

Úvod

Dané téma jsem si zvolil z prostého důvodu. Mám rád výzvy a v době výběru tématu se světem prohánělo onemocnění Covid-19, a tak byl výběr jasný. Prvoplánově to měla být pouze aplikace na zobrazení aktuálních statistik Covid-19, ale to by bylo moc málo náročné, a tak jsem přidal přihlašovací systém, monitorování pacientů atd. S průběhem práce jsem nadmíru spokojen. Po dokončení aplikace jsem vymýšlel následně aktualizace a novinky, kupříkladu obnovení hesla pomocí emailu bylo přidáno v říjnu roku 2021. V době vytváření tohoto dokumentu (23.11.2021) mám již projekt hotov, a tak nemůžu úplně hovořit o průběhu vývoje aplikace, nýbrž budu mluvit spíše o problémech, které nastaly v průběhu vývoje. Cílem aplikace je naučit se nové algoritmy a metody, taktéž bych tímto projektem rád podpořil průběh celosvětového problému s Covid-19.

1 Teoretická část

V této části si popíšeme, co je vlastně Covid-19, použité technologie a použitý software. Dále se také dočtete o problémech, které by mohly nastat během vývoje projektu.

1.1 Onemocnění Covid-19

"Označení koronavirus se používá pro jakýkoli virus, patřící do podčeledi Coronaviridae. Jde o souhrnné označení pro čtyři čeledi virů, které způsobují onemocnění u zvířat a lidí s různým stupněm závažnosti. Název je odvozen od charakteristického uspořádání povrchových struktur lipidového obalu virů ve tvaru sluneční korony. Může způsobit běžné obtíže, jako je nachlazení, kašel, dýchací obtíže, teploty. Ale také smrtící choroby, jako je dýchací onemocnění zvané těžký akutní respirační syndrom (SARS; Severe Acute Respiratory Syndrome) či infekci MERS (Middle East Respiratory Syndrome)."[1]

1.1.1 Nakažlivost

Nakažlivost nemoci Covid-19 je extrémně vysoká. Přenáší se kapénkou (tělesné tekutiny, které vylučujeme z úst při kašli nebo kýchnutí), v menších případech i tělesným kontaktem a v těch největších případech aerosolem. Podle světové zdravotnické organizace je potřeba, aby index nakažlivosti byl nižší než 1 k dosáhnutí zaniknutí nebo aspoň minimalizaci Covid-19. Index nakažlivosti je číslo, které značí kolik lidí nakazí jeden nemocný člověk.

[1] *Koronavirus (Covid-19): Co to je? Jak se přenáší? Jak se chránit? Zastaví jej alkohol a další mýty kolem...* [online]. 2020 [cit. 2021-11-23]. Dostupné z: https://www.patria.cz/zpravodajstvi/4349449/koronavirus-covid-19-co-to-je-jak-se-prenasi-jak-se-chranit-zastavi-jej-alkohol-a-dalsi-myty-kolem.html

1.1.2 Vakcína

Vakcinací obyvatelstva můžeme uskutečnit úplné nebo částečné vymizení nemoci Covid-19. Je uváděno, že k tomu je zapotřebí proočkovanost nejméně 70% obyvatelstva.

"V klinických studiích fáze III prokázalo několik vakcín účinnost až 95 % při prevenci symptomatické infekce covid-19. Národní regulační orgány (alespoň jednoho státu) schválily k dubnu 2021 třináct vakcín pro veřejné použití: dvě RNA vakcíny (vakcínu Pfizer–BioNTech a vakcínu Moderna), pět konvenčních inaktivovaných vakcín (BBIBP-CorV od společnosti čínské společnosti Sinopharm, CoronaVac od čínské společnosti Sinovac, Covaxin od společnosti Bharat Biotech, WIBP-CorV a CoviVac), čtyři vakcíny s virovým vektorem (Sputnik V od ruského Gamalejova institutu, vakcína Oxford–AstraZeneca, Ad5-nCoV od čínské společnosti CanSino Biologics a vakcína Johnson & Johnson) a dvě peptidové vakcíny (ruská EpiVacCorona a čínská RBD-Dimer). Podle Světové zdravotnické organizace bylo k březnu 2021 celosvětově v klinických studiích 73 vakcín, z toho 24 v první fázi, 33 ve fázi I. až II. a 16 v závěrečné, III. fázi.

V Evropské unii, a tedy i v České republice, se k dubnu 2021 očkuje vakcínami BioNTech-Pfizer, Moderna, Oxford-AstraZeneca a Johnson & Johnson."[2]

1.1.3 Symptomy

Mezi častější symptomy řadíme:

- Horečka;
- Kašel;
- Spavost;
- Ztráta chuti nebo čichu.

K těm méně častým bychom mohli zařadit:

- Bolest krku;
- Bolest hlavy;
- Bolesti celého těla;
- Průjem;
- Vyrážka;
- Červené nebo podrážděné oči.

A v neposledním místě jsou vážné, často smrtelné symptomy

- Potíže s dýcháním nebo dušnost;
- Ztráta řeči, pohyblivosti nebo zmatenost;
- Bolest na hrudi.

[2] C*ovid-19* [online]. 2020 [cit. 2021-11-23]. Dostupné z: https://cs.wikipedia.org/wiki/Covid-19

1.1.4 Úmrtnost a smrtnost

"V rané fázi pandemie Světová zdravotnická organizace uvedla odhady MSI mezi 0,3 % a 1 %. V červenci 2020 hlavní vědecký pracovník WHO uvedl, že průměrný odhad expertní fóra WHO pro MSI byl přibližně 0,6 %. V srpnu WHO zjistila, že studie zahrnující údaje ze širokého sérologického testování v Evropě ukázaly, že odhady MSI konvergují přibližně na 0,5–1 %. Na řadě míst jako v New Yorku a italském Bergamu byly stanoveny pevné dolní limity MSI, protože MSI nemůže být nižší než míra hrubá míra smrtelnosti obyvatelstva. K 10. červenci v New Yorku s 8,4 mil obyvateli zemřelo 23 377 jedinců (18 758 potvrzených a 4 619 pravděpodobných) na covid-19 (0,3 % populace). Testování na protilátky v New Yorku odhadlo MSI přibližně 0,9 % až ~ 1,4 %. V provincii Bergamo zemřelo 0,6 % populace. V září 2020 americké Středisko pro kontrolu a prevenci nemocí oznámilo předběžné odhady věkově specifických MSI pro účely plánování veřejného zdraví.

Varianta alfa, která se během zimy a jara 2021 rozšířila a stala se dominantní variantou ve většině evropským zemích, zvyšuje smrtnost o 30 % až 100 %. Varianta delta, která se v létě 2021 masivně šíří Evropou, posílá do nemocnice každé 75. nakažené dítě. "[3]

1.2 Použité technologie

V projektu jsem využil spousty technologií. Některé jsem se i učil nově právě k vývoji projektu. Představme si jen ty, které jsou nejvíce náročné nebo ty, které jsou velmi užitečné do budoucna.

1.2.1 Databázový server

Databázový server slouží k zobrazení a editaci dat kdekoliv a kdykoliv. V mém případě používám databázový server MySQL. Jedná se o jeden z nejpoužívanějších v klasickém programování desktopových aplikací. Komunikace mezi klientem a databázovým serverem probíhá šifrované a pomocí příkazů SQL ("Structured Query Language").

"SQL vznikl v jedné z výzkumných laboratoří IBM, stejně jako teorie relačních databází. Na počátku 70. let, když výzkumníci IBM vyvinuli rané relační systémy DBMS (nebo RDBMS), vytvořili datový podjazyk pro provoz na těchto systémech. Předběžnou verzi tohoto podjazyka pojmenovali SEQUEL (Structured English QUEry Language). Když však přišel čas formálně uvolnit svůj dotazovací jazyk jako produkt, zjistili, že jiná společnost již měla ochrannou známku na název produktu „Sequel". Proto se marketingoví géniové v IBM rozhodli dát vydanému produktu název, který se lišil od SEQUEL, ale přesto byl rozpoznatelný jako člen stejné rodiny. Tak to pojmenovali SQL, vyslovováno ess-que-ell. Ačkoli oficiální výslovnost je ess-que-ell, lidé si zvykli vyslovovat to „Sequel" v prvních dnech před

[3] Covid-19 [online]. 2020 [cit. 2021-11-26]. Dostupné z: https://cs.wikipedia.org/wiki/Covid-19

vydáním a pokračovali v tom. Tato praxe přetrvala až do současnosti; někteří lidé řeknou „Sequel“ a jiní řeknou „S-Q-L“, ale oba mluví o stejné věci.”[4]

„SQL je široce populární, protože nabízí následující výhody:

- Umožňuje uživatelům přístup k datům v systémech správy relačních databází.
- Umožňuje uživatelům popisovat data.
- Umožňuje uživatelům definovat data v databázi a manipulovat s nimi.
- Umožňuje vložení do jiných jazyků pomocí modulů SQL, knihoven a předkompilátorů.
- Umožňuje uživatelům vytvářet a rušit databáze a tabulky.
- Umožňuje uživatelům vytvářet pohled, uloženou proceduru, funkce v databázi.
- Umožňuje uživatelům nastavit oprávnění k tabulkám, procedurám a pohledům.”[5]

1.2.2 Git, aneb verzovací systém

“Vytvoříme program a dovedeme ho funkční fáze. Nyní určitě nechceme při dalších aktualizacích rozbít, co už jsme vytvořili. Jak to ale uděláme? Pomocí kopie celého programu, kterou si uložíme někam bokem? Ano, takto můžeme postupovat, jenže ne vždy se nám povede udělat kopie ve správný okamžik a navíc bychom je dělali často a komplikovali by naší práci. Git toto řeší za nás pomocí tzv. uzlů (z ang. *commit*).

Abych vám mohl vysvětlit princip uzlů, musíme se nejdříve podívat, jak se pomocí gitu projekt ukládá. Místo, ve kterém pracujeme je tzv. *workspace*, tedy naše „pracovní místo“. Jedná se o složku sloužící pro práci s aktuálními soubory. Zde git detekuje provedené změny a jakmile se rozhodneme udělat aktualizaci, tak jen všechny „sbalíme“ do jednoho uzlu a pošleme do repozitáře. Tím uděláme novou verzi našeho projektu, avšak všechny předešlé verze (uzly) jsou nám stále dostupné (neodstraní se). Když tedy uděláme velkou chybu, není problém se vrátit o pár uzlů nazpátek.”[6]

Když tedy máme definici Git, můžeme si shrnout, proč jsem ji využil. Využil jsem přesněji Github, který funguje pod strukturou Git. Github mi tedy umožnil kontrolovat verze a při nalezení nějaké chyby jsem mohl celý kód pouze vrátit zpátky do fáze, kdy celý projekt fungoval tak, jak má. Github a jemu podobné jsou v dnešní době velice důležité, jak pro samotný verzovací systém, tak pro práci v týmu.

[4] Volný překlad - What is SQL? [online]. Allen G. Taylor [cit. 2021-11-28]. Dostupné z: https://www.dummies.com/programming/sql/what-is-sql/

[5] Volný překlad - SQL [online]. [cit. 2021-11-28]. Dostupné z: https://www.tutorialspoint.com/sql/sql-overview.htm

[6] PEKAŘ, Lukáš. CO JE TO GIT A PROČ HO POUŽÍVAT?. Bonsai Development [online]. 16. 09. 2018 [cit. 2022-01-25]. Dostupné z: https://bonsai-development.cz/clanek/co-je-to-git-a-proc-ho-pouzivat

Následný lehký tzv. *code review*, tedy prohlížení kódu ostatních vývojářů, komentování, potvrzování nebo zamítávání.

1.2.3 Rest API

"Rozhraní API neboli rozhraní pro programování aplikací je sada pravidel, která definují, jak se mohou aplikace nebo zařízení vzájemně připojit a komunikovat. REST API je rozhraní API, které je v souladu s principy návrhu REST nebo architektonického stylu reprezentativního přenosu stavu. Z tohoto důvodu jsou REST API někdy označována jako RESTful API."[7]

Každý vyslaný dotaz vrátí vždy kód, který značí, jestli se daný příkaz na serveru vykonal, jestli se k němu vůbec dostal atd. Daný kód má vždy tři čísla, ve kterých je vždy první číslo typ kódu. Pojďme si uvést všech pět typů kódů, které nám může dotazované zařízení zaslat zpět:

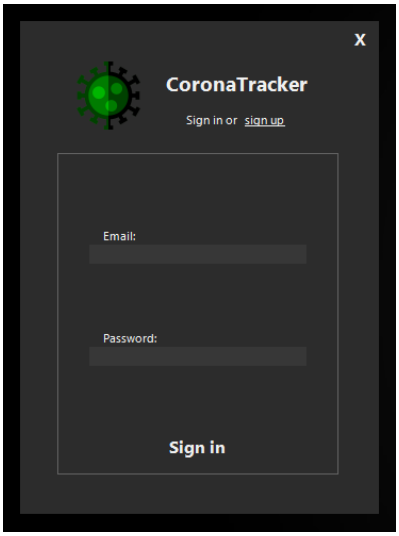
- **1xx** – Provizorní odpověď zařízení. Je jich většinou zasláno více do zdrojového zařízení.
- **2xx** – Značí, že cílové zařízení úspěšně přijalo a vykonalo kód.
- **3xx** – Je zaslán zdrojovému zařízení v případě, že je potřeba pro vykonání kódu udělat ještě nějakou věc. Kupříkladu zdrojový počítač již vyslal jiný požadavek na cílové zařízení, a tak musí počkat na jeho ukončení.
- **4xx** – Chybové hlášky na straně zdrojového zařízení. Například uživatel se snaží poslat příkaz na neexistující URL cílového zařízení
- **5xx** – Jedná se o chybové hlášky na cílovém zařízení. Tedy ty, které neovlivnil, ani nijak neovlivní zdrojové zařízení.

1.2.4 Přihlašovací systém

Přihlašovací systémy jsou v dnešní době na denním pořádku. Když jdeme na Facebook, máme zde přihlašovací systém. Když jdeme na e-mail, máme zde přihlašovací systém. Každý přihlašovací systém je originální, ale základ mají všichni stejný. A to přihlašování, registrování a někdy i obnovení hesla. Můj projekt obsahuje všechny výše zmíněné sekce. V mém případě jsem k celému přihlašovacímu systému použil šifrování SHA256. "Je součástí SHA-2 sady kryptografických hašovacích funkcí, navržených americkou Národní bezpečnostní agenturou (NSA) a publikovaných v roce 2001 NIST jako americký federální standard pro zpracování informací (FIPS). Hašovací funkce je algoritmus, který transformuje (hašuje) libovolnou sadu datových prvků, jako je textový soubor, na jedinou hodnotu s pevnou délkou (hash). Vypočítaná hašovací hodnota pak může být použita k ověření integrity kopií původních dat bez poskytnutí jakýchkoli prostředků k odvození uvedených původních dat. Hodnota hash je nevratná a může být volně distribuována, ukládána a používána pro účely srovnání. SHA je zkratka pro Secure

[7] Volný překlad - What is a REST API? [online]. [cit. 2021-11-28]. Dostupné z: https://www.ibm.com/cloud/learn/rest-apis

Hash Algorithm. SHA-2 obsahuje značný počet změn oproti svému předchůdci."[8]



Obrázek 1 - ukázka mého přihlašovacího okna

Ve svém projektu jsem použil tzv. salty. Jedná se o přidané kódy do šifrování k zamezení nabourání hesel tzv. "brute force" (tvrdé zkoušení hesel). Mám-li například heslo *heslo123* a můj salt bude *sakif45gsd* a dám jej před heslo, budu posílat do šifrovací metody heslo *sakif45gsdheslo123*. Saltů můžu mít, kolik chci a nemají žádnou strukturu. V mém případě používám dva salty a to *6&eL#YwFJFqD*, který přidávám před heslo a druhý *zyQ@^cVX9H67*, který přidávám za heslo. Tedy heslo *heslo123* pomocí mé metody zašifruje jako heslo *6&eL#YwFJFqDheslo123zyQ@^cVX9H67*. Takovéto heslo by počítač musel zjišťovat opravdu dlouho, než by ho zjistil.

1.2.5 Spouštěcí parametry aplikace

Aplikaci je možné zapnout s tzv. vstupními / spouštěcími parametry. Ty slouží k zapnutí programu s nějakými odlišnými funkcemi. V mém projektu jich mám přesně čtyři.

- "-devmode" – slouží k zapnutí programu v programátorském módu. Tedy zapne aplikaci bez nutnosti přihlášení rovnou v administrátorském módu. Tento parametr je rozhodně nebezpečný a v normální aplikaci pro zákazníka bych ho rozhodně neimplementoval.
- "-logoff" – vypne logování aplikace. Není tak nebezpečný jako předchozí parametr, ale taktéž bych jej neimplementoval do výsledného programu pro zákazníka.
- "-showlog" – otevře i s programem logovací okno. Tento parametr, bych taktéž neimplementoval do výsledného programu, ačkoliv není tak

[8] Volný překlad - Ultimate Hashing and Anonymity toolkit [online]. [cit. 2021-11-28]. Dostupné z: https://md5hashing.net/hash/sha256

nebezpečný. Pouze by uživatel viděl více do pozadí celého programu, než by potřeboval.

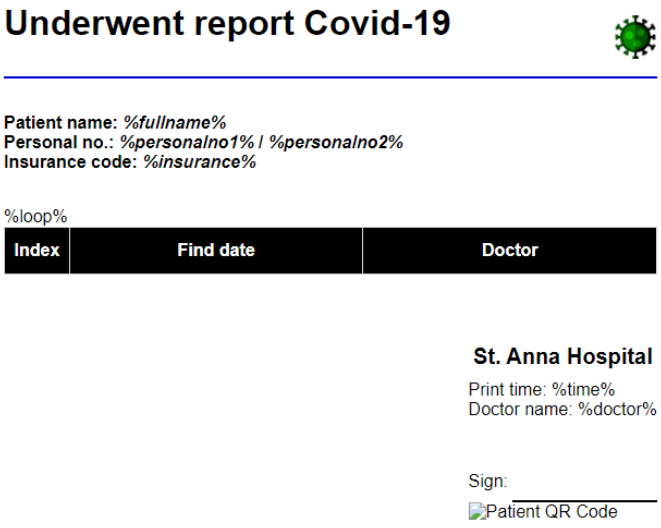
- "-vx.x.x" – spustí program s uvedenou verzí. Místo "x" je třeba doplnit konkrétní verze, příp. podverze. Prvomyšlenkově je tento parametr nevinný, ačkoliv při správném verzování by mohlo dojít k problémům. Na příklad kdybych změnil strukturu databáze a uživatel by si na svém počítači spustil program se starou strukturou s tímto parametrem, mohlo by dojít k poškození jak programu, tak i databáze jako takové.

1.2.6 Logování důležitých eventů

Celá aplikace zálohuje důležité momenty pro následnou správu a hledání případných chyb do souboru. Celý logovací systém je autonomní a umí automaticky rozpoznat z jaké metody, souboru a dokonce řádku byl daný log zavolán. Tedy formát jednoho záznamu vypadá takto "[26/11/2021 22:00:28] Program.cs:Main:31 » Creating discord webhook instance". Z daného řádku můžeme přečíst, kdy byl daný záznam vytvořen. Dále z jakého souboru, metody a řádku byl záznam vytvořen. A jako poslední máme samotnou zprávu. Momentálně je i v plánu přidání typu záznamu, jestli se tedy jedná o upozornění nebo jen informativní zprávu.

1.2.7 Export dat pomocí PDF

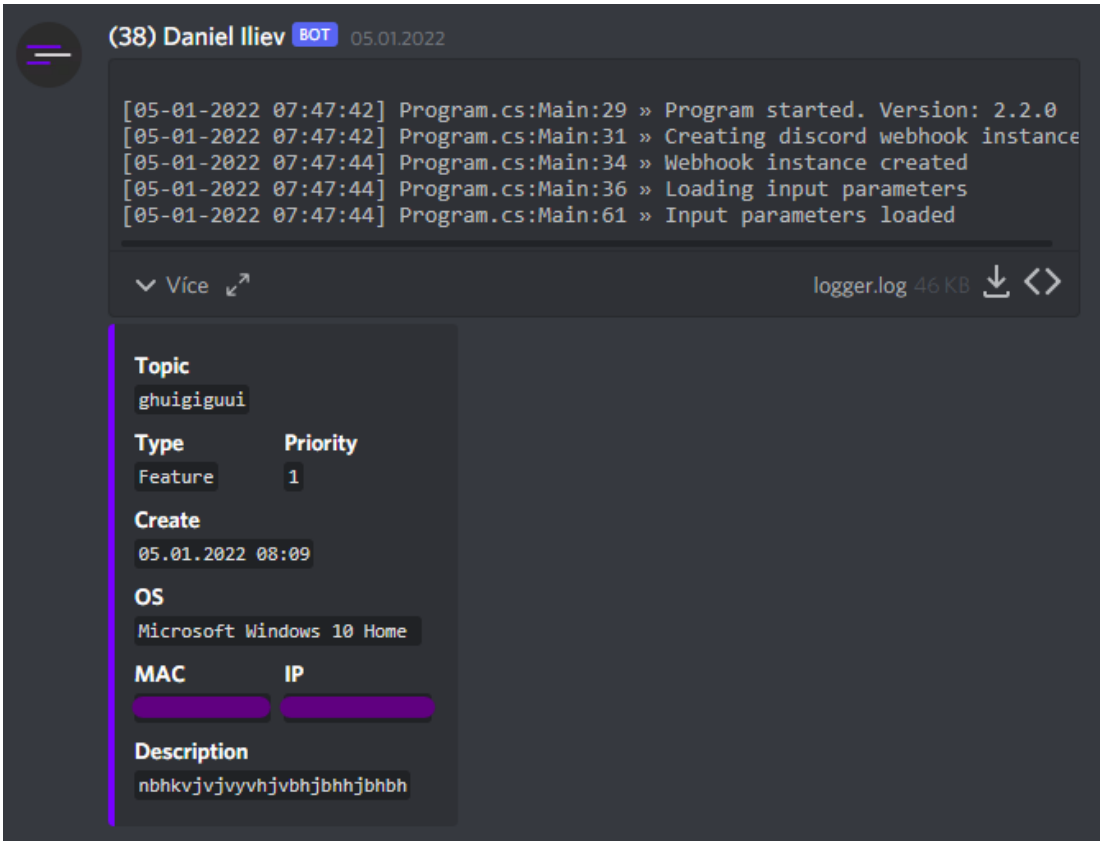
Aplikace umí exportovat potřebná data jako PDF. Například lze vygenerovat a vytisknout nálezy Covid-19 pro daného pacienta. V dokumentu se nachází důležitá data, jako pacientovo jméno, rodné číslo a číslo pojišťovny. Dále přímo nálezy a jejich indexy, datum nálezu a jméno doktora, který jej nalezl. Jako poslední informace jsou čas vygenerování / vytisknutí, jméno doktora a QR kód pacienta. Export probíhá vytvořením HTML kódu, vyplněním potřebných dat a následného exportování jako PDF.



Obrázek 2 - vzor stránky pro generování výstupních dat

1.2.8 Nahlašovací systém chyb

Aplikace umí automaticky nahlásit chyby pomocí tlačítka a následného zadání potřebných údajů. Automaticky se rovnou odešle i celý log a informace jako IP adresa nebo MAC adresa. Celý report se odesílá na komunikační server Discord a to pomocí metody Rest API. Samozřejmě tyto zprávy jsou v soukromém kanálu, do kterého mají přístup pouze ověření uživatelé. Taktéž se zpráva automaticky upraví profilovou fotografií přihlášeného doktora, jeho jméno a příjmení a jeho ID. Níže je náhled testovacího nahlášení. Můžeme zde vidět i kousek odeslaného logu.



Obrázek 3- nahlášení na komunikační aplikaci Discord

1.2.9 Instalátor aplikace

Aplikace má vlastní instalační aplikaci, ve které si může uživatel nastavit, chce-li aplikaci nainstalovat pouze pro svůj účet nebo pro všechny účty. Dále si může nastavit cílovou lokaci aplikace. Instalátor aplikace není povinný, ale rozhodně je velice užitečný.

2 Praktická část

V této části popíši veškerý vývoj aplikace. A to jeho části, problémy a nově naučené technologie, které jsem se naučil během vývoje.

2.1 Základní myšlenková mapa

Jak jsem zmínil již v úvodu tohoto dokumentu, aplikaci jsem začal vyvíjet z důvodu aktuální pandemie Covid-19. Taktéž jsem zmínil, že mám rád výzvy, a to určitě podpořím, jak využitými zdroji, tak přímo zde v praktické části s implementací technologií. První myšlenka aplikace byla zobrazování aktuálních dat o Covid-19 a to přesněji potvrzené případy, úmrtí, počet uzdravení a počet hospitalizovaných. Samozřejmě by šlo v aplikaci vybrat přímo stát, pro který by se data zobrazovala. Po dlouhodobém přemýšlení mi to přišlo jako extrémně jednoduchá aplikace a nebyla by to potom výzva. Nastává tedy další myšlenka aplikace. Nemocniční aplikace pro správu pacientů a jejich data s tématem Covid-19, a abych zachoval i minulou myšlenku, dám do aplikace sekci s aktuálními daty Covid-19. Vyplývá nám tedy aplikace, ve které budou sekce:

- "Home" – Základní sekce aplikace. Jsou v ní uvedeny úplně základní informace o aplikaci. Tedy: Covid data v České republice, datum a čas, použité zdroje, autor aplikace a aktivní zaměstnanci (doktoři), kteří se aktualizují automaticky přímo přes databázi;
- "Dashboard" – Zde aplikace zobrazuje pár souhrnných dat o pandemii Covid-19. Přesněji: počet registrovaných pacientů, počet vakcinovaných pacientů, graf potvrzených pacientů s Covid-19 za posledních šest měsíců, počet aktuálně potvrzených pacientů a celkem nálezů Covid-19;
- "Countries" – V této sekci se ukazují aktuální data ve všech dostupných státech poskytovaných třetí stranou, ze které získávám data;
- "Patient" – Hlavní sekce celé aplikace. Pracuje se zde s pacienty, jejich osobními informacemi, ale taktéž s jejich proděláním Covid-19 a zda byli očkováni. Tato sekce se rozděluje na další tři pod sekce:
    - "List" – přidávání, odebírání a editace pacientů a jejich osobních informací;
    - "Finds" – přidávání, odebírání a editace nálezů onemocnění Covid-19 pacientů;
    - "Vaccinations" – přidávání, odebírání a editace očkování pacientů;
- "Vaccine" – Zde aplikace umožňuje přidat, odebrat a editovat typy vakcín pro následné implikování a přidání pro pacienta;
- "Settings" – V této sekci si může uživatel přenastavit svou aplikaci a jeho uživatelská data. Lze tedy nastavit:
    - Profilovou fotku;
    - Telefonní číslo uživatele;
    - Změna hesla;

- Automatické přihlašování více v kapitole **2.6.1 Automatické přihlášení**;
- V případě povolených práv může uživatel měnit práva jiných uživatelů.

Dále se zde nachází další tři okna, která nespadají strukturou pod hlavní uživatelské rozhraní a jimi jsou:

- Přihlašovací okno – slouží pro přihlášení, registraci a obnovení hesla;
- Nahlašovací okno – slouží pro nahlášení chyby, upozornění na nedostatky nebo návrh zlepšení;
- Logovací okno – slouží k zobrazení tzv. logů v reálném čase. Zobrazuje se pouze se spouštěcím parametrem "-showlog".

2.2 Vývoj databáze

Základní myšlenku aplikace máme. Teď je na řadě vytvoření databáze. Mnozí programátoři by se mnou nesouhlasili v postupu, ale dle mého si při vytváření databáze můžeme vzpomenout na více informací a dalších sekcí, které bychom rádi do celé aplikace implementovali. Další věc, kterou by mi jistě spousty programátorů vytklo je, že první skriptuji databázi a až poté dělám ER diagramy. Veškeré struktury databází jsem skriptoval v jazyce SQL na vlastním serveru MySQL. Pro rychlé znázornění jazyka SQL bych rád ukázal příklad vytvoření entity – tedy "tabulky".

```
1 CREATE TABLE IF NOT EXISTS ProgramData(
2     ProgramData_Key varchar(256) NOT NULL PRIMARY KEY,
3     ProgramData_Value varchar(2048) NOT NULL
4 );
```

Obrázek 4 - skript pro vytvoření entity

Po vytvoření entity s názvem „Program data", která obsahuje dva atributy, takovouto entitu ve svém programu používám k uchování důležitých dat, které nepotřebují vlastní entitu. Kupříkladu k aktuální verzi. Tedy kdybych chtěl do této entity vložit data o aktuální verzi, vypadal by skript následovně. Kde „Version" zastupuje atribut „ProgramData_Key" a „3.1.0" pro „ProgramData_Value".

```
1 INSERT INTO ProgramData
2 VALUES('Version', '3.1.0');
```

Obrázek 5 - skript pro zapsání verze do entity

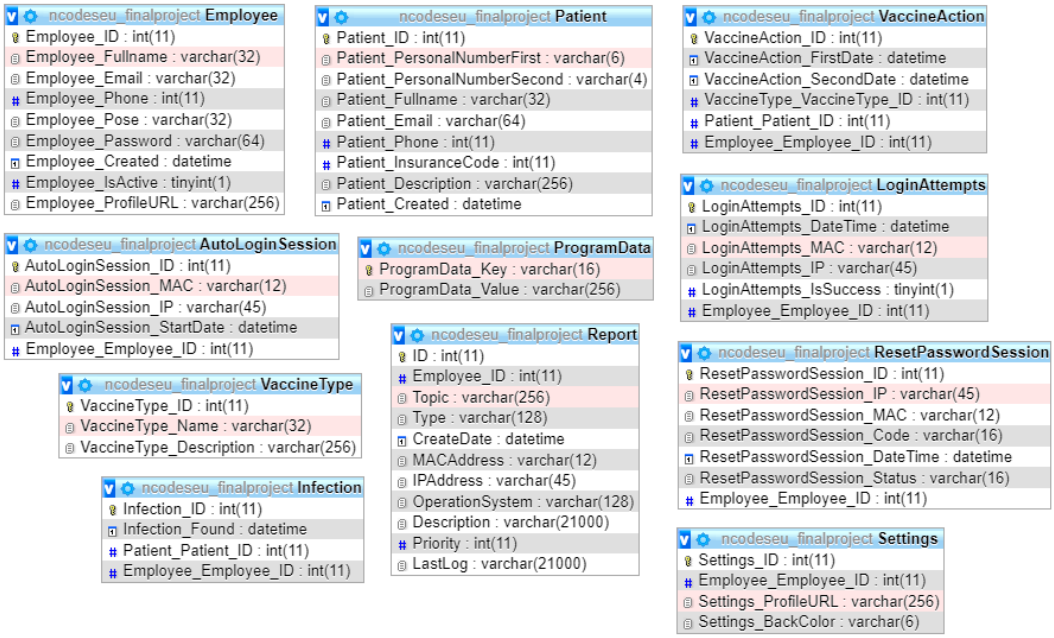
První zpracování databáze obsahovalo osm entit. Přesněji:

- Patient – ukládá data o pacientovi;
- Employee – ukládá data o uživatelích;
- LoginAttempts – ukládá data o pokusech o přihlášení;

- Infection – ukládá data o nálezech onemocnění Covid-19;
- Settings – ukládá nastavení jednotlivých uživatelů (již není);
- VaccineType – ukládá typy vakcín;
- VaccineAction – ukládá vakcinaci pacientů;
- ProgramData – ukládá ostatní data, která by nepotřebovala mít vlastní entitu.

Tato struktura ale neposloužila svým účelům zcela, jelikož přibývalo novinek, které jsem v aplikaci chtěl. Postupem času byla struktura upravována a toto je její výsledný vzhled. Struktura databáze proběhla do další úrovně. V další verzi přibyla nová entita „AutoLoginSession“, která umožnovala uživatele automaticky přihlašovat. Více v kapitole **2.6.1 Automatické přihlášení**.

Finální podoba databáze je o další dvě entity větší. Přesněji řečeno o „Report“, který umožňuje uchování dat nahlašování uživatelů více v kapitole **2.6.5 Nahlašovací systém**. Druhou je „ResetPasswordSession“, do kterého se ukládají data o obnovení hesel.

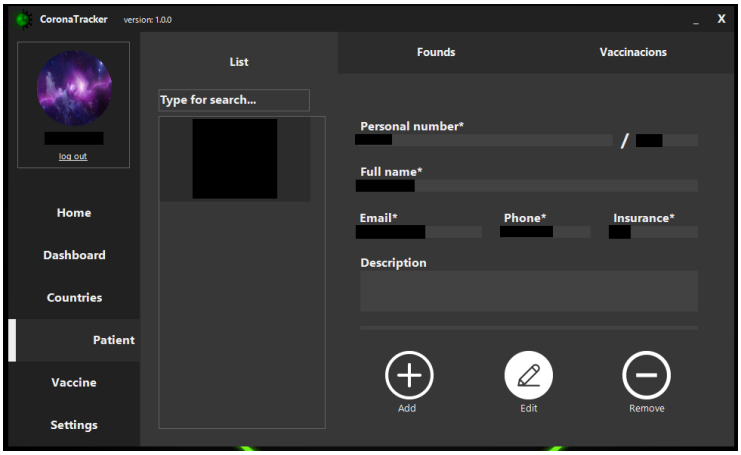


Obrázek 6 – Finální ER diagram databáze

2.3 Vývoj vzhledu aplikace

Vzhled aplikace je velice důležitý a není radno jej odbýt. Vývoj vzhledu dělám jako druhý krok, jelikož zde si opět mohu domyslet další nedostatky aplikace. V celkovém vzhledu aplikace se neděly tak zásadní změny jako v databázi, ačkoliv jich zde proběhlo mnoho. Vyčtu zde jen ty nejdůležitější.

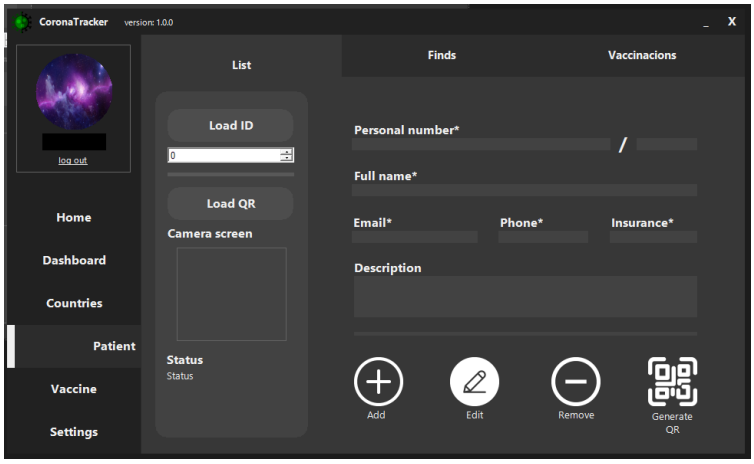
První vzhled aplikace, který již udává, jakým grafickým směrem bude aplikace směřovat.



Obrázek 7- první grafický návrh aplikace

Můžeme zde vidět tmavé pozadí s bílým fontem. Hlavní lišta se nachází nahoře, kde můžeme vidět logo aplikace, název, aktuální verzi a tlačítka na minimalizování aplikace a její vypnutí. Na levé straně můžeme vidět hlavní menu, na kterém se nachází mimo hlavních sekcí i profilová fotka uživatele, jeho jméno a tlačítko odhlásit se. Fotka je ze sekce pacientů a podsekcí list pacientů. Je zde vidět starý způsob vyhledávání a vybírání pacientů, které bylo později nahrazeno QR čtečkou.

Finální grafická podoba té samé sekce je taková to.



Obrázek 8 - finální grafický návrh aplikace

Je tedy obohacena QR čtečkou místo zdlouhavého vybírání ze seznamu. Dále je možnost vybrat uživatele pomocí jeho registračního ID. Taktéž je zde navíc tlačítko pro odeslání QR kódu přímo pacientovi na email.

Aplikace není responzivní a došlo tak k vytvoření mnoha problémů, které se opravovaly přibližně měsíc. Je zajímavé, jak naprosto triviální chyba může způsobit takové problémy. Ale více v kapitole **2.7.4 Problémy s responzivitou**.

2.4 Vývoj funkční stránky aplikace

Funkční stránka aplikace je dle mého nejdůležitější součástí vývoje aplikace. Ačkoliv tuto stránku vývoje běžní uživatelé nevidí, bez ní by vůbec nic nemohlo fungovat. Je proto potřeba před vývojem tzv. backendu vědět, do čeho jdeme, a proto je třeba mít hotové nějaké myšlenkové mapy aplikace a příp. i databázi.

2.4.1 Příprava struktury aplikace

Aby se v aplikaci nevyznal jen autor algoritmů, je potřeba aplikaci nějakým způsobem strukturovat. Vytvoříme si tedy základní strukturu:

- "Database" – věci, které budeme používat pro práci s databází;
- "Enums" – speciální proměnné, které v sobě můžou ukládat pouze specifické názvy. Pro znázornění pro mou aplikaci mám enum "EmployeePoseEnum", který zajišťuje, jaký má daný uživatel pozici. Můžu do daného enumu vložit pouze hodnoty:
    - "User" – uživatel bez jakéhokoliv přístupu. Kdokoliv se může do aplikace registrovat a používat úplně základní údaje;
    - "Employee" – zaměstnanec / lékař. Má přístup do většiny aplikace;
    - "Leader" – vedení nemocnice. Má přístup do celé aplikace, včetně úpravy rolí ostatních uživatelů;
    - "Developer" – vývojář aplikace – Stejný přístup jako "Leader";
    - "Null" – slouží pro errorové statusy.
- "Instances" – pro vlastní proměnné, do kterých mohu ukládat vlastní data podle vlastní struktury;
- "Resources" – pro uložení veškerých lokálních grafických zdrojů, z kterých aplikace čerpá;
- "SubForms" – všechny Formuláře (grafická uživatelská rozhraní), které aplikace využívá. Kromě hlavního Formuláře s názvem UI;
- "Timers" – časovače, které slouží k spuštění nějakého algoritmu za nějaký interval. V mém případě slouží například k automatickému odhlášení po čtyřech hodinách neaktivity nebo k zobrazování aktuálního času v hlavní sekci;
- "Utils" – ukládáme veškeré věci, které nespadají do ostatních sekcí.

Dále si začneme tvořit jednotlivé instance pro databáze a jejich následné mapování. Tedy pro každou entitu musíme vytvořit instanci odpovídající atributům dané entity. Taktéž bychom měli mít již plně zaplněný "SubForms", jelikož vývoj vzhledu aplikace máme již za sebou.

2.4.2 Propojení pozadí s popředím

Nyní následuje druhá fáze vývoje, kdy musíme propojit vzhled / popředí aplikace (tzv. front-end) s pozadím aplikace (tzv. back-end). Jedná se o vytvoření algoritmů například pro zmáčknutí tlačítka, změny aktuální sekce, odesílání emailů, generování PDF souborů, atd.

Začněme přímým propojováním popředí a pozadí aplikace, tedy zprovozněním přepínání sekcí. Není to nic složitého, pouze je potřeba přepnout podsekci z minulé na další. V předchozích verzích se jednalo o načtení úplně nově vypadající podsekce, ale v posledních verzích bylo přidáno ukládání dat do paměti (tzv. "cache system"), více v kapitole **2.6.6 Ukládání do paměti RAM**.

Další na řadě je vytvoření ostatních algoritmů. Například odesílání e-mailu bylo zajímavé programování, jelikož je potřeba, aby taková metoda byla asynchronní (může běžet v pozadí). Kdyby nebyla, tak by se program zasekl do té doby, dokud by e-mail nebyl odeslaný. To rozhodně nechceme, a tak tedy přichází na řadu tzv. "multi-threading", který značí to, že program pracuje s více vlákny. Tedy nezamrzne při delším zpracování algoritmů.

```
25      /// <summary>
26      /// Function to write an email async
27      /// </summary>
28      /// <param name="aID"> variable for user's ID </param>
29      /// <param name="afirstPersonalNumber"> variable for first personal number </param>
30      /// <param name="asecondPersonalNumber"> variable for second personal number </param>
31      /// <param name="amailTo"> variable for email write to </param>
        1 reference
32      public static void AsyncWrite(int aID, int afirstPersonalNumber, int asecondPersonalNumber, string amailTo)
33      {
34
35          ID = aID;
36          firstPersonalNumber = afirstPersonalNumber;
37          secondPersonalNumber = asecondPersonalNumber;
38          mailTo = amailTo;
39
40          Thread mailer = new Thread(new ThreadStart(Write));
41          mailer.Start();
42      }
```

Obrázek 9 - asynchronní metoda pro odesílání e-mailu

2.4.3 Propojení aplikace s databází

Poslední fází vývoje je propojení aplikace přímo s databázovým serverem. Jestliže jsme nepřeskočili žádnou fázi, tak implementace bude jednoduchá. Jen je zdlouhavá, jelikož musíme při každém načtení Formuláře aktualizovat data přímo z databáze. Taktéž musíme aktivovat přihlašovací systém, který se v době prvotního vývoje aplikace nachází pouze ve fázi přihlašování – tedy žádná registrace, žádné obnovení hesla, žádné automatické banování účtů atd.

2.5 Testování aplikace

Testování aplikace je většinou nejdelší proces celého vývoje aplikace. I v mém případě testování aplikace bylo největší částí celého vývoje. Kvůli testování jsem i přidal nahlašovací systém, více v kapitole **2.6.5 Nahlašovací systém**. Průběh testování v mém případě probíhal jednodenním testováním a následným dvou až čtyř

denním opravováním chyb nebo přidáváním novinek. Celé testování jsem dělal synchronně s aplikací Trello, která umožňuje zapisovat poznámky a zapisovat celý vývoj aplikace. Přímo přes aplikaci jsem si tedy mohl psát progres. Celý testovací proces trval přes tři měsíce. Pro upřesnění testování v mém případě beru od verze 1.1.0.

2.6 Implementace nových funkcí

Od procesu testování byly přidávány nové funkce. Některé z nich stojí za zmínění. Přidávání bylo v mnoha případech velice obtížné, ale při implementaci většiny jsem se naučil spoustu nových funkcí a způsobů, jak programovat. Při práci na projektu jsem se mnohé naučil, a tak se i můj princip programování trochu pozměnil, což je vidět ve výsledném kódu.

2.6.1 Automatické přihlášení

Automatické přihlášení by nemělo chybět v žádné aplikaci, která obsahuje přihlašovací systém, a proto jsem se rozhodl ji implementovat i do svého projektu. Implementace byla obtížnější, jelikož jsem musel vymyslet, jak přesně povolit specifickému uživateli se přihlásit automaticky. Je zde mnoho metod. Například uložení emailu a hesla lokálně do počítače a následně načtení při zapnutí aplikace. Nebo zápisem do databáze: MAC adresa, IP adresa a uživatel, kterého si má databáze pamatovat. V mém projektu jsem použil druhou metodu. Myslím si, že je bezpečnější, jelikož veškerá komunikace probíhá přímo se serverem, a tudíž není nic důležitého uloženého přímo u uživatele v počítači. Jak bylo zmíněno výše, implementace byla složitější, jelikož bylo potřeba vytvořit entitu v databázi a následně zakomponovat automatické přihlášení hned po zapnutí aplikace. Zapnutí automatického přihlášení se nachází v sekci “Settings”, kde jej můžete zapnout jediným tlačítkem a pak jej jen potvrdit.

2.6.2 Obnova hesla pomocí kódu z emailu

Další z funkcí, která stojí za zmínění. Obnova hesla je další z funkcí, kterou by měla obsahovat každá aplikace, co umožňuje se zaregistrovat a používat aplikaci bez jakéhokoliv zasáhnutí administrátorem. Implementace byla v tomto případě opět těžší, jelikož jsem pro naprosté zabezpečení uživatelova účtu přidal entitu do databáze. Stačilo by pouze vytvořit lokální proměnnou s kódem, který je potřeba zadat pro úspěšnou obnovu hesla. Ale i s menšími znalostmi nabourávání aplikací lze toto nabourat, a to jednoduchým programem “CheatEngine”, který se nabourává do ostatních aplikací a mění lokální proměnné programů. Tedy by se změnil kód na jakýkoliv a nemusel by se zadávat ten, který přišel cílenému uživateli na email. Proto zadávám kód i do databáze a musí se po následném zadání shodovat nejen s lokální proměnnou, ale i s výše zmíněným prvkem v databázi. Taktéž z bezpečnostních důvodů ukládám data jako z které IP adresy a MAC adresy byl požadavek odeslán pro následné dohledání uživatele, který chtěl změnit jinému uživateli heslo. Obnovu hesla je možné provést pouze v přihlašovacím okně v sekci “Forgot your password”

a provádí se v třech krocích. Krok první je zadání e-mailu cíleného uživatele. V tomto kroku se aplikace snaží získat ID uživatele, kterému chceme obnovit heslo a následně uložení jak kódu, tak všech ostatních důležitých informací do databáze. Dalším krokem je zadání daného kódu, který má cílený uživatel na e-mailu, do aplikace a opětovné potvrzení. Posledním krokem je přímé zadání nového hesla. Opět klikneme na tlačítko a máme resetované heslo.

2.6.3 Spouštěcí parametry

Tato funkce by se přímo ve výsledném programu moc nevyužívala, jelikož je spíše pro zběhlé uživatele a administrátory aplikace, ale má své výhody ji mít. Kupříkladu je možné aplikaci spustit v režimu ukazování logového systému přímo v reálném čase na dalším okně. Všechny spouštěcí parametry jsem již vypisoval výše, více informací v sekci **2.2.4 Spouštěcí parametry aplikace**. Implementace nebyla vůbec složitá, jelikož je to pouze vstupní parametr hlavní metody, který musíme zkontrolovat jakou má podobu. Nejtěžší z celé implementace bylo vytvoření logovacího okna, jelikož je potřeba synchronizovat více procesů na jednou. V tomto případě to nebylo ale nic těžkého, jelikož synchronizace Formulářů je dosti jednoduchá a přizpůsobená jednoduché implementaci.

2.6.4 Logovací systém

Velice důležitá funkce pro správný průběh nahlašování chyb. Díky celému logovacímu systému je možné přijít na mnoho chyb, jak na straně vývojáře, tak na straně uživatele, který mohl jen něco špatně zadat. Původně jsem zamýšlel využít již existující autonomní systém Log4net2, ale v celém logovacím systému se našly fatální chyby, které umožňovali běžným uživatelům ovládat pozadí celé aplikace bez jakýkoliv přidaných práv. Tento systém jsem tedy zavrhnul a byl jsem nucen si vytvořit svůj vlastní.

```
99+ references
public static void Log(string value, [CallerLineNumber] int lineNumber = 0, [CallerMemberName] string caller = "null", [CallerFilePath] string file = "null")
{
    string line = $"\n[{DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss")}] {file.Substring(file.LastIndexOf('\\') + 1)}:{caller}:{lineNumber} » {value}";
    log += line;

    if (DoLogForm)
    {
        if(Application.OpenForms.OfType<LogForm>().FirstOrDefault() != null)
            LogForm.Log(line);
    }

    index++;
    if(index == 10)
    {
        index = 0;
        Save();
        log = "";
    }
}

4 references
public static void Save()
{
    writer = new StreamWriter("logger.log", true);
    if(log.Length > 2)
        log.Substring(1);
    writer.Write(log);
    writer.Flush();
    writer.Close();
}

1 reference
public static string GetLog()
{
    return log;
}
```

Obrázek 10 - ukázka vlastního logovacího systému

Logovací systém funguje na tzv. stackování (vkládání několika hodnot na sebe) do maximálního čísla. V mém případě deset, kde po dosažení daného čísla, dojde k uložení všech dat, která byla nastackována. Tento způsob jsem si zvolil pro jeho nenáročnost k zasahování do souborů, jelikož kdybych ukládal po každém řádku, mohlo by dojít k problému s právy na soubor (stává se v případech, kdy program rychle zapisuje do souboru, tak v systému zůstane, že do souboru někdo / něco zapisuje, a tak nepovolí někomu / něčemu jinému do něj zapisovat). Logovací soubor byl vyvíjen jako jeden z prvních funkcí navíc a nyní bych ho naprogramoval jinak, lépe. Například ve starém případě se nejedná o úplné stackování, jelikož zde data ukládám do jedné proměnné. Dnes bych to ukládal do pole a pak s tím mnohem jednodušeji pracoval.

2.6.5 Nahlašovací systém

Nahlašovací systém, který lze otevřít kdykoliv, většina aplikací nemá. Myslím si, že je to škoda. Důvodem, proč tyto systémy neimplementují do svých projektů, je z bezpečnostního hlediska serveru. Protože při jakémkoliv nahlášení dojde k propojení se serverem a uložení dat o nahlášení. Dle mého se při správném zabezpečení nemůže přehltit server. Kdybych uměl dělat webové stránky, rozhodně bych naprogramoval správný postup komunikace mezi serverem a aplikací. Bohužel mé znalosti sahají pouze do vývoje aplikací, a tak tedy musím celý proces nechat na databázi. Není to tak bezpečné, jelikož uživatel stále komunikuje s databází přímo. I v databázi se však dají vytvořit jistá pravidla pro uložení. Jak jsem výše zmiňoval, průběhem vývoje jsem se naučil spousty nových způsobů programování, a tak bych ověření pro nahlášení již dělal jiným způsobem. Jediné ověření, které se teď v aplikaci nachází, je možnost otevření nahlašovacího okna pouze jednou od spuštění aplikace. Uživatel tedy nemůže načíst více nahlašovacích oken najednou. Ale rozhodně to není správný postup na řešení nahlašovacího systému. Mít více času tato funkce by byla jistě první, kterou bych předělal.

2.6.6 Ukládání do paměti RAM

Ukládání do paměti RAM, neboli tzv. “cache” systém, je v dnešní době velice důležitý jak ze strany uživatele, tak ze strany vývojářů, a hlavně jejich serverů. Vezměme si příklad přímo z aplikace. Máme zde seznam států a aktuální informace o stavu Covid-19 v nich. Každé načtení seznamu by znamenalo dotázání se třetí strany, kterou používám pro získávání obrázků vlajek, na každou zemi (v mém seznamu zhruba 50) a jejich vlajku. Pokud by uživatel byl takový, který rád testuje a ničí aplikace, rozhodně by se pokusil o mačkání tlačítka opakovaně za sebou. Mohlo by tak dojít k odesílání několika desítek dotazů za sekundu. Třetí strana takovéto situace většinou řeší dočasným, případně permanentním zakázáním využívání celé třetí strany a jakékoliv komunikace s ní.

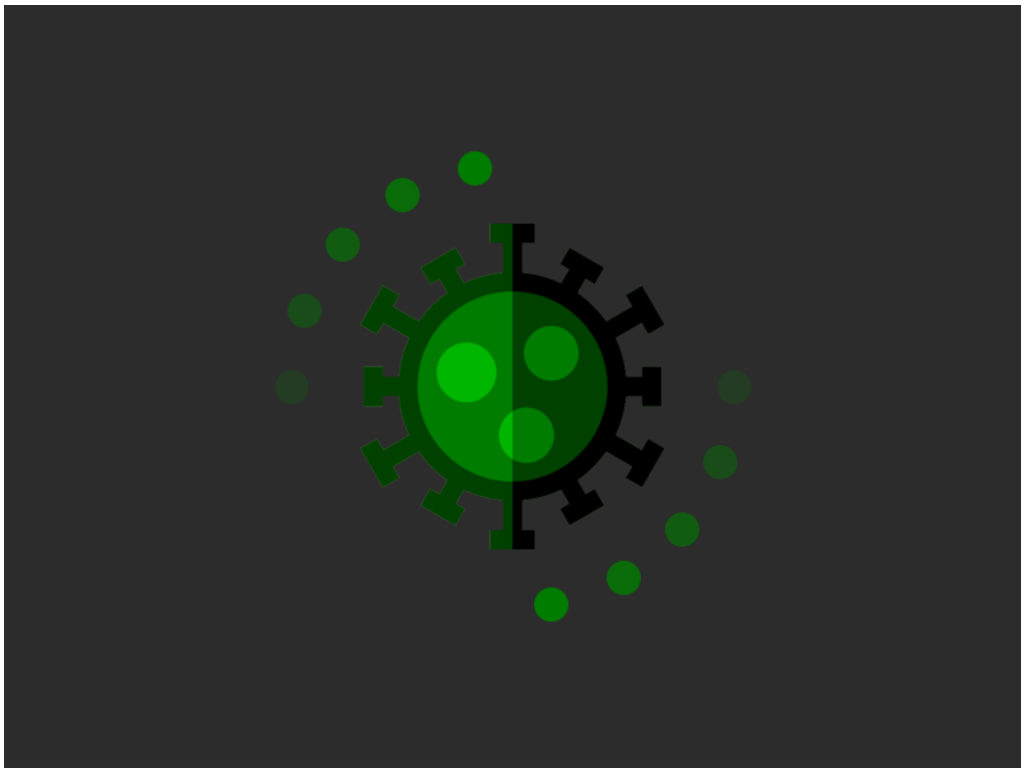
Rozebrali jsme se si tedy, jeden z mnoha problémů nepoužívání cache systému. Teď k tomu, co vlastně celý ten systém je. Jedná se o ukládání důležitých a časově

trvalých dat do paměti RAM, která jsou buď velice důležitá a musí se obnovovat anebo ta, která se nemusí aktualizovat tak často. K tomuto dochází většinou automaticky při použití proměnných, ale jde i o jejich zachování po přesunutí někam jinam, s možností navrácení se zpět ke starým datům.

Implementace cache systému je složitá, jelikož se kvůli němu musí předělat celý mechanismus programu a jeho fungování. Díky tomuto procesu se zapínání aplikace z jedné sekundy zpomalilo na deset a více sekund. Došlo tedy k ohromnému znásobení zapínací doby, ale přepínání mezi jednotlivými sekcemi a daty z třetích stran se z pohledu uživatele načítají daleko rychleji. Z pohledu vývojáře je ulehčení třetích stran a na straně vlastní databáze, neboť se na servery posílá daleko méně požadavků o data. Obsah celého kódu se znásobil a došlo k celkové změně fungování načítání pod formulářů. Kupříkladu musel jsem změnit otevírání vždy nového okna pod formuláře, který zabere jeden řádek, na otevírání pod formuláře, kterého musím nalézt uloženého v cache systému. Taktéž jsem musel nějak naplnit daný systém. Jedná se celkově o funkci, kterou uživatelé buď vůbec nepostřehnou anebo jim to naopak zpomalí načítání kódu, ale s výsledným rychlejším přepínání a načítání celkově v aplikaci.

2.6.7 Načítací obrazovka při načítání aplikace

V minulé kapitole jsme si pověděli o tom, proč a jak jsme vytvořili cache systém. Ale jelikož se nám zvedl načítací čas z jedné sekundy na deset a více sekund, bylo potřeba vytvořit načítací obrazovku. Jelikož při načítací fázi se uživateli nic nezobrazilo, a tak si mohl uživatel myslet, že se ani nic neděje. Mohlo tak dojít k opakovanému zapínání programu, příp. nahlášení jako chyby.



Obrázek 11 - úvodní načítací stránka

Celá implementace načítací obrazovky se jevila jako velice jednoduchá. Je to přeci jen okno, které zobrazuje gif obrázek a při načtení se dané okno zavře. Pravda byla ale úplně opačná. Musel jsem zesynchronizovat vícero jader, a tentokrát jsem opravdu sám musel pracovat s jádry v pravém slova smyslu. Rád bych dodal, že vyřešení problému práce s vícero jádry je vysokoškolské studium a jedná se tak o velice náročnou práci. Načítání jsem tedy vložil do speciální třídy, kterou jsem spustil v hlavním jádře. Před hlavním načítáním jsem spustil a uložil si do cache systému jádro s načítacím oknem. A při načtení celé aplikace jsem pouze zavřel a zastavil vlákno s načítacím formulářem.

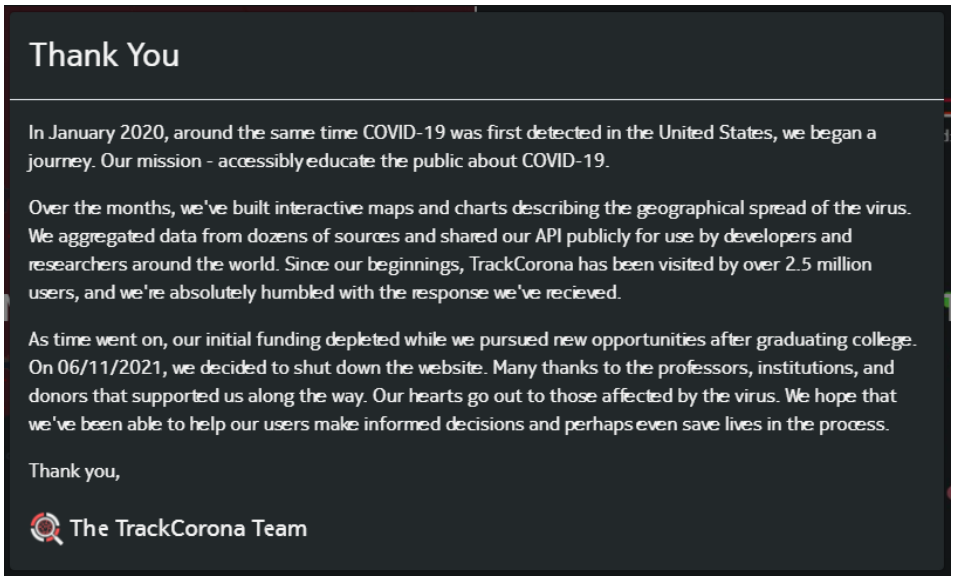
2.7 Problémy s vývojem

Během vývoje aplikace došlo k mnoha problémům v celé aplikaci a okolo ní – tedy vývoje. Za některé problémy rozhodně mohu já a za některé nemůžu tak úplně já – jednalo se především o chyby třetích stran. Řešení těchto problémů mnohdy znamenalo mnoho hodin neskutečného hledání chyby a následného opravení.

Samozřejmě zde nejsou uvedeny všechny problémy, které během vývoje nastaly. Jsou zde uvedeny jen ty nejdůležitější nebo ty, které zasáhly nejvíce celý průběh vývoje.

2.7.1 Ukončení třetí strany pro získávání dat o Covid-19

Aplikace umí získávat data v reálném čase o pandemii Covid-19 ze států dostupných třetí stranou, ze které data získávám. Při implementování této funkce vše fungovalo, jak mělo, dokud se třetí strana nerozhodla zastavit dodávání informací. Musel jsem tedy najít jinou stranu, která byla ochotná a schopná dodat mé aplikaci potřebná data, která by následně zobrazovala uživatelům. Taktéž jsem musel brát v potaz, že spousta třetích stran dodávajících data v reálném čase, jsou placené, a tak jsem si je nemohl dovolit využívat. Naštěstí jsem tedy nakonec našel třetí stranu, která mi dovolovala získávat data a nic neplatit.



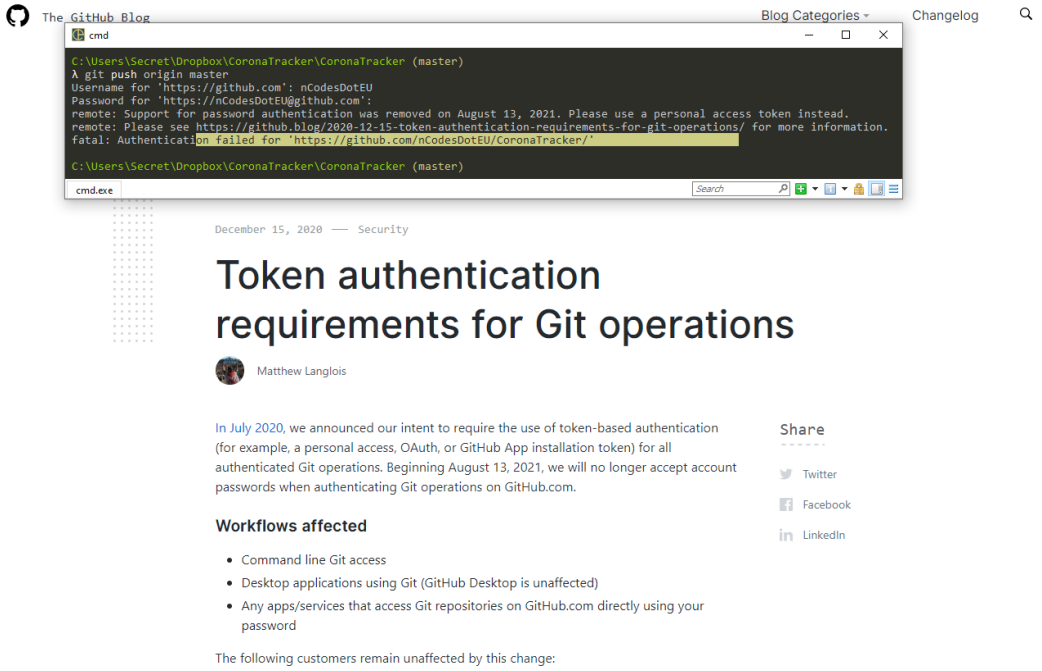
Obrázek 12 - poděkování třetí strany za používání jejich služeb

2.7.2 Git a problém s třetí stranou

Celá aplikace, jak jsem již zmínil výše, funguje společně s Githubem, více v kapitole

1.2.2 Git, aneb verzovací systém. Aplikaci jsem propojoval ručně přes příkazový řádek, který napodoboval chování jako v operačním systému Linux. Bez jakýchkoliv nutných instalací mi umožňoval využívat služeb Git, tedy i Github. Bohužel mi toto bylo zakázáno z důvodu nedostatečných práv. Jelikož Github vytvořil systém pro správu práv, tak jsem si buď musel přidat práva pro úpravu daného projektu přímo z aplikace třetí strany nebo si stáhnout jejich aplikaci. Vzhledem k tomu, že jejich aplikace byla rychlejší a lepší řešení, rozhodl jsem se o využití právě aplikace Github Desktop, která mi umožňovala vše, co příkazový řádek, a i mnoho dalších funkcí navíc. Nakonec jsem zjistil, že prostředí aplikace Github Desktop je i efektivnější vzhledem k výkonu a času. Všechno se v aplikaci dá krásně naklikat a spravovat

velice jednoduše. Tedy přesun z příkazového řádku do aplikace pro správu verzí byl krok dobrým směrem.



Obrázek 13 - chybová hláška při nahrávání na Github

2.7.3 Zavirovaný počítač

Ačkoliv tento problém přímo nesouvisí s vývojem aplikace, dotklo se ho to taktéž. Bohužel se mi někde povedlo stáhnout opravdu extrémní vir, který v tu chvíli dostal přístup do celého mého počítače. Tedy mi někdo vykradl celou krypto peněženku, ve které tedy nebylo zrovna moc peněz, takže mě to tolik netrápilo. Co mě trápilo více je, že jsem měl ve stolním počítači uložených dost hesel. Kupříkladu i heslo od školního emailu, který mi byl zablokován přímo firmou Microsoft, jelikož se právě z mého emailu zkoušel útočník za krátkou chvíli odeslat přes dva tisíce emailů. Ale abych se dostal k souvislosti s vývojem aplikace. Aplikace je, jak jsem zmiňoval výše, připojena k databázovému serveru, více v kapitole **1.2.1 Databázový server**. Databázový server je můj na mém WEB hostingu, na kterém mám taktéž své vlastní webové stránky. Vzhledem k tomu, že útočník získal všechna má hesla, podařilo se mu smazat celou databázi a veškerá nastavení u mého hostingu. Musel jsem tedy celou strukturu databáze vytvářet znova. Naštěstí jsem měl uložený aktuální ER Diagram databáze, takže její vytvoření mi nezabralo moc dlouho. Dále jsem musel i přímo v programu upravit hesla a všechny další tajné informace pro připojování k emailům nebo právě zmíněné databázi.

2.7.4 Problém s responzivitou

Největším problémem byla responzivita. Jednalo se o poslední věc, kterou jsem na celé aplikaci dělal. Na svém počítači jsem aplikaci spustil a fungovala naprosto v

pořádku a bez jakýchkoliv známek chyb. Spustil jsem tu samou verzi u sebe na notebooku a některé texty byly ustřiženy a některé byly zase posunuty. Tento problém jsem řešil zhruba měsíc, než jsem přišel na to, že nejen samotný Formulář má nastavení pro automatické nastavení velikosti, ale i dané texty jej mají taktéž. Nastavil jsem tedy na všechny texty, kterých se to týkalo, toto nastavení na zakázáno – tedy, aby se neměnila jejich velikost sama. A již všechno fungovalo tak, jak mělo.

2.8 Přehled verzí

V aplikaci funguje verzování tak, že uživatel s nižší verzí, než je aktuální verze, si musí program aktualizovat sám. Aplikace uživateli otevře přímo stránky, na kterých lze nejnovější verzi najít a stáhnout.

Můj verzovací systém je následující. První číslo značí velký posun v aplikaci, větší aktualizaci nebo změnu ve struktuře databáze. Druhé číslo jsou klasické malé aktualizace a třetí je pouze pro vývojářské účely.

2.8.1 Základní pojmy verzí

Máme tři základní pojmy při zveřejňování nových verzí programů:

- "Feature" – novinka, nápad;
- "Bug Fix" – ošetření chyby;
- "Hot Bug Fix" – ošetření důležité / extrémní chyby. Většinou dva tři dny od vydání poslední verze.

2.8.2 Verze 1.1.0

Jedná se o první verzi aplikace, ve které byla spousta chyb a postrádala mnoho funkcí, které jsem přidával postupem času.

2.8.3 Verze 1.2.0

- Feature – žádný
- Bug Fix – žádný
- Hot Bug Fix
    - Opravení vypuštění tajné informace o přihlašovacích datech

2.8.4 Verze 1.3.0

- Feature – žádný
- Bug Fix
    - Nastavení maximální délky telefonního čísla na devět znaků
    - Při pokusu o načítání QR kódu, nemá-li počítač kameru, se spustí vlastní chybová hláška
- Hot Bug Fix – žádný

2.8.5 Verze 1.4.0

- Feature

    - Možnost obnovení hesla pomocí kódu z e-mailu
- Bug Fix – žádný
- Hot Bug Fix – žádný

2.8.6 Verze 1.5.0

- Feature
    - Možnost výběru vstupní kamery pro načítání QR kódů
    - Přidání spouštěcích parametrů
    - Změna zašifrování ze SHA1 na SHA256
- Bug Fix – žádný
- Hot Bug Fix – žádný

2.8.7 Verze 2.1.0

- Feature
    - Přidání dalších spouštěcích parametrů
    - Přidání logovacího systému
    - Přidání nahlašovacího systému
    - Přidání data a času do hlavní sekce
    - Napojení aplikace na Discord
- Bug Fix
    - Zakázání uživatelům bez práv náhled do sekcí, do kterých nemají práva
    - Vymazání vývojářské zprávy po přihlášení
- Hot Bug Fix – žádný

2.8.8 Verze 2.2.0

- Features
    - Pokus o přihlášení po zmáčknutí klávesy enter
    - Při najetí kurzorem na texty, které někam odkazují, změna myši na typ kliknutí
    - Přidání cache systému pro Formuláře
    - Posílání logovacího souboru při nahlášení
- Bug Fix
    - Změna funkce odesílání e-mailu na funkci asynchronní
    - Opravení zobrazování vlajek států
    - Přidání zprávy při pokusu o zadání špatných dat do registrace uživatele
    - Oprava špatně psaného slova “uncorrect” na “incorrect”
    - Opravení nefunkčního nahlašovacího systému
    - Přidání zprávy uživatelům bez práv snažícím se kliknout na jiné sekce
    - Otevření pouze jednoho nahlašovacího okna
- Hot Bug Fix

    - Kliknutí na stejnou sekci neobnoví celou sekci
    - Opravení nefunkčnosti cache systému pro automaticky přihlášené uživatele

2.8.9 Finální verze 3.1.0

- Features
    - Přidání načítacího okna
    - Přidání sekce přihlášených uživatelů do hlavní sekce
    - Přidaní celé aplikace do cache systému
    - Přidání seznamu dostupných států, které třetí strana podporuje
- Bug Fix
    - Opravení chyby zobrazující graf o jeden měsíc posunutý dopředu
- Hot Bug Fix - žádný

3 Závěr

V celé aplikaci jsem udělal spoustu funkcí, které jsou v rámci vyšších studií, a to většinou na vysokých školách. Při vývoji aplikace jsem se naučil opravdu mnoho nových algoritmů a způsobů programování. Bohužel aplikace nebude mít žádné aktivní využití v provozu v žádné nemocnici nebo kdekoliv jinde. A tak tedy zůstává pouze na mém Github profilu jako projekt do mého portfolia a životopisu. Taktéž z projektu budu čerpat funkce, které můžu jen nakopírovat a zaimplementovat do jiných projektů. Celý průběh vývoje, až na pár nedostatků a extrémních problémů s vývojem, si cením a vážím. Myslím si, že se mi aplikace povedla jak po vzhledové stránce, tak i po té funkční. Dále si myslím, že jsem splnil téma aplikace vzhledem k tomu, že jsem musel využít znalosti snad ze všech předmětů vyučovaných na škole.

4 Využité technologie

- CitacePro [online]. [cit. 2022-01-25]. Dostupné z: https://www.citacepro.com
- Visual Studio 2017 [software]. [cit. 2022-01-25]. Verze 16.11.4. Dostupné z: https://visualstudio.microsoft.com/vs/
- Visual Studio 2022 [software]. [cit. 2022-01-25]. Verze 17.0.5. Dostupné z: https://visualstudio.microsoft.com/vs/
- Microsoft Word 2007 [software]. [cit. 2022-01-25]. Dostupné z: CD-ROM
- PhpMyAdmin [online]. [cit. 2022-01-25]. Dostupné z: https://www.phpmyadmin.net
- Typora [software]. [cit. 2022-01-25]. Verze 0.11.17 (beta). Dostupné z: https://typora.io
- Visual Studio Code [software]. [cit. 2022-01-25]. Verze 1.63.2. Dostupné z: https://code.visualstudio.com
- Covid-19 data [online]. [cit. 2022-01-25]. Dostupné z: https://rapidapi.com/Gramzivi/api/covid-19-data/
- Flaticon [online]. [cit. 2022-01-25]. Dostupné z: https://www.flaticon.com
- Github [online]. [cit. 2022-01-25]. Dostupné z: https://github.com
- Github Desktop [software]. [cit. 2022-01-25]. Verze 2.9.6. Dostupné z: https://desktop.github.com
- StackOverFlow [online]. [cit. 2022-01-25]. Dostupné z: https://stackoverflow.com
- W3Schools [online]. [cit. 2022-01-25]. Dostupné z: https://www.w3schools.com
- Mockaroo [online]. [cit. 2022-01-25]. Dostupné z: https://www.mockaroo.com
- QR Code Generator [online]. [cit. 2022-01-25]. Dostupné z: https://goqr.me/api/