PRAXE

Datum: 22.02. 2024	Střední průmyslová škola, Chomutov, Školní 50, příspěvková organizace	Třída: V4
Číslo úlohy: 4.	Přístupový systém s RFID	Jméno: Tomáš Bartoš

Zadání:

S využitím periferie čteček RFID čipů připojené pomocí USB / sériového rozhraní sestavte aplikaci, která bude na základě informací uložených v SQLite databázi vyhodnocovat oprávněnost požadavku na přístup z daného místa.

Výsledné řešení musí obsahovat následující funkcionalitu:

- Obsluha dvou čteček připojených na periferii pomocí RS485 rozhraní
- Identifikace čtečky a způsobu načtení informace (načtení čipu, manuální zadání kódu na klávesnici). Simulujte tři místa události (klávesnice u vstupu, čip v místě A, čip v místě B)
- Identifikace osoby a vyhodnocení oprávněnosti přístupu na základě místa události, aktuálního času a dne v týdnu
- Uživatelsky použitelné rozhraní pro nastavení údajů do databáze (vlastník, členství ve skupinách, nastavení
- přístupových oprávnění a pravidel pro přístup)
- Zobrazení výsledku vyhodnocení žádosti a logování všech událostí do tabulky databáze ve formátu umožňujícím efektivní prohledávání na úrovni SQL dotazu (zamyslete se jak by mohlo např. vypadat dohledání vstupů do konkrétní kanceláře)

Teorie:

Řešení této úlohy je postaveno na dvou hlavních pilířích. Prvním z pilířů je samotný software úlohy, který má z hlediska komplexity podíl na celkovém řešení nejvyšší. Druhým pilířem je samotný hardware, který se v tomto konkrétním případě sestává ze dvou čteček a RFID čipů pro identifikaci jednotlivých uživatelů.

Samotné čtečky pracují na principu elektromagnetické indukce, přičemž čtečka vysílá do prostoru elektromagentické vlny, které při dostatečné vzdálenosti od RFID tagů aktivují interní RFID čipy karet, čímž jim poskytnou dostatek energie k odeslání identifikačních dat čtečce.

Z hlediska dělení existují:

- Aktivní čtečky
  - Mají vlastní zdroj energie a vysílají neustále svůj signál
  - Příklad = klíčky od auta
- Pasivní čtečky
  - Nemají vlastní zdroj energie a k odeslání dat nutná blízkost ke čtečce
  - Příklad = typické RFID tagy

Protože čtečky samotné nejsou schopny odesílat data ke zpracování bezdrátově, v úloze jsou propojeny jednotlivé komponenty řešení (deska se čtečkami a PC) pomocí rozhraní RS485. RS485 je sériové průmyslové rozhraní používané pro diferenciální přenos dat, který je prováděn half-duplexně. Co se technických parametrů týče je zde podpora komunikace až na vzdálenost 1200 metrů a rychlostmi dosahujícími maximálně 10Mb/s. Na rozhraní se zařízení připojují jako do síťové topologie Bus, tudíž je nutné terminovat oba konce páteřní linky.

Po softwarové stránce je zde nejdůležitější technologií technologie SQLite. Jedná se o light-weight relační databázový systém, který je často preferován pro svou jednoduchost, rychlost a snadnou integraci do různých projektů. Tato open-source databáze vyniká svou schopností běžet bez nutnosti samostatného databázového serveru, což usnadňuje vývojářům práci a umožňuje snadné vložení do aplikací.

Pro práci s SQLite technologií v C# je potřeba nainstalovat knihovnu Systém.Data.SQLite buď skrz packet manager NuGet, či přímo ze stránek https://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki.

Základní propojení vrstvy programu a databáze se pak provádí následovně:

1. Propojení se souborem databáze:
```
SQLiteConnection connection = new SQLiteConnection("Data
Source=database.db;Version=3;");
connection.Open();
```
2. Tvorba příkazu:
```
SQLiteCommand command = connection.CreateCommand();
command.CommandText = "CREATE TABLE car (id INTEGER PRIMARY KEY, manufacturer TEXT,
year INTEGER)";
```
3. Provedení příkazu:
```
command.ExecuteNonQuery();
```

Popis programu:

Formulář kontroly:

Ihned po spuštění programu se ve fázi inicializace inicializují globální objekty pro práci s databází (DatabaseController), pro reprezentaci akcí a dat čteček (ReaderController) a pro logování událostí (Log Controller). Spolu s tímto počátečním vytvořením instancí globálních objektů jsou také inicializovány 3 timery:

- pollTimer = slouží pro kontrolu stavu sériové linky
- accessTimer = slouží pro schování panelu access/granted po autentifikacin
- clockTimer = slouží k updatu aktuálního času

V neposlední řadě je inicializace dokončena inicializací samotného sériového rozhraní a získáním času.

Následuje hlavní „smyčka" programu, která je spíše než smyčka pouze sled eventů a jejich výsledné handlování. Nejdůležitějším z eventů je checkStatus, event handler, který se vykonává vždy při dovření časového limitu pollTimeru a pomocí porovnání aktuálního počtu přijatých dat (readCounter) a naposledy přečtené hodnoty tohoto počtu (lastReadStatus) determinuje, zda byl načten nový tag či zadán nový kód. V případě, že ano, je na základě adresy zadaný/načtený kód převeden do originální podoby (kód byl zadán na num-padu) či ponechán v podobě původní (byl načten RFID tag). Jak kód, tak adresa jsou převedeny na odpovídající globální objekty (kód na objekt User -> metoda AssignFromCode, adresa na Reader -> metoda AssignFromAddress a uživatel na Group -> metoda AssignGroup) a je provedena autentifikace, zobrazení jejího výsledku a uložení logu do databáze.

Způsob, jakým je uživateli ověřen přístup do dané sekce budovy je založen na jednotlivých permissions, které jsou přiřazeny skupinám uživatelů. Databázová struktura počítá s tím, že jakýkoliv uživatel bude součástí skupiny a daná skupina může mít díky relační tabulce groupPermission více oprávnění. Jakmile je tedy načten tag uživatele a program se dostane do fáze autentikace, iteruje se skrz veškeré permissions přiřazené jedné skupině a v závislost na aktuální čas, den a adrese čtečky je uživateli přístup povolen či zakázán.

Po načtení tagu/kódu a zobrazení stavu oprávnění accessTimer svým přerušením přesně po 5 sekundách od zobrazení informací o uživateli data schová. Důležité je podotknout, že pro schování dat uživatelů je nutno použít delegáta, jelikož daný timer běží v jiném vlákně.

Formulář administrace:

V moment spuštění se spustí inicializační proces stejný jako u formuláře kontroly, ovšem je v něm vynechána fáze inicializace sériového rozhraní (SerialController) a z veškeré dropdown listy a list-boxy jsou naplněny daty v databázi (metody populateGroups, PopulatePermissionsList, PopulateReaders).

Co se běhu samotného programu týče jsou uživateli nabízeny interakce se 3 sekcemi:

- sekce úpravy uživatelů
- sekce úpravy skupin
- sekce úpravy oprávnění

Ač je každá ze sekcí jiná co se vzhedlu týče, principem co se back-endu týče se liší jen minimálně. Každá z těchto sekcí totiž nabízí CRUD operace na UI levelu aplikace.

Co se akcí spojených s editem uživatele týče, tak ty vždy začínají vyplněním datových polí, kdy jsou vyplněna všechna pole i s kódem a administrátor tak vytváří nového uživatele, či jsou vyplněna pouze pole pro jméno a příjmení a uživatel si nechává podrobnější data vyhledávat v databázi a následně volí, zda by rád uživatele smazal (metoda user_delete_button_Click) či nechal propsat data z polí do databáze (metoda user_save_button_Click).

Samotné metody pro úpravy jsou vždy založeny na stejném principu:

Získání ID uživatele

Formátování UPDATE či DELETE query tak, aby obsahovala příslušné ID

Execute SQLite příkazu

Kontrola chyb

Notifikace uživatele o stavu provedené operace

Úprava skupin pracuje na stejném principu, ovšem s tím rozdílem, že vyhledávání dat není podmíněno kliknutém na tlačítko „vyhledat“ ale pouze výběrem příslušné skupiny. Data, která jsou dohledávána jsou pak jednotlivá oprávnění.

Pro identifikaci skupiny je zde využito samotného názvu, ze kterého se následně generuje objekt GroupController pro snazší práci.

Stejně jako úprava skupin funguje i úprava oprávnění. Načítaná data jsou jednotlivé denní a hodinové rozsahy společně s daty o čtečce.

Rozbor Proměnných a metod:

Metody:

Form1.cs

Typ	Název metody	Argumenty	Popis
void	Form1_Load	object sender, EventArgs e	Metoda sloužící k inicializaci formuláře.
void	clockTimerElapsed	object sender, ElapsedEventArgs e	Obsluha přerušení časovače pro update aktuálního času.
void	checkStatus	object sender, ElapsedEventArgs e	Obsluha přerušení časovače pro získání dat ze čtečky.
void	admin_btn_Click	object sender, EventArgs e	Metoda vyvolaná kliknutím na tlačítko pro přístup do admin panelu.
void	adminViewClosed	object sender, FormClosedEventArgs e	Metoda vyvolaná po zavření administračního panelu.
void	showData	int addr	Metoda pro zobrazení dat o uživateli v GUI.
void	ShowLogs	žádné	Metoda pro zobrazení posledních událostí v logu v GUI.
void	accessTimerElapsed	object sender, ElapsedEventArgs e	Obsluha přerušení časovače pro smazání polí v GUI.
void	clearPanel0	žádné	Metoda pro vyčištění panelu čtečky 0.
void	clearPanel1	žádné	Metoda pro vyčištění panelu čtečky 1.
void	clearPanel2	žádné	Metoda pro vyčištění panelu čtečky 2.

AdminView.cs

Typ	Název metody	Argumenty	Popis
void	AdminView_Load	object sender, EventArgs e	Inicializuje komponenty a naplňuje prvky GUI daty.
void	pollTimerElapsed	object sender, ElapsedEventArgs e	Kontroluje a zobrazuje data uživatele na základě zadaného kódu.
void	ShowUserData	bool dataFromReader = false	Zobrazuje data uživatele v GUI.
void	PopulateGroups	ComboBox box	Naplňuje ComboBox s seznamem skupin.
void	PopulatePermissionsList	žádné	Naplňuje ListBox s seznamem oprávnění.
void	PopulatePeermissionsCombo	žádné	Naplňuje ListBox s oprávněními pro skupiny.
void	PopulateReaders	žádné	Naplňuje ComboBox s názvy čteček.
void	user_find_button_Click	object sender, EventArgs e	Hledá a zobrazuje data uživatele podle jména a příjmení.
void	NotifyUser	string message	Zobrazuje zprávy pro uživatele.
void	ResetGlobalObjects	žádné	Resetuje globální objekty uživatelů a skupin.
void	ResetData	žádné	Resetuje data v GUI.
void	user_save_button_Click	object sender, EventArgs e	Ukládá nebo aktualizuje informace o uživateli.
void	user_delete_button_Click	object sender, EventArgs e	Smaže záznam o uživateli.

DatabaseController.cs

Typ	Název	Argumenty	Popis
Void	InitDataba	Žádné	Inicializuje databázi a vytváří s ní spojení.
Void	CreateTab	string tableName, string tableDefinition	Vytváří tabulku v databázi s daným názvem a definicí.
Void	CreateTab	Žádné	Vytváří potřebné tabulky v databázi.
Void	InsertLog	int userID, int readerID, DateTime accessTime, bool accessResult	Vkládá záznam o přístupu do tabulky logů.
Void	InsertNew	int startDay, int endDay, string startTime, string endTime, int readerID	Vkládá nové oprávnění (permission) do databáze.
Void	DeletePer	int id	Maže oprávnění z databáze na základě zadaného ID.
Void	UpdatePe	int id, int startDay, int endDay, string startTime, string endTime, int readerID	Aktualizuje existující oprávnění v databázi.
SQLiteDataReader	GetGroup	int groupID	Získává skupinu na základě ID.
Int	Entries	string table	Vrací počet záznamů v dané tabulce.
Void	InsertUser	string name, string surname, string code, int groupID	Vkládá nového uživatele do databáze.
Void	UpdateUs	int id, string name, string surname, string code, int groupID	Aktualizuje existujícího uživatele v databázi.
Void	DeleteUse	int id	Maže uživatele z databáze na základě zadaného ID.

GroupController.cs

Typ	Název	Argumenty	Popis
Void	GroupController	DatabaseController databaseController	Konstruktor třídy, inicializuje instanci třídy GroupController.
Void	AssignModel	SQLiteDataReader rdr	Přiřazuje model skupiny na základě dat vrácených z čtečky SQLiteDataReader.
Void	AssignFromId	int id	Přiřazuje model skupiny na základě zadaného ID.
Void	AssignFromName	string name	Přiřazuje model skupiny na základě zadaného jména.
Void	GetAllPermissions	Žádné	Získává všechna oprávnění a ukládá je do seznamu permissions.
List<string>	GetGroups	Žádné	Získává názvy všech skupin a vrací je jako seznam.
Int	CreateNewGroup	Žádné	Vytváří novou skupinu v databázi.
Void	UpdateGroup	Žádné	Aktualizuje existující skupinu v databázi.
Void	DeleteGroup	Žádné	Maže existující skupinu z databáze.

LogController.cs

Typ	Název	Argumenty	Popis
Void	LogController	DatabaseController databaseController	Konstruktor třídy, inicializuje instanci třídy LogController.
LogModel	GetLogModel	SQLiteDataReader rdr	Získává model záznamu o přístupu na základě dat vrácených z čtečky SQLiteDataReader.
List<LogModel>	GetLatestLogs	int count = 5	Získává nejnovější záznamy o přístupech.
Void	LogEntry	int userID, int readerID, bool accessResult	Zaznamenává vstup do databáze.

PermissionController.cs

Návratový typ	Název	Argumenty	Popis
Void	PermissionController	DatabaseController databaseController	Konstruktor třídy, inicializuje instanci třídy PermissionController.
Void	AssignModel	SQLiteDataReader rdr	Přiřazuje model oprávnění na základě dat vrácených z čtečky SQLiteDataReader.
String	GetString	Žádné	Vrací řetězec představující informace o oprávnění.
Void	AssignFromId	int id	Přiřazuje model na základě zadaného ID.
Bool	AccessGranted	int day, DateTime time, int reader	Kontroluje, zda je určité oprávnění uděleno v daný čas, den a na místě.
List<PermissionController>	GetAllPermissions	Žádné	Získává všechna oprávnění z databáze.
String	GetReaderName	Žádné	Získává název čtečky z databáze.
Void	CreateNewPermission	int startDay, int endDay, string startTime, string endTime, int readerID	Vytváří nové oprávnění.
Void	DeletePermission	int id	Maže oprávnění.
PermissionController	GetPermissionByString	string permissionString	Získává oprávnění na základě zadaného řetězce.
Void	UpdatePermission	int id, int startDay, int endDay, string startTime, string endTime, int readerID	Aktualizuje oprávnění.

ReaderController.cs

Typ	Název	Argumenty	Popis
Void	ReaderController	DatabaseController databaseController	Konstruktor třídy, inicializuje instanci třídy ReaderController.
Void	AssignModel	SQLiteDataReader reader	Přiřazuje model čtečky na základě dat vrácených z čtečky SQLiteDataReader.
Void	AssignFromAddress	int address	Přiřazuje model na základě adresy čtečky.
Void	AssignFromName	string name	Přiřazuje model na základě jména čtečky.
List<string>	GetReaderNames	Žádné	Získává jména všech čteček z databáze.

SerialController.cs

Typ	Název	Argumenty	Popis
Void	SerialController	string port, int baudRate	Konstruktor třídy, inicializuje instanci třídy SerialController.
Void	Init	Žádné	Inicializuje spojení tím, že přiřadí událostní handler a otevře port.
Void	dataReceived	object sender, SerialDataReceivedEventArgs e	Metoda pro přidání přijatých dat do bufferu.
Int	convertToDec	string code	Metoda pro převod hexadecimálního čísla na desítkové.
Void	closeConnection	Žádné	Metoda pro uzavření sériového spojení.
Void	openConnection	Žádné	Metoda pro otevření sériového spojení.

UserController.cs

Typ	Název	Argumenty	Popis
Void	UserController	DatabaseController databaseController	Konstruktor třídy, inicializuje instanci třídy UserController.
Void	AssignGroup	Žádné	Metoda, která přiřazuje skupinu uživateli.
String	GetGroupName	Žádné	Metoda pro získání názvu skupiny.
Bool	HasAccess	int readerID	Metoda, která zjišťuje, zda má uživatel přístup k danému čtečce.
Void	AssignModel	SQLiteDataReader reader	Metoda, která přiřazuje model z čteče.
Void	AssignFromCode	string code	Metoda, která přiřazuje model z kódu přístupu.
Void	AssignFromName	string name, string surname	Metoda, která přiřazuje model z jména a příjmení.
Void	CreateUser	Žádné	Metoda pro vytvoření nového uživatele.
Void	UpdateUser	string code, int groupID	Metoda pro aktualizaci uživatele.
Void	DeleteUser	Žádné	Metoda pro smazání uživatele.

Proměnné:

Form1.cs		
Typ	**Název**	**Popis**
Timer	pollTimer	Timer pro periodické ověřování stavusériové linky.
Timer	accessTimer	Timer pro schování panelu "granted"/"denied".
Timer	clockTimer	Timer pro zobrazení aktuálního času.
DatabaseController	dbController	Instance třídy pro ovládání databáze.
SerialController	serialController	Instance třídy pro ovládání sériového portu.
ReaderController	readerController	Instance třídy pro ovládání čteček.
UserController	userController	Instance třídy pro ovládání uživatelů.
LogController	logController	Instance třídy pro ovládání logů.
AdminView	adminView	Instance třídy pro zobrazení administrátorského rozhraní.
String	code	Kód aktuálního uživatele.
Int	lastReadStatus	Stav posledního čtení.

AdminView.cs		
Typ	**Název**	**Popis**
Timer	pollTimer	Timer pro periodické ověřování stavu linky.
DatabaseController	databaseController	Instance třídy pro ovládání databáze.
UserController	userController	Instance třídy pro ovládání uživatelů.
GroupController	groupController	Instance třídy pro ovládání skupin.
SerialController	serialController	Instance třídy pro ovládání sériového portu.
ReaderController	readerController	Instance třídy pro ovládání čteček.
PermissionController	permissionController	Instance třídy pro ovládání oprávnění.
List<PermissionController>	currentPermissions	Seznam aktuálních oprávnění.
String	code	Kód aktuálního uživatele.
Int	lastReadStatus	Stav posledního čtení.

GroupController.cs		
Typ	**Název**	**Popis**
GroupModel	groupObj	Instance třídy reprezentující skupinu.
List<PermissionController>	permissions	Seznam oprávnění spojených se skupinou.
DatabaseController	dbController	Instance třídy pro ovládání databáze.

LogController.cs		
Typ	**Název**	**Popis**
DatabaseController	dbController	Instance třídy pro ovládání databáze.
LogModel	logModel	Instance třídy reprezentující záznam v logu.
List<LogModel>	logs	Seznam instancí třídy LogModel, reprezentujících nejnovější záznamy v logu.

ReaderController.cs		
Typ	**Název**	**Popis**
DatabaseController	dbController	Instance třídy pro ovládání databáze.
ReaderModel	readerObj	Instance třídy reprezentující čtečku karet.
SQLiteDataReader	rdr	Datareader pro čtení dat z databáze.
List<string>	readerNames	Seznam názvů čteček karet v databázi.

SerialController.cs		
Typ	**Název**	**Popis**
SerialPort	serialPort	Instance třídy pro komunikaci přes sériový port.
String	buffer	Buffer pro ukládání příchozích dat.
Dictionary<int, int>	codes	Slovník pro uchovávání kódů.
Boolean	dataReady	Určuje, zda jsou data připravena k zpracování.
String	currentCode	Aktuální kód získaný ze sériového portu.
Int	currentAddr	Aktuální adresa získaná ze sériového portu.
Int	readCounter	Počítadlo počtu provedených čtení ze sériového portu.
String	port	Název portu pro sériovou komunikaci.
Int	baudRate	Rychlost baudů pro sériovou komunikaci.
SerialDataReceivedEventHandler	dataReceived	Událost označující, že byla přijata data ze sériového portu.
String	data	Přijatá data ze sériového portu.
String	pattern	Regulární výraz pro vyhledání určitých vzorů ve vstupních datech.
MatchCollection	matches	Kolekce shod nalezených pomocí regulárního výrazu.
Match	match	Shoda nalezená pomocí regulárního výrazu.
String	g0, g1, g2, g3, g4	Skupiny zachycené z shody pomocí regulárního výrazu.
Int	reader_addr	Adresa čtečky, která byla použita k přečtení kódu.
Int	startIndex	Index, na kterém začíná shoda nalezená pomocí regulárního výrazu v bufferu.
String	newBuffer	Nový buffer po odebrání přečteného kódu.
System.Globalization.NumberStyles	HexNumber	Určuje, jaký typ čísla má být načten z řetězce.

UserController.cs		
Typ	**Název**	**Popis**
UserModel	userObj	Instance třídy UserModel reprezentující uživatele.
DatabaseController	dbController	Instance třídy pro ovládání databáze.
GroupController	groupController	Instance třídy pro ovládání skupin.

GroupModel.cs		
Typ	**Název**	**Popis**
Int	ID	Identifikátor skupiny.
String	name	Název skupiny.

GroupPermissionModel.cs		
Typ	**Název**	**Popis**
Int	ID	Identifikátor oprávnění skupiny.
Int	groupID	Identifikátor skupiny.
Int	permissionID	Identifikátor oprávnění.

LogModel.cs		
Typ	**Název**	**Popis**
Int	ID	Identifikátor logu.
Int	userID	Identifikátor uživatele.
Int	readerID	Identifikátor čtečky.
DateTime	accessTime	Čas přístupu.
String	accessResult	Výsledek přístupu (např. "Povolený"/"Zakázaný").

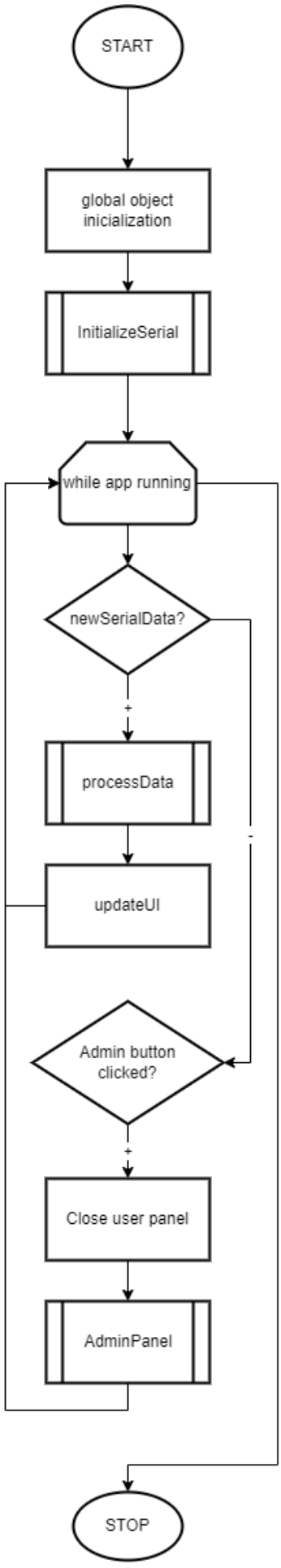
PermissionModel.cs		
Typ	**Název**	**Popis**
Int	ID	Identifikátor oprávnění.
Int	startDay	Počáteční den platnosti oprávnění.
Int	endDay	Koncový den platnosti oprávnění.
String	startTime	Čas začátku platnosti oprávnění.
String	endTime	Čas konce platnosti oprávnění.
Int	readerID	Identifikátor čtečky, ke které se oprávnění vztahuje.

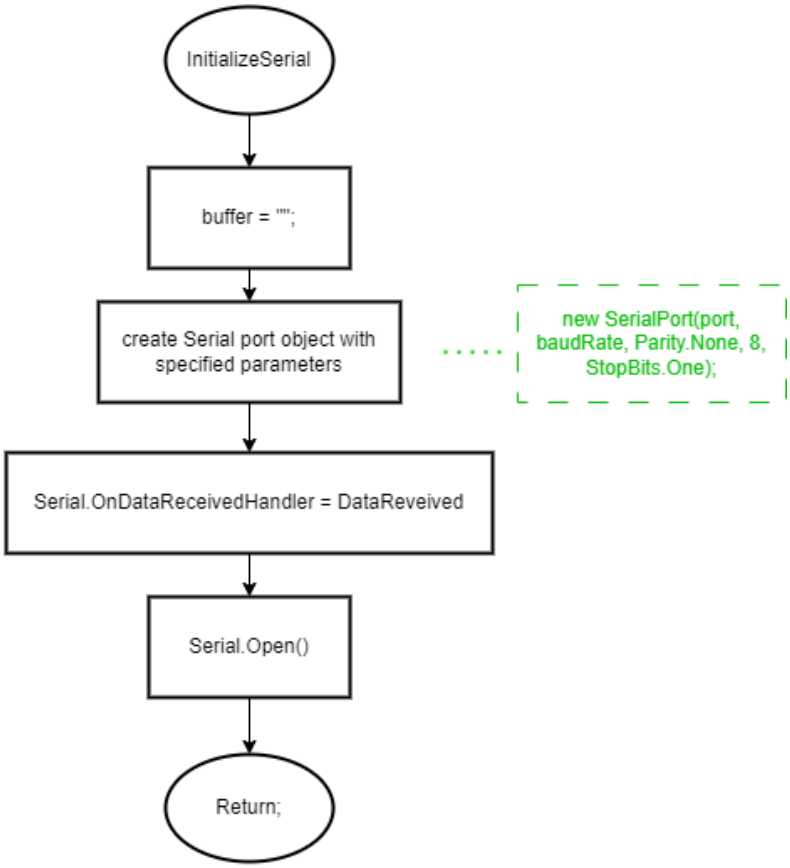
ReaderModel.cs		
Typ	**Název**	**Popis**
Int	ID	Identifikátor čtečky.
String	name	Název čtečky.
String	uuid	UUID reprezentující schéma dekódování pro regulární výrazy.

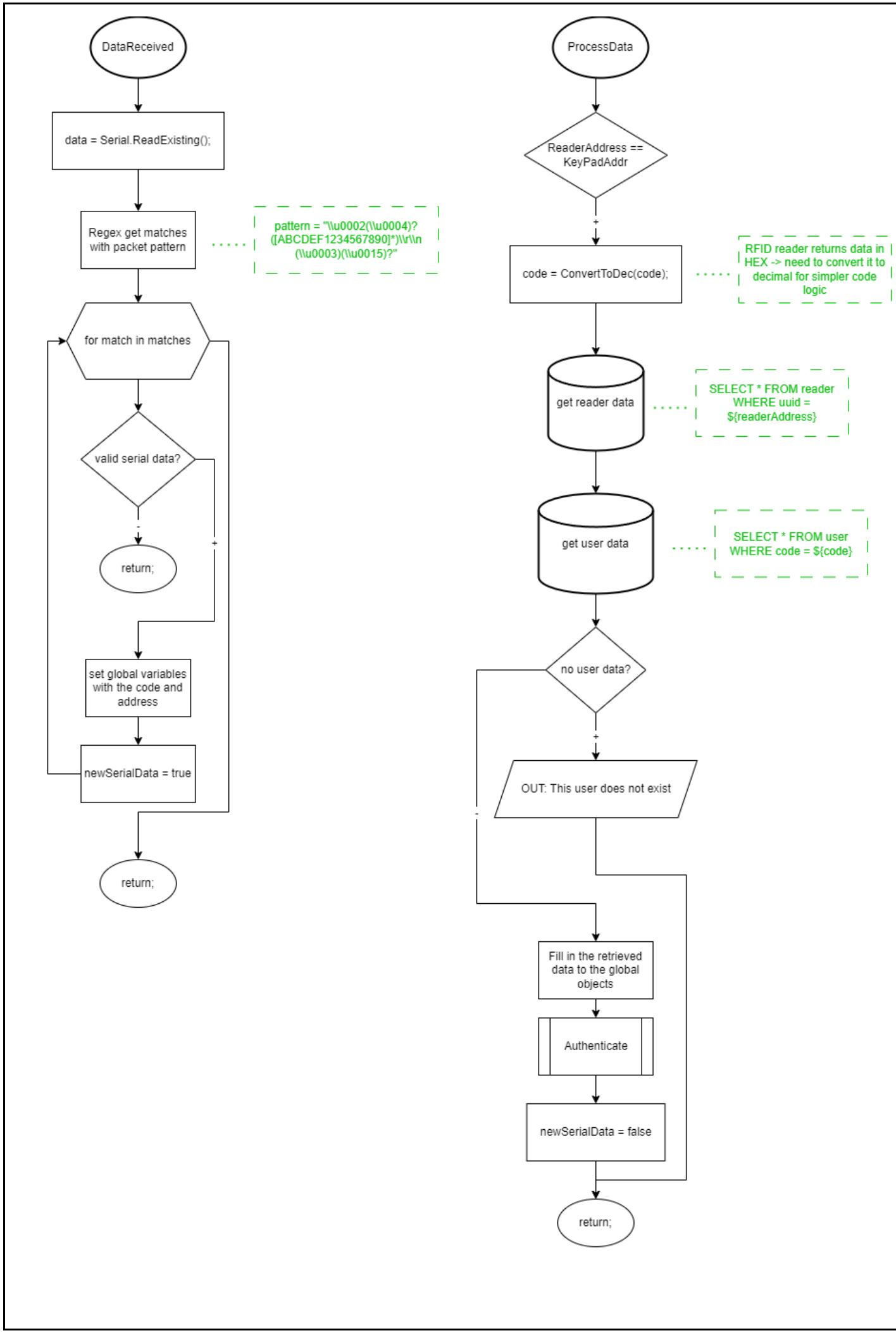
UserModel.cs		
Typ	**Název**	**Popis**
Int	ID	Identifikátor uživatele.
String	name	Jméno uživatele.
String	surname	Příjmení uživatele.
String	code	Kód přístupu uživatele.
Int	groupID	Identifikátor skupiny, ke které uživatel patří.

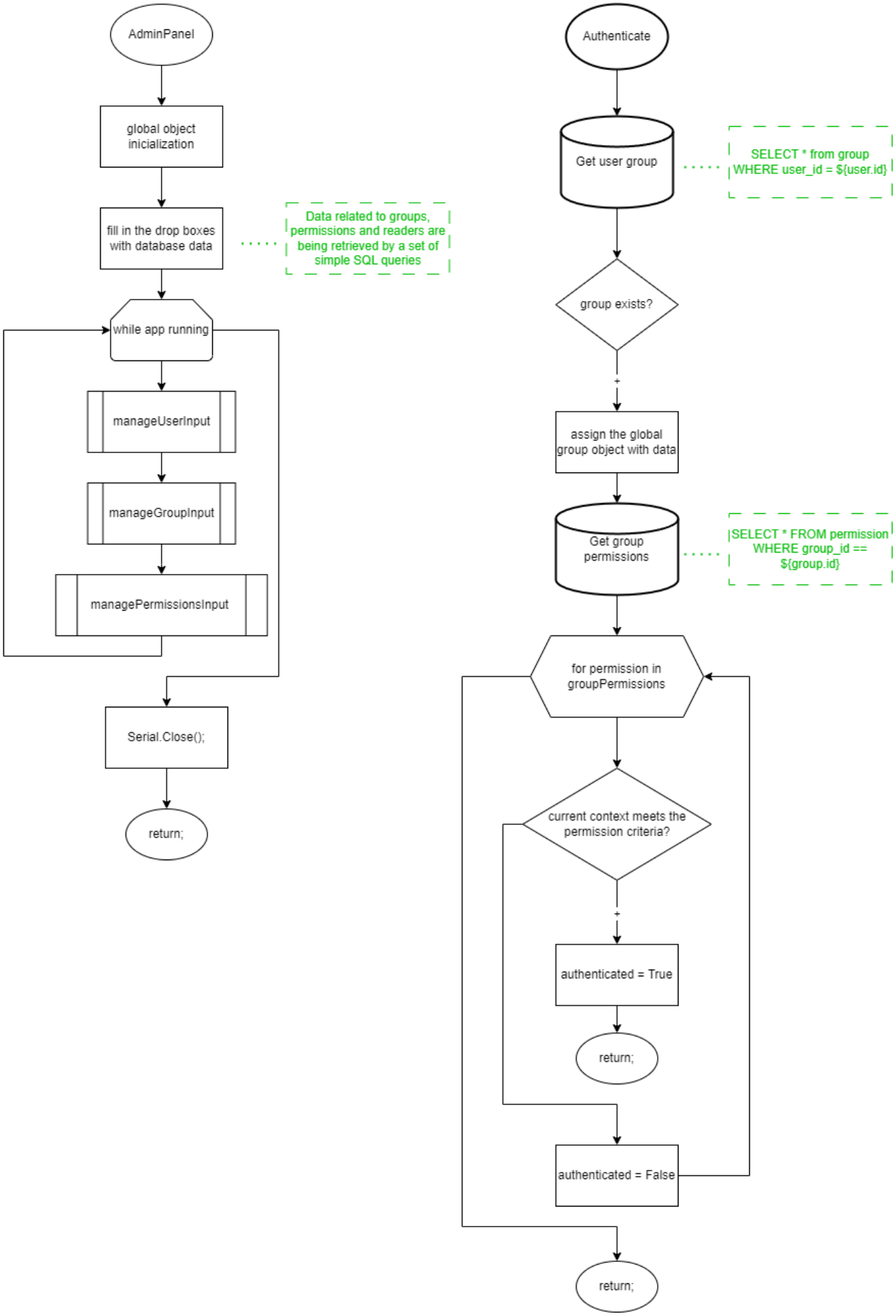
DatabaseController.cs		
Typ	**Název**	**Popis**
string	path	Cesta k databázovému souboru
bool	fillData	Indikuje, zda je databáze plná dat
SQLiteConnection	connection	SQLite spojení
SQLiteCommand	command	SQLite příkaz

Vývojový diagram:






DataReceived
data = Serial.ReadExisting();
Regex get matches with packet pattern
pattern = "\\u0002(\\u0004)?([ABCDEF1234567890]*)\\r\\n(\\u0003)(\\u0015)?"
for match in matches
valid serial data?
-
return;
+
set global variables with the code and address
newSerialData = true
return;
ProcessData
ReaderAddress == KeyPadAddr
+
code = ConvertToDec(code);
RFID reader returns data in HEX -> need to convert it to decimal for simpler code logic
get reader data
SELECT * FROM reader WHERE uuid = ${readerAddress}
get user data
SELECT * FROM user WHERE code = ${code}
no user data?
+
OUT: This user does not exist
-
Fill in the retrieved data to the global objects
Authenticate
newSerialData = false
return;


AdminPanel
global object inicialization
fill in the drop boxes with database data
Data related to groups, permissions and readers are being retrieved by a set of simple SQL queries
while app running
manageUserInput
manageGroupInput
managePermissionsInput
Serial.Close();
return;
Authenticate
Get user group
SELECT * from group WHERE user_id = ${user.id}
group exists?
assign the global group object with data
Get group permissions
SELECT * FROM permission WHERE group_id == ${group.id}
for permission in groupPermissions
current context meets the permission criteria?
authenticated = True
return;
authenticated = False
return;

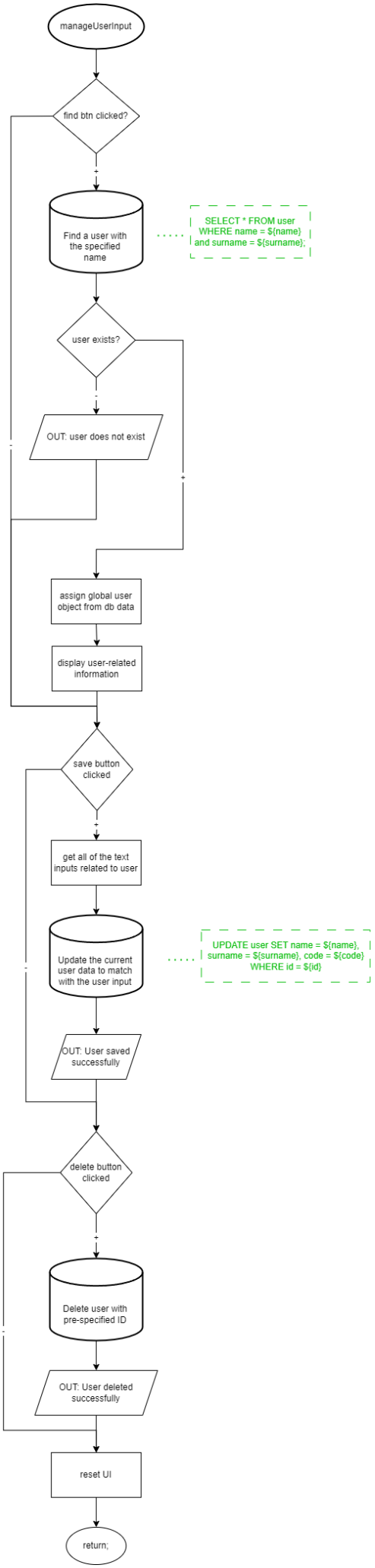
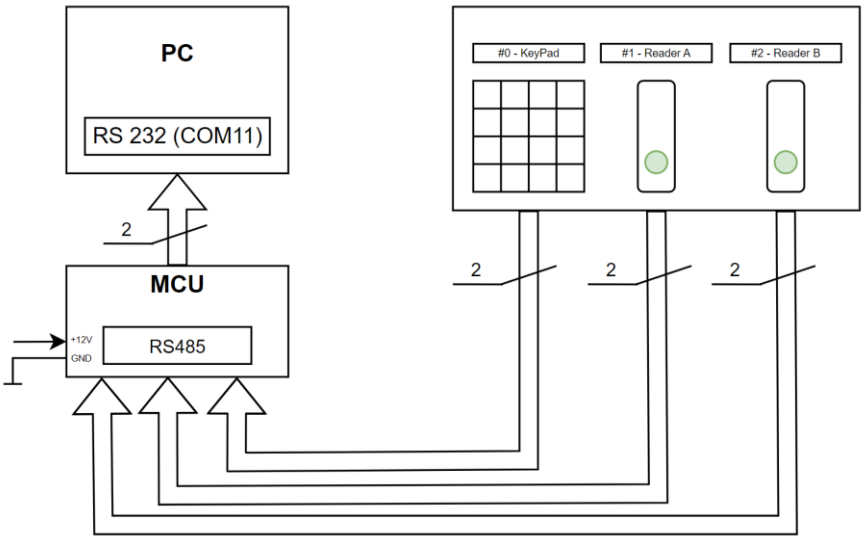
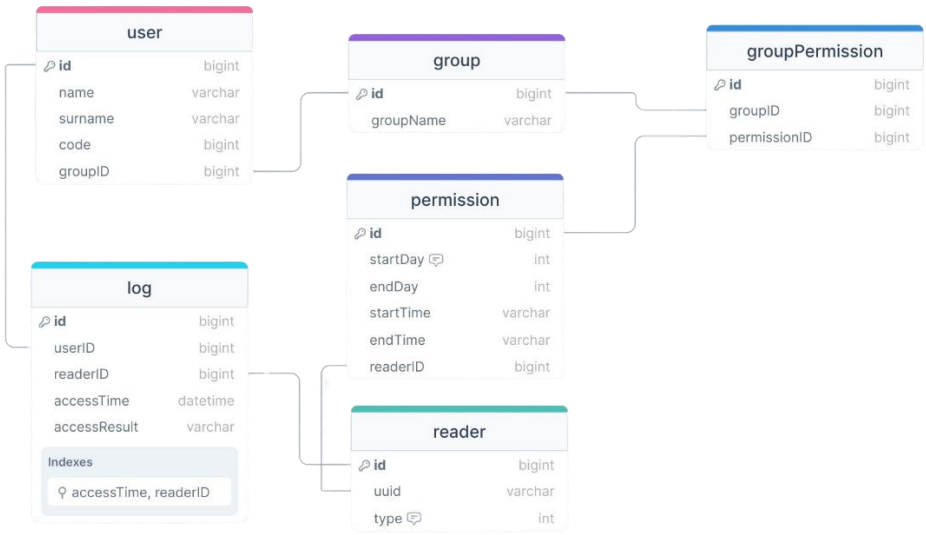

manageUserInput
find btn clicked?
Find a user with the specified name
SELECT * FROM user WHERE name = ${name} and surname = ${surname};
user exists?
OUT: user does not exist
assign global user object from db data
display user-related information
save button clicked
get all of the text inputs related to user
Update the current user data to match with the user input
UPDATE user SET name = ${name}, surname = ${surname}, code = ${code} WHERE id = ${id}
OUT: User saved successfully
delete button clicked
Delete user with pre-specified ID
OUT: User deleted successfully
reset UI
return;

Schéma zapojení:

a. Schéma zapojení:



b. Databázové schéma:



Komentovaný výpis programu:

Form1.cs:

```
using Entra.Controller;
using Entra.Model;
using Entra.View;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Timers;
using System.Windows.Forms;
```

```
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;


namespace Entra
{
    public partial class Form1 : Form
    {
        public System.Timers.Timer pollTimer { get; set; }
        public System.Timers.Timer accessTimer { get; set; }
        public System.Timers.Timer clockTimer { get; set; }
        public DatabaseController dbController { get; set; }
        public SerialController serialController { get; set; }
        public ReaderController readerController { get; set; }
        public UserController userController { get; set; }
        public LogController logController { get; set; }
        public AdminView adminView { get; set; }

        public string code { get; set; }

        internal int lastReadStatus { get; set; }
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // global objects
            dbController = new DatabaseController();
            readerController = new ReaderController(dbController);
            logController = new LogController(dbController);

            // poll timer settings
            pollTimer = new System.Timers.Timer();
            pollTimer.Interval = 200;
            pollTimer.Elapsed += checkStatus;
            pollTimer.Start();

            // access timer settings
            accessTimer = new System.Timers.Timer();
            accessTimer.Interval = 5000;
            accessTimer.Elapsed += accessTimerElapsed;

            // clock timer setup
            clockTimer = new System.Timers.Timer();
            clockTimer.Interval = 1000;
            clockTimer.Elapsed += clockTimerElapsed;
            clockTimer.AutoReset = true;
            clockTimer.Start();

            serialController = new SerialController("COM13", 19200);
            serialController.Init();
```

```
            lastReadStatus = 0;

            // setup the GUI
            clearPanel0();
            clearPanel1();
            clearPanel2();
            time_label.Text = DateTime.Now.ToString("dddd, HH:mm:ss");

        }

        private void clockTimerElapsed(object sender, ElapsedEventArgs e)
        {
            // method that executes every second -> get current time and display it
            DateTime time = DateTime.Now;
            time_label.Invoke((MethodInvoker)delegate
            {
                // get the current name of the day as well as the clock
                time_label.Text = DateTime.Now.ToString("dddd, HH:mm:ss");
            });
        }

        private void checkStatus(object sender, ElapsedEventArgs e)
        {
            // function for checking the status of the serial controller so that if new
data apper
            // on the serial bus the GUI responds appropriately
            // should be as simple as checking the model class that the serial controller
will return
            if (lastReadStatus != serialController.readCounter)
            {
                int reader_address = serialController.currentAddr;
                if(reader_address == 0)
                {
                    code =
serialController.convertToDec(serialController.currentCode).ToString();
                }
                else{
                    code = serialController.currentCode;
                }

                // assign the database model from address so further details could be
extracted
                readerController.AssignFromAddress(reader_address);

                if (readerController.readerObj.uuid != "")
                {
                    // the rederdata got parsed successfully
                    // now I can get the user data from the database
                    userController = new UserController(dbController);
                    userController.AssignFromCode(code);

                    if(userController.userObj != null)
                    {
                        // the user data got parsed successfully
```

```
                        // now I can show the data in the panel

                        // assign the group data to the user

                        userController.AssignGroup();

                        if(userController.groupController.groupObj == null)
                        {
                            return;
                        }

                        // show the final data
                        showData(reader_address);

                        // log the entry
                        logController.LogEntry(userController.userObj.ID, reader_address,
userController.HasAccess(reader_address));

                        // show logs
                        ShowLogs();
                    }
                    else
                    {
                        // user does not exist in the databse -> pass
                        //MessageBox.Show("User for this code does not exist");
                    }
                }
                lastReadStatus = serialController.readCounter;
            }

        }

        private void admin_btn_Click(object sender, EventArgs e)
        {
            // method for redirecting the user to the admin page
            // stop the internal timers
            pollTimer.Stop();
            accessTimer.Stop();
            clockTimer.Stop();

            // close the serial connection so that it can be used by the admin panel

            // open the admin panel
            serialController.readCounter = 0;
            adminView = new AdminView();
            adminView.FormClosed += new FormClosedEventHandler(adminViewClosed);
            adminView.Show();
        }

        private void adminViewClosed(object sender, FormClosedEventArgs e)
        {
            // method for reseting the form status after the admin panel is closed

            // reopen the serial connection
```

```
            serialController.openConnection();
            // restart timers
            pollTimer.Start();
            accessTimer.Start();
            clockTimer.Start();
        }

        private void showData(int addr)
        {

            if (addr == 0)
            {
                reader_panel_0.Invoke((MethodInvoker)delegate
                {
                    user_input_0.Text = $"{userController.userObj.name}
{userController.userObj.surname}";
                    code_input_0.Text = code.ToString();
                    group_input_0.Text = userController.GetGroupName();

                    bool status = userController.HasAccess(addr);

                    if (status)
                    {
                        granted_panel_0.Visible = true;
                        denied_panel_0.Visible = false;
                    }
                    else
                    {
                        granted_panel_0.Visible = false;
                        denied_panel_0.Visible = true;
                    }
                });

            }
            else if (addr == 1)
            {
                reader_panel_1.Invoke((MethodInvoker)delegate
                {
                    user_input_1.Text = $"{userController.userObj.name}
{userController.userObj.surname}";
                    code_input_1.Text = code.ToString();
                    group_input_1.Text = userController.GetGroupName();
                    bool status = userController.HasAccess(addr);

                    if (status)
                    {
                        granted_panel_1.Visible = true;
                        denied_panel_1.Visible = false;
                    }
                    else
                    {
                        granted_panel_1.Visible = false;
                        denied_panel_1.Visible = true;
                    }
```

```
                });
            }
            else if (addr == 2)
            {
                reader_panel_1.Invoke((MethodInvoker)delegate
                {
                    user_input_2.Text = $"{userController.userObj.name}
{userController.userObj.surname}";
                    code_input_2.Text = code.ToString();
                    group_input_2.Text = userController.GetGroupName();
                    bool status = userController.HasAccess(addr);

                    if (status)
                    {
                        granted_panel_2.Visible = true;
                        denied_panel_2.Visible = false;
                    }
                    else
                    {
                        granted_panel_2.Visible = false;
                        denied_panel_2.Visible = true;
                    }
                });
            }
            accessTimer.Start();
        }

        private void ShowLogs()
        {
            // method for showing the logs from the db

            // get the list of logs -> by default only first five
            List<LogModel> logs = logController.GetLatestLogs();

            if(logs == null)
            {
                return;
            }

            logs_text.Clear();
            foreach (LogModel log in logs)
            {
                logs_text.Text += $"{log.accessTime} - {log.userID} - {log.readerID} -
{log.accessResult}\n";
            }
        }

        private void accessTimerElapsed(object sender, ElapsedEventArgs e)
        {
            // method for clearing the data after user access

            reader_panel_0.Invoke((MethodInvoker)delegate
            {
                clearPanel0();
```

```
            });
            reader_panel_1.Invoke((MethodInvoker)delegate
            {
                clearPanel1();
            });
            reader_panel_2.Invoke((MethodInvoker)delegate
            {
                clearPanel2();
            });
        }

        private void clearPanel0()
        {
            user_input_0.Text = "";
            code_input_0.Text = "";
            group_input_0.Text = "";
            granted_panel_0.Visible = false;
            denied_panel_0.Visible = true;
        }

        private void clearPanel1()
        {
            user_input_1.Text = "";
            code_input_1.Text = "";
            group_input_1.Text = "";
            granted_panel_1.Visible = false;
            denied_panel_1.Visible = true;
        }
        private void clearPanel2()
        {
            user_input_2.Text = "";
            code_input_2.Text = "";
            group_input_2.Text = "";
            granted_panel_2.Visible = false;
            denied_panel_2.Visible = true;
        }
    }
}
```

AdminView.cs:

```
using Entra.Controller;
using Entra.Model;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Timers;
using System.Windows.Forms;
```

```
namespace Entra.View
{
    public partial class AdminView : Form
    {
        public System.Timers.Timer pollTimer { get; set; }
        public DatabaseController databaseController { get; set; }
        public UserController userController { get; set; }
        public GroupController groupController { get; set; }
        public SerialController serialController { get; set; }
        public ReaderController readerController { get; set; }
        public PermissionController permissionController { get; set; }
        public List<PermissionController> currentPermissions { get; set; }


        public string code { get; set; }
        internal int lastReadStatus { get; set; }

        public AdminView()
        {
            InitializeComponent();
        }

        private void AdminView_Load(object sender, EventArgs e)
        {
            // assign the global components
            databaseController = new DatabaseController();
            userController = new UserController(databaseController);
            groupController = new GroupController(databaseController);
            readerController = new ReaderController(databaseController);
            permissionController = new PermissionController(databaseController);
            currentPermissions = new List<PermissionController>();

            // initialize the serial controller
            serialController = new SerialController("COM13", 19200);
            serialController.Init();

            // global variabled
            lastReadStatus = 0;
            code = "";

            //setup the poll timer
            pollTimer = new System.Timers.Timer();
            pollTimer.Interval = 200;
            pollTimer.Elapsed += pollTimerElapsed;
            pollTimer.Start();

            // set the listbox to be able to select multiple items
            group_permissions.SelectionMode = SelectionMode.MultiExtended;

            // pupulate table with data
            PopulateGroups(user_group_combo);
            PopulateGroups(group_group_combo);
            PopulatePermissionsList();
```

```
            PopulatePeermissionsCombo();
            PopulateReaders();

        }

        private void pollTimerElapsed(object sender, ElapsedEventArgs e)
        {
            // method for checking if any code has been entered
            // the sole puprose of this method is to auto-fill the code input to search
user

            if (lastReadStatus != serialController.readCounter)
            {
                // get the scanned code
                code = serialController.currentCode;

                userController.AssignFromCode(code);

                // check if the model was occupied with data
                if(userController.userObj != null)
                {
                    // user exists -> populate the current user group
                    groupController.AssignFromId(userController.userObj.groupID);
                    // call function to show the found user
                    ShowUserData(true);
                }
                // if the model has data -> show it
                // else display error message
                lastReadStatus = serialController.readCounter;
            }
        }

        private void ShowUserData(bool dataFromReader = false)
        {
            name_input.Text = userController.userObj.name;
            surname_input.Text = userController.userObj.surname;

            if(dataFromReader)
            {
                code_input.Text = code.ToString();
            }
            else
            {
                code_input.Text = userController.userObj.code.ToString();
            }

            // populate the group comboboxes
            PopulateGroups(user_group_combo);
            PopulateGroups(group_group_combo);

            Console.Error.WriteLine("User group name = " + groupController.groupObj.name);


            // find out the index of the specific user group
```

```
            int selectedIndex =
user_group_combo.FindString(groupController.groupObj.name);

            // select the group item
            user_group_combo.SelectedIndex = selectedIndex;

        }

        private void PopulateGroups(ComboBox box)
        {
            // method for populating the group combobox
            // get the list of groups
            currentPermissions.Clear();

            List<string> groups = groupController.GetGroups();

            box.Items.Clear();

            foreach(string group in groups)
            {
                box.Items.Add(group);
            }

        }

        private void PopulatePermissionsList()
        {
            // method for populating the permissions list

            // get the list of permissions
            currentPermissions.Clear();
            List<PermissionController> permissions =
permissionController.GetAllPermissions();

            if(permissions == null)
            {
                // we have no permissions -> return
                NotifyUser("No permissions in db");
                return;
            }

            // cler the listbox
            permission_combo.Items.Clear();

            foreach(PermissionController permission in permissions)
            {
                permission_combo.Items.Add(permission.GetString());
                currentPermissions.Add(permission);
            }
        }

        private void PopulatePeermissionsCombo()
        {
            // method for populating the permissions combobox
```

```
            // get the list of permissions
            List<PermissionController> permissions =
permissionController.GetAllPermissions();

            if (permissions == null)
            {
                // we have no permissions -> return
                NotifyUser("No permissions in db");
                return;
            }

            // cler the listbox
            group_permissions.Items.Clear();

            foreach (PermissionController permission in permissions)
            {
                group_permissions.Items.Add(permission.GetString());
                currentPermissions.Add(permission);
            }
        }

        public void PopulateReaders()
        {
            // method that fills in the data of a textbox with the reader names
            reader_combo.Items.Clear();

            List<string> readers = readerController.GetReaderNames();

            foreach(string reader in readers)
            {
                reader_combo.Items.Add(reader);
            }
        }

        private void user_find_button_Click(object sender, EventArgs e)
        {
            // reset the data
            ResetGlobalObjects();

            // user clicked the find button
            // check if required fields are set
            if(name_input.Text == "" || surname_input.Text == "")
            {
                NotifyUser("Please fill in the name and surname before searching.");
                return;
            }

            // try to assign the usercontroller with data
            userController.AssignFromName(name_input.Text, surname_input.Text);

            // check if data assigned -> user exists
            if(userController.userObj == null)
            {
```

```
                NotifyUser("User not found.");
                ResetData();
                return;
            }

            // user exists -> populate the current user group
            groupController.AssignFromId(userController.userObj.groupID);

            if(groupController.groupObj == null)
            {
                NotifyUser("User has no assigned groups;");
            }

            // call function to show the found user
            ShowUserData(false);
        }

        private void NotifyUser(string message)
        {
            // message for displaying user related messages
            MessageBox.Show(message);
        }

        private void ResetGlobalObjects()
        {
            // method for resetting the data
            userController.userObj = null;
            groupController.groupObj = null;
        }

        private void ResetData()
        {
            // method for resetting the data
            name_input.Text = "";
            surname_input.Text = "";
            code_input.Text = "";
            user_group_combo.SelectedIndex = -1;
            group_group_combo.SelectedIndex = -1;
        }

        private void user_save_button_Click(object sender, EventArgs e)
        {
            if (name_input.Text == "" || surname_input.Text == "" || code_input.Text == ""
|| user_group_combo.SelectedIndex == -1)
            {
                NotifyUser("You must fill in all the fields before saving.");
                return;
            }

            // check if the user exists
            userController.AssignFromName(name_input.Text, surname_input.Text);

            if (userController.userObj == null)
            {
```

```
            // user with such credentials does not exist -> create new user
            // lookup the group for this user

            groupController.AssignFromName(user_group_combo.SelectedItem.ToString());

            if (groupController.groupObj == null)
            {
                NotifyUser("Invalid user group.");
                return;
            }

            // assign the user controller with the new user data
            userController.userObj = new UserModel
            {
                name = name_input.Text,
                surname = surname_input.Text,
                code = code_input.Text,
                groupID = groupController.groupObj.ID
            };

            userController.CreateUser();

            NotifyUser("User successfully created.");

            ResetGlobalObjects();
            ResetData();
        }

        // user exists -> model has been assigned -> update the user
        try
        {
            groupController.AssignFromName(user_group_combo.SelectedItem.ToString());
        }
        catch
        {

        }

        if (groupController.groupObj == null)
        {
            NotifyUser("Invalid user group.");
            return;
        }

        userController.UpdateUser(code_input.Text, groupController.groupObj.ID);

        NotifyUser("User data successfull updated.");

        ResetGlobalObjects();
        ResetData();
    }

    private void user_delete_button_Click(object sender, EventArgs e)
    {
```

```
            if (name_input.Text == "" || surname_input.Text == "" || code_input.Text == ""
|| user_group_combo.SelectedIndex == -1)
            {
                NotifyUser("You must fill in all the fields before saving.");
                return;
            }

            // check if the user exists
            userController.AssignFromName(name_input.Text, surname_input.Text);

            if (userController.userObj == null)
            {
                NotifyUser("User not found.");
                return;
            }

            groupController.AssignFromName(user_group_combo.SelectedItem.ToString());
            if (groupController.groupObj == null)
            {
                NotifyUser("Invalid user group.");
                return;
            }

            userController.DeleteUser();

            ResetGlobalObjects();
            ResetData();

            NotifyUser("User deleted successfully.");

        }

        private void group_group_combo_SelectedIndexChanged(object sender, EventArgs e)
        {
            if(group_group_combo.SelectedIndex == -1)
            {
                // return on no selection -> user will provide input
                return;
            }
            // reset data
            groupController.permissions.Clear();
            currentPermissions.Clear();
            groupController.groupObj = null;

            // on each select get permissions for that group
            PopulatePermissionsList();

            DeselectPermissionItems();

            // get the group of the selected index
            groupController.AssignFromName(group_group_combo.SelectedItem.ToString());

            if (groupController.groupObj == null)
            {
```

```
                NotifyUser("Invalid user group.");
                return;
            }

            // highlight the permissions for the selected group
            groupController.GetAllPermissions();

            foreach(PermissionController permission in groupController.permissions)
            {
                // check the listbox for match in the string content and highlight the
matches
                for(int i = 0; i < group_permissions.Items.Count; i++)
                {
                    if (group_permissions.Items[i].ToString() == permission.GetString())
                    {
                        group_permissions.SetSelected(i, true);
                    }
                }
            }
        }

        private void DeselectPermissionItems()
        {
            // method for deselecting all items in the listbox
            for(int i = 0; i < group_permissions.Items.Count; i++)
            {
                group_permissions.SetSelected(i, false);
            }
        }

        private void group_save_button_Click(object sender, EventArgs e)
        {
            if (group_group_combo.SelectedIndex == -1)
            {
                if(group_group_combo.Text == "")
                {
                    NotifyUser("You must select a group or enter a new group name before
saving.");
                    return;
                }
                // create new group object and save it with the currently selected
permissions
                groupController.groupObj = new GroupModel
                {
                    name = group_group_combo.Text
                };

                foreach(var item in group_permissions.SelectedItems)
                {
                    // get the selected permissions and assign them to the group
                    foreach(PermissionController permission in currentPermissions)
                    {
                        if(permission.GetString() == item.ToString())
                        {
```

```
                        groupController.permissions.Add(permission);
                    }
                }
            }

            int status = groupController.CreateNewGroup();

            if(status == 0)
            {
                PopulateGroups(user_group_combo);
                PopulateGroups(group_group_combo);
                NotifyUser("Group successfully created.");
            }
            else
            {
                NotifyUser("Error while creating group.");
            }

            return;
        }

        // user selected an index -> just update the new permissions
        Console.Error.WriteLine("Selected group is being updated ...");

        // get the group contreoller from selected
        groupController.AssignFromName(group_group_combo.SelectedItem.ToString());

        if(groupController.groupObj == null)
        {
            NotifyUser("Invalid user group.");
            return;
        }

        // clear the permissions
        groupController.permissions.Clear();

        // set new permissions
        foreach (var item in group_permissions.SelectedItems)
        {
            // get the selected permissions and assign them to the group
            foreach (PermissionController permission in currentPermissions)
            {
                if (permission.GetString() == item.ToString())
                {
                    groupController.permissions.Add(permission);
                }
            }
        }

        // update the group
        groupController.UpdateGroup();
        ResetGroupData();
        NotifyUser("Group succesfully updated.");
    }
```

```
        private void ResetGroupData()
        {
            group_group_combo.SelectedIndex = -1;
            group_permissions.ClearSelected();

            groupController.permissions.Clear();
            groupController.groupObj = null;
        }

        private void group_delete_button_Click(object sender, EventArgs e)
        {
            // method for deleting a specified group
            if (group_group_combo.SelectedIndex == -1)
            {
                NotifyUser("You must select a group before deleting.");
                return;
            }

            // get the group contreoller from selected
            groupController.AssignFromName(group_group_combo.SelectedItem.ToString());

            if(groupController.groupObj == null)
            {
                NotifyUser("Invalid user group.");
                return;
            }

            groupController.DeleteGroup();
            ResetGroupData();
            PopulateGroups(user_group_combo);
            PopulateGroups(group_group_combo);
            NotifyUser("Group succesfully deleted.");
        }

        private void group_permissions_MouseDown(object sender, MouseEventArgs e)
        {
            // Check if the right mouse button is clicked
            if (e.Button == MouseButtons.Right)
            {
                // Get the index of the item at the mouse pointer's location
                int index = group_permissions.IndexFromPoint(e.Location);

                // If an item is found at the mouse pointer's location
                if (index != ListBox.NoMatches)
                {
                    // Unselect the item
                    group_permissions.SetSelected(index, false);
                }
            }
        }

        private void permission_combo_SelectedIndexChanged(object sender, EventArgs e)
        {
```

```
            // method for updating the groupbox input data to match permission data
            if(permission_combo.SelectedIndex == -1)
            {
                return;
            }

            // get the selected permission
            PermissionController permission =
permissionController.GetPermissionByString(permission_combo.SelectedItem.ToString());
            if (permission == null)
            {
                NotifyUser("Invalid permission");
                return;
            }

            // populate the input fields
            start_day_numeric.Value = permission.permissionObj.startDay;
            end_day_numeric.Value = permission.permissionObj.endDay;
            start_time_input.Text = permission.permissionObj.startTime;
            end_time_input.Text = permission.permissionObj.endTime;

            // get the if of a reder from the combobobox
            string reader_name =  permission.GetReaderName();

            // get the item index to select from the combobox
            int selectedIndex = reader_combo.FindString(reader_name);

            // set the selected index
            reader_combo.SelectedIndex = selectedIndex;
        }

        private void permissions_save_button_Click(object sender, EventArgs e)
        {
            // check the permission combobox for index -> if -1 -> adding new permission,
otherwise updating

            if (start_day_numeric.Value > 6 || end_day_numeric.Value > 6 ||
start_time_input.Text == "" || end_time_input.Text == "" || reader_combo.SelectedIndex ==
-1)
            {
                NotifyUser("You must fill in all the fields before saving.");
                return;
            }

            if (!DateTime.TryParse(start_time_input.Text, out DateTime start) ||
!DateTime.TryParse(end_time_input.Text, out DateTime end))
            {
                NotifyUser("Invalid time format.");
                return;
            }

            if (DateTime.Parse(start_time_input.Text) >
DateTime.Parse(end_time_input.Text))
            {
```

```
                NotifyUser("Start time must be smaller than end time.");
                return;
            }

            if (start_day_numeric.Value > end_day_numeric.Value)
            {
                NotifyUser("Start day must be smaller than end day.");
                return;
            }

            if (permission_combo.SelectedIndex == -1)
            {
                // create new permission

                // get the reader id from the combobox
                readerController.AssignFromName(reader_combo.SelectedItem.ToString());

                if(readerController.readerObj == null)
                {
                    NotifyUser("Invalid reader.");
                    return;
                }

                // create new permission
                permissionController.CreateNewPermission((int)start_day_numeric.Value,
(int)end_day_numeric.Value, start_time_input.Text, end_time_input.Text,
readerController.readerObj.ID);

                ResetPermissionData();

                NotifyUser("Permission successfully created.");
                PopulatePermissionsList();
                PopulatePeermissionsCombo();
                return;
            }

            // update the permission
            PermissionController permission =
permissionController.GetPermissionByString(permission_combo.SelectedItem.ToString());
            if (permission.permissionObj == null)
            {
                NotifyUser("Invalid permission");
                return;
            }

            // get the reader id from the combobox
            readerController.AssignFromName(reader_combo.SelectedItem.ToString());

            if (readerController.readerObj == null)
            {
                NotifyUser("Invalid reader.");
                return;
            }
```

```
            permissionController.UpdatePermission(permission.permissionObj.ID,
(int)start_day_numeric.Value, (int)end_day_numeric.Value, start_time_input.Text,
end_time_input.Text, readerController.readerObj.ID);
            ResetPermissionData();
            PopulatePermissionsList();
            PopulatePeermissionsCombo();
            NotifyUser("Permission successfully updated.");
        }

        public void ResetPermissionData()
        {
            permission_combo.Text = "";
            permission_combo.SelectedIndex = -1;
            start_day_numeric.Value = 0;
            end_day_numeric.Value = 0;
            start_time_input.Text = "";
            end_time_input.Text = "";
            reader_combo.SelectedIndex = -1;

            readerController.readerObj = null;
        }

        private void permissions_delete_button_Click(object sender, EventArgs e)
        {
            // method for deleting permissions from the databse
            if (permission_combo.SelectedIndex == -1)
            {
                NotifyUser("You must select a permission before deleting.");
                return;
            }

            // get the selected permission
            PermissionController permission =
permissionController.GetPermissionByString(permission_combo.SelectedItem.ToString());
            if (permission.permissionObj == null)
            {
                NotifyUser("Invalid permission");
                return;
            }
            permissionController.DeletePermission(permission.permissionObj.ID);
            ResetPermissionData();
            PopulatePermissionsList();
            PopulatePeermissionsCombo();
            NotifyUser("Permission successfully deleted.");
        }
    }
}
```

DatabaseController.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Data.SQLite;
using System.Windows.Forms;
using System.IO;
using System.Data.Common;
using Entra.Model;

namespace Entra.Controller
{
    public class DatabaseController
    {
        public string path { get; set; }
        public bool fillData { get; set; }
        private SQLiteConnection connection;
        private SQLiteCommand command;

        public DatabaseController(string userPath = "entra.db")
        {
            fillData = false;
            path = userPath;
            InitDatabase();
            CreateTables();

            if (fillData)
            {
                // fill the database with some data
                return;
            }
        }

        private void InitDatabase()
        {
            if (!File.Exists(path))
            {
                try
                {
                    SQLiteConnection.CreateFile(path); // create a db file to connect to
                    fillData = true;
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"An error occured: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

            try
            {
                connection = new SQLiteConnection($"Data Source={path};Version=3;");
                connection.Open();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Nastala chyba: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
            }

            command = new SQLiteCommand(connection);
        }

        private void CreateTable(string tableName, string tableDefinition)
        {
            try
            {
                command.CommandText = $@"
                CREATE TABLE IF NOT EXISTS `{tableName}` (
                    {tableDefinition}
                )";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error creating table {tableName}: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void CreateTables()
        {
            CreateTable("groupPermission", @"
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                groupID INTEGER NOT NULL,
                permissionID INTEGER NOT NULL,
                FOREIGN KEY(groupID) REFERENCES 'group'(id),
                FOREIGN KEY(permissionID) REFERENCES permission(id)
            ");

            CreateTable("log", @"
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                userID INTEGER NOT NULL,
                readerID INTEGER NOT NULL,
                accessTime TEXT NOT NULL,
                accessResult TEXT NOT NULL,
                FOREIGN KEY(userID) REFERENCES user(id),
                FOREIGN KEY(readerID) REFERENCES reader(id)
            ");

            CreateTable("group", @"
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL
            ");
```

```
            CreateTable("user", @"
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                surname TEXT,
                code TEXT NOT NULL UNIQUE,
                groupID INTEGER NOT NULL,
                FOREIGN KEY(groupID) REFERENCES 'group'(id)
            ");

            CreateTable("reader", @"
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                uuid INTEGER NOT NULL,
                FOREIGN KEY(uuid) REFERENCES permission(readerID)
            ");

            CreateTable("permission", @"
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                startDay INTEGER NOT NULL,
                endDay INTEGER NOT NULL,
                startTime TEXT NOT NULL,
                endTime TEXT NOT NULL,
                readerID INTEGER NOT NULL,
                FOREIGN KEY(readerID) REFERENCES reader(id)
            ");

            // Index creation query for the 'log' table
            try
            {
                command.CommandText = @"
                    CREATE INDEX IF NOT EXISTS log_accesstime_readerid_index
                    ON log(accessTime, readerID)";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error creating index on 'log' table:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void InsertLog(int userID, int readerID, DateTime accessTime, bool
accessResult)
        {
            try
            {
                command = new SQLiteCommand(connection);
```

```
                command.CommandText = $"INSERT INTO 'log' (userID, readerID, accessTime,
accessResult) VALUES ({userID}, {readerID}, '{accessTime.ToString("yyyy-MM-dd
HH:mm:ss")}', '{(accessResult ? "granted" : "denied")}');";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error inserting log: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void InsertNewPermission(int startDay, int endDay, string startTime, string
endTime, int readerID)
        {
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"INSERT INTO 'permission' (startDay, endDay,
startTime, endTime, readerID) VALUES ({startDay}, {endDay}, '{startTime}', '{endTime}',
{readerID});";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error inserting permission: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void DeletePermission(int id)
        {
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"DELETE FROM 'permission' WHERE id = {id};";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
```

```
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error deleting permission: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void UpdatePermission(int id, int startDay, int endDay, string startTime,
string endTime, int readerID)
        {
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"UPDATE 'permission' SET startDay = {startDay},
endDay = {endDay}, startTime = '{startTime}', endTime = '{endTime}', readerID = {readerID}
WHERE id = {id};";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error updating permission: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public SQLiteDataReader GetGroup(int groupID)
        {
            Console.Error.WriteLine("Getting group ...");
            try
            {
                using (command = new SQLiteCommand(connection))
                {
                    command.CommandText = $"SELECT * FROM [group] WHERE id = @id";
                    command.Parameters.AddWithValue("@id", groupID);

                    SQLiteDataReader result = command.ExecuteReader();
                    if (result != null)
                    {
                        if (result.Read())
                        {
                            Console.Error.WriteLine("DEBUGGED = " +
Convert.ToString(result["name"]));
```

```
                        return result;
                    }
                }
            }
        }
        catch (SQLiteException ex)
        {
            MessageBox.Show($"SQLite 30 error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"An 30 error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        return null;
    }

    public List<int> GetGroupPermissionIDs(int groupID)
    {
        List<int> permissionIDs = new List<int>();

        try
        {
            using (command = new SQLiteCommand(connection))
            {
                command.CommandText = $"SELECT permissionID FROM groupPermission WHERE
groupID = {groupID}";

                using (SQLiteDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        permissionIDs.Add(Convert.ToInt32(reader["permissionID"]));
                    }
                }
            }
        }
        catch (SQLiteException ex)
        {
            MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        return permissionIDs;
    }

    public List<string> GetReaderNames()
    {
```

```
            List<string> readerNames = new List<string>();

            try
            {
                using (command = new SQLiteCommand(connection))
                {
                    command.CommandText = "SELECT name FROM reader";

                    using (SQLiteDataReader reader = command.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            readerNames.Add(reader["name"].ToString());
                        }
                    }
                }

                return readerNames;
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle general exceptions
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            return readerNames; // Return an empty list in case of error
        }


        public int CreateNewGroup(string name, List<PermissionController> permissions)
        {
            try
            {
                using (SQLiteCommand command = new SQLiteCommand(connection))
                {
                    command.CommandText = $"INSERT INTO [group] (name) VALUES (@name);";
                    command.Parameters.AddWithValue("@name", name);
                    command.ExecuteNonQuery();

                    // Get the latest group ID from the group table
                    int groupID = GetGroupId(name);

                    if (groupID == -1)
                    {
                        // Return error flag
                        return -1;
                    }
```

```
                    // Insert permissions for the group
                    foreach (PermissionController permission in permissions)
                    {
                        using (SQLiteCommand permissionCommand = new
SQLiteCommand(connection))
                        {
                            permissionCommand.CommandText = $"INSERT INTO groupPermission
(groupID, permissionID) VALUES (@groupID, @permissionID);";
                            permissionCommand.Parameters.AddWithValue("@groupID",
groupID);
                            permissionCommand.Parameters.AddWithValue("@permissionID",
permission.permissionObj.ID);
                            permissionCommand.ExecuteNonQuery();
                        }
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return -1; // Return error flag
            }
            catch (Exception ex)
            {
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return -1; // Return error flag
            }
            return 0;
        }


        public void UpdateGroup(int id, string name, List<PermissionController>
permissions)
        {
            try
            {
                using (SQLiteCommand command = new SQLiteCommand(connection))
                {
                    command.CommandText = $"UPDATE [group] SET name = @name WHERE id =
@id";
                    command.Parameters.AddWithValue("@name", name);
                    command.Parameters.AddWithValue("@id", id);
                    command.ExecuteNonQuery();

                    // Delete all permissions for the group
                    using (SQLiteCommand deleteCommand = new SQLiteCommand(connection))
                    {
                        deleteCommand.CommandText = $"DELETE FROM groupPermission WHERE
groupID = @groupID";
                        deleteCommand.Parameters.AddWithValue("@groupID", id);
                        deleteCommand.ExecuteNonQuery();
```

```
                    }

                    // Insert permissions for the group
                    foreach (PermissionController permission in permissions)
                    {
                        using (SQLiteCommand permissionCommand = new
SQLiteCommand(connection))
                        {
                            permissionCommand.CommandText = $"INSERT INTO groupPermission
(groupID, permissionID) VALUES (@groupID, @permissionID);";
                            permissionCommand.Parameters.AddWithValue("@groupID", id);
                            permissionCommand.Parameters.AddWithValue("@permissionID",
permission.permissionObj.ID);
                            permissionCommand.ExecuteNonQuery();
                        }
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void DeleteGroup(int id)
        {
            try
            {
                using (SQLiteCommand command = new SQLiteCommand(connection))
                {
                    command.CommandText = $"DELETE FROM [group] WHERE id = @id";
                    command.Parameters.AddWithValue("@id", id);
                    command.ExecuteNonQuery();

                    // Delete all permissions for the group
                    using (SQLiteCommand deleteCommand = new SQLiteCommand(connection))
                    {
                        deleteCommand.CommandText = $"DELETE FROM groupPermission WHERE
groupID = @groupID";
                        deleteCommand.Parameters.AddWithValue("@groupID", id);
                        deleteCommand.ExecuteNonQuery();
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
            }
            catch (Exception ex)
            {
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public int GetGroupId(string name)
        {
            try
            {
                using (command = new SQLiteCommand(connection))
                {
                    command.CommandText = $"SELECT id FROM [group] WHERE name = @name";
                    command.Parameters.AddWithValue("@name", name);

                    object result = command.ExecuteScalar();
                    if (result != null)
                    {
                        return Convert.ToInt32(result);
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return -1;
        }


        public List<PermissionModel> GetAllPermissions()
        {
            List<PermissionModel> permissions = new List<PermissionModel>();

            try
            {
                using (command = new SQLiteCommand(connection))
                {
                    command.CommandText = "SELECT * FROM permission";
                    using (SQLiteDataReader reader = command.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            permissions.Add(new PermissionModel
                            {
                                ID = Convert.ToInt32(reader["id"]),
```

```
                                startDay = Convert.ToInt32(reader["startDay"]),
                                endDay = Convert.ToInt32(reader["endDay"]),
                                startTime = Convert.ToString(reader["startTime"]),
                                endTime = Convert.ToString(reader["endTime"]),
                                readerID = Convert.ToInt32(reader["readerID"])
                            });
                        }
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            return permissions;
        }
        public string GetReaderName(int readerID)
        {
            // method for getting the name of the reader from the database
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"SELECT name FROM reader WHERE id = {readerID};";

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    // if has data -> return the reader
                    return reader["name"].ToString();
                }
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return "";
        }
```

```
        public SQLiteDataReader GetModelReader(string table, string column, string code)
        {
            SQLiteDataReader reader = null;

            try
            {
                using (SQLiteCommand command = new SQLiteCommand(connection))
                {
                    command.CommandText = $"SELECT * FROM [{table}] WHERE {column} =
@code";
                    command.Parameters.AddWithValue("@code", code);

                    reader = command.ExecuteReader();

                    if (reader.Read())
                    {
                        return reader;
                    }
                }
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle general exceptions
                MessageBox.Show($"General error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            return null;
        }

        public SQLiteDataReader GetReaderWithName(string name)
        {
            // get reader object from db with just the reader name

            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"SELECT * FROM reader WHERE name = '{name}';";

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    // if has data -> return the reader
                    // get reader name and print it to console
                    Console.WriteLine(reader["name"]);
                    return reader;
                }
```

```
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return null;
        }

        public SQLiteDataReader GetReaderWithAddress(int address)
        {
            // get reader object from db with just the reader name

            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"SELECT * FROM reader WHERE uuid = '{address}';";

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    // if has data -> return the reader
                    // get reader name and print it to console
                    Console.WriteLine(reader["uuid"]);
                    return reader;
                }
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"An error occurred: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return null;
        }


        public SQLiteDataReader GetUserByName(string name, string surname)
        {
```

```
            // function that select the user with name and surname form database and
returns its reader
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"SELECT * FROM 'user' WHERE name = '{name}' AND
surname = '{surname}';";

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    // if has data -> return the reader
                    return reader;
                }
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite errpr: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"General error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            return null;
        }

        public SQLiteDataReader GetUserByCode(string code)
        {
            // function that returns the reader of the user with the specific code
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"SELECT * FROM 'user' WHERE code = '{code}';";

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    // if has data -> return the reader
                    return reader;
                }
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite errpr: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
```

```
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"General error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return null;
        }
        public List<string> GetGroupNames()
        {
            List<string> groupNames = new List<string>();

            try
            {
                using (SQLiteCommand command = new SQLiteCommand(connection))
                {
                    command.CommandText = "SELECT name FROM [group]";

                    using (SQLiteDataReader reader = command.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            string groupName = reader["name"].ToString();
                            groupNames.Add(groupName);
                        }
                    }
                }
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle general exceptions
                MessageBox.Show($"General error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            return groupNames;
        }



        public int Entries(string table)
        {
            int index = 0;
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"SELECT * FROM '{table}';";
```

```
                SQLiteDataReader reader = command.ExecuteReader();
                if (reader.Read())
                {
                    // if has data -> return the reader
                    index++;
                }
                reader.Close();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"Chyba SQLite: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"Nastala chyba při čtení kódu prodejce: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return index; // no data  = null
        }

        public void InsertUser(string name, string surname, string code, int groupID)
        {
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"INSERT INTO 'user' (name, surname, code, groupID)
VALUES ('{name}', '{surname}', '{code}', {groupID});";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"General error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void UpdateUser(int id, string name, string surname, string code, int
groupID)
        {
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"UPDATE 'user' SET name = '{name}', surname =
'{surname}', code = '{code}', groupID = {groupID} WHERE id = {id};";
```

```
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"General error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void DeleteUser(int id)
        {
            // method that deletes the user with the specific id
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"DELETE FROM 'user' WHERE id = {id};";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"SQLite error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"General error: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
}
```

GroupController.cs:

```
using Entra.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SQLite;
using System.Windows.Forms;

namespace Entra.Controller
{
```

```
    public class GroupController
    {
        public GroupModel groupObj { get; set; }
        public List<PermissionController> permissions { get; set; }

        private DatabaseController dbController { get; set; }

        public GroupController(DatabaseController databaseController)
        {
            dbController = databaseController;
            permissions = new List<PermissionController>();
        }

        public void AssignModel(SQLiteDataReader rdr)
        {
            try
            {
                groupObj = new GroupModel
                {
                    ID = Convert.ToInt32(rdr["id"]),
                    name = Convert.ToString(rdr["name"]),
                };
            }
            catch (Exception ex)
            {
                Console.WriteLine($"An error occured: {ex.Message}");
            }
        }

        public void AssignFromId(int id)
        {
            SQLiteDataReader rdr = dbController.GetModelReader("group", "id",
id.ToString());
            // check if no data returned from reder
            if (rdr != null)
            {
                AssignModel(rdr);
                return;
            }

            groupObj = null;
        }

        public void AssignFromName(string name)
        {
            SQLiteDataReader rdr = dbController.GetModelReader("group", "name", name);
            // check if no data returned from reder
            if (rdr != null)
            {
                AssignModel(rdr);
                return;
            }
            groupObj = null;
        }
```

```
        public void GetAllPermissions()
        {
            // method that gets all of the permissions and stores them in the permissions
list
            permissions.Clear();

            List<int> permissionIDs = dbController.GetGroupPermissionIDs(groupObj.ID);
            foreach (int id in permissionIDs)
            {
                PermissionController permissionController = new
PermissionController(dbController);
                permissionController.AssignFromId(id);
                permissions.Add(permissionController);
            }
        }

        public List<string> GetGroups()
        {
            List<string> groups = dbController.GetGroupNames();

            return groups;
        }

        public int CreateNewGroup()
        {
            // method for creating a new group
            int status = dbController.CreateNewGroup(groupObj.name, permissions);
            return status;

        }

        public void UpdateGroup()
        {
            // method for updating a group
            dbController.UpdateGroup(groupObj.ID, groupObj.name, permissions);
        }

        public void DeleteGroup()
        {
            // method for deleting a group
            dbController.DeleteGroup(groupObj.ID);
        }
    }
}
```

LogController.cs:

```
using Entra.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Data.SQLite;

namespace Entra.Controller
{
    public class LogController
    {
        private DatabaseController dbController;
        public LogController(DatabaseController databaseController)
        {
            dbController = databaseController;
        }

        public LogModel GetLogModel(SQLiteDataReader rdr)
        {
            LogModel logModel = new LogModel();
            try
            {
                logModel = new LogModel
                {
                    ID = Convert.ToInt32(rdr["id"]),
                    userID = Convert.ToInt32(rdr["userID"]),
                    readerID = Convert.ToInt32(rdr["readerID"]),
                    accessTime = Convert.ToDateTime(rdr["accessTime"]),
                    accessResult = Convert.ToString(rdr["accessResult"])
                };
            }
            catch (Exception ex)
            {
                Console.WriteLine($"An error occured: {ex.Message}");
            }

            return logModel;
        }

        public List<LogModel> GetLatestLogs(int count = 5)
        {
            // method that gets the latest logs and returns them as a list of strings
            List<LogModel> logs = new List<LogModel>();
            SQLiteDataReader rdr = dbController.GetModelReader("log", "1", "1");

            if(rdr == null)
            {
                return null;
            }

            int i = 0;

            while (rdr.Read() && i < count)
            {
                LogModel logObj = GetLogModel(rdr);
                logs.Add(logObj);
                i++;
            }
```

```
            return logs;
        }

        public void LogEntry(int userID, int readerID, bool accessResult)
        {
            // method for logging an entry to the database
            DateTime accessTime = DateTime.Now;
            dbController.InsertLog(userID, readerID, accessTime, accessResult);
        }
    }
}
```

PermissionController.cs:

```
using Entra.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SQLite;

namespace Entra.Controller
{
    public class PermissionController
    {
        public PermissionModel permissionObj { get; set; }
        private DatabaseController dbController;

        public PermissionController(DatabaseController databaseController)
        {
            dbController = databaseController;
        }

        public void AssignModel(SQLiteDataReader rdr)
        {
            try
            {
                permissionObj = new PermissionModel
                {
                    ID = Convert.ToInt32(rdr["id"]),
                    startDay = Convert.ToInt32(rdr["startDay"]),
                    endDay = Convert.ToInt32(rdr["endDay"]),
                    startTime = Convert.ToString(rdr["startTime"]),
                    endTime = Convert.ToString(rdr["endTime"]),
                    readerID = Convert.ToInt32(rdr["readerID"])
                };
            }
            catch (Exception ex)
            {
                Console.WriteLine($"An error occured: {ex.Message}");
            }
        }
```

```
        public string GetString()
        {
            return $"{GetReaderName()} [{permissionObj.startDay}...{permissionObj.endDay}]
{permissionObj.startTime}-{permissionObj.endTime}";
        }

        public void AssignFromId(int id)
        {
            // function that asigns the model from a given id
            SQLiteDataReader rdr = dbController.GetModelReader("permission", "id",
id.ToString());
            AssignModel(rdr);
        }

        public bool AccessGranted(int day, DateTime time, int reader)
        {
            // method for checking if certain permission is grated at a given time, day
and place

            if (permissionObj.startDay <= day && permissionObj.endDay >= day)
            {
                DateTime now = DateTime.Parse($"{time.Hour}:{time.Minute}");
                DateTime start = DateTime.Parse(permissionObj.startTime);
                DateTime end = DateTime.Parse(permissionObj.endTime);

                if (start <= now && now <= end)
                {
                    if (permissionObj.readerID - 1 == reader)
                    {
                        return true;
                    }
                }
            }
            return false;
        }

        public List<PermissionController> GetAllPermissions()
        {
            // method for getting all permissions from the database
            List<PermissionModel> permissions = dbController.GetAllPermissions();
            List<PermissionController> permissionControllers = new
List<PermissionController>();

            foreach (PermissionModel permission in permissions)
            {
                PermissionController permissionController = new
PermissionController(dbController);
                permissionController.permissionObj = permission;
                permissionControllers.Add(permissionController);
            }

            return permissionControllers;
        }
```

```
        public string GetReaderName()
        {
            // method for getting the name of the reader from the database
            return dbController.GetReaderName(permissionObj.readerID);
        }

        public void CreateNewPermission(int startDay, int endDay, string startTime, string
endTime, int readerID)
        {
            // method for creating a new permission
            dbController.InsertNewPermission(startDay, endDay, startTime, endTime,
readerID);
        }

        public void DeletePermission(int id)
        {
            // method for deleting a permission
            dbController.DeletePermission(id);
        }

        public PermissionController GetPermissionByString(string permissionString)
        {
            // method for getting a permission by a given string
            List<PermissionModel> permissions = dbController.GetAllPermissions();
            foreach (PermissionModel permission in permissions)
            {
                PermissionController permissionController = new
PermissionController(dbController);
                permissionController.permissionObj = permission;
                if (permissionController.GetString() == permissionString)
                {
                    return permissionController;
                }
            }
            return null;
        }

        public void UpdatePermission(int id, int startDay, int endDay, string startTime,
string endTime, int readerID)
        {
            // method for updating a permission
            dbController.UpdatePermission(id, startDay, endDay, startTime, endTime,
readerID);
        }
    }
}
```

ReaderController.cs:

```
using Entra.Model;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Data.SQLite;
using System.Windows.Forms;
using System.Text.RegularExpressions;
using System.Security.Cryptography;

namespace Entra.Controller
{
    public class ReaderController
    {
        public ReaderModel readerObj { get; set; }
        private DatabaseController dbController;

        public ReaderController(DatabaseController databaseController)
        {
            dbController = databaseController;
            readerObj = new ReaderModel();
        }

        public void AssignModel(SQLiteDataReader reader)
        {
            try
            {
                readerObj = new ReaderModel
                {
                    ID = Convert.ToInt32(reader["id"]),
                    name = Convert.ToString(reader["name"]),
                    uuid = Convert.ToString(reader["uuid"])
                };
            }
            catch(Exception ex)
            {
                MessageBox.Show($"An error occured: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void AssignFromAddress(int address)
        {
            // compare the current reader address and assign the local reader model

            SQLiteDataReader rdr = dbController.GetReaderWithAddress(address);

            if(rdr == null)
            {
                readerObj = null;
                return;
            }

            // get the address(uuid) of the memory and convert it to integer
            int readerUUID = Convert.ToInt32(rdr["uuid"]);
            AssignModel(rdr);
```

```
        }

        public void AssignFromName(string name)
        {
            SQLiteDataReader rdr = dbController.GetReaderWithName(name);
            if(rdr == null)
            {
                MessageBox.Show($"No data in database -> unable to create reader model",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            AssignModel(rdr);
        }

        public List<string> GetReaderNames()
        {
            // get all the reader names from the database
            List<string> readerNames = dbController.GetReaderNames();
            return readerNames;
        }
    }
}
```

SerialController.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO.Ports;
using System.Drawing.Drawing2D;
using System.Text.RegularExpressions;
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Security.Policy;
using System.Security.Cryptography;

namespace Entra.Controller
{
   public class SerialController
    {
        public SerialPort serialPort { get; set; }
        public string buffer { get; set; }

        public Dictionary<int, int> codes { get; set; }
        public bool dataReady { get; set; }

        public string currentCode { get; set; }
        public int currentAddr { get; set; }

        public int readCounter { get; set; }
```

```
        public SerialController(string port, int baudRate)
        {
            // set up the global objects
            serialPort = new SerialPort(port, baudRate, Parity.None, 8, StopBits.One);
            codes = new Dictionary<int, int>();
            dataReady = false;
            currentAddr = -1;
            currentCode = "";
            readCounter = 0;
        }

        public void Init()
        {
            // initialize the connection by assigning the venet handeler and opening the
port
            serialPort.DataReceived += new SerialDataReceivedEventHandler(dataReceived);
            try
            {
                serialPort.Open();
            }
            catch { }
        }

        private void dataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            // method for adding the received data to buffer
            string data = serialPort.ReadExisting();

            // add data to buffer
            buffer += data;
            // check if we have any parseable data int the buffer -> regex and capture
groups
            string pattern =
"\\u0002(\\u0004)?([ABCDEF1234567890]*)\\r\\n(\\u0003)(\\u0015)?";

            // Match the input against the pattern
            MatchCollection matches = Regex.Matches(buffer, pattern);

            // Process each match
            foreach (Match match in matches)
            {
                // Access the captured data using the named group "data"
                string g0 = match.Groups[0].Value; // full code
                string g1 = match.Groups[1].Value; // 0004
                string g2 = match.Groups[2].Value; // actual code
                string g3 = match.Groups[3].Value; // 0003
                string g4 = match.Groups[4].Value; //0015

                Console.Error.WriteLine("G2 = " + g2);


                if(g2 == "")
                {
                    return;
```

```
                }

                // after debug add the groups to the dict

                int reader_addr = 0;
                //int converted = 0;

                if (g1 != "")
                {
                    // chip keypad
                    if(g2.Length == 4)
                    {
                        // keypad code input was used
                        reader_addr = 0;
                        //converted = convertToDec(g2);
                    }
                    else if(g2.Length == 8)
                    {
                        // keypad reader was used
                        reader_addr = 1;
                        //converted = Convert.ToInt32(g2, 16);
                    }
                }
                else
                {
                    // chip only
                    reader_addr = 2;
                    //converted = Convert.ToInt32(g2, 16);
                }

                currentCode = g2;
                currentAddr = reader_addr;

                readCounter++;

                // remove byte from buffer
                int startIndex = buffer.IndexOf(g0);
                string newBuffer = buffer.Remove(startIndex, g0.Length);
                buffer = newBuffer;
            }

            buffer = "";
        }

        public int convertToDec(string code)
        {
            int numberRepre = int.Parse(code,
System.Globalization.NumberStyles.HexNumber); ;

            return numberRepre;
        }

        public void closeConnection()
        {
```

```
            serialPort.Close();
        }
        public void openConnection()
        {
            try
            {
                serialPort.Open();
            }
            catch
            {

            }
        }
    }
}
```

UserController.cs:

```
using Entra.Model;
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Entra.Controller
{
    public class UserController
    {
        public UserModel userObj { get; set; }

        private DatabaseController dbController;
        public GroupController groupController;

        public UserController (DatabaseController databaseController)
        {
            dbController = databaseController;
            userObj = new UserModel();
            groupController = new GroupController(databaseController);
        }

        public void AssignGroup()
        {
            // method that assigns the group to the user
            //Console.WriteLine(userObj.groupID.ToString()) ;
            SQLiteDataReader rdr = dbController.GetGroup(userObj.groupID);

            if(rdr == null)
            {
                return;
            }
```

```
            groupController.AssignModel(rdr);

        }

        public string GetGroupName()
        {
            // simple method for getting the name of the group
            return groupController.groupObj.name;
        }

        public bool HasAccess(int readerID)
        {
            // method that iterates through the permissions of the group and checks if the
user has access to the given reader
            groupController.GetAllPermissions();

            foreach(PermissionController permission in groupController.permissions)
            {
                if(permission.AccessGranted((int)DateTime.Now.DayOfWeek, DateTime.Now,
readerID))
                {
                    return true;
                }
            }
            return false;
        }

        public void AssignModel(SQLiteDataReader reader)
        {
            // method that assigns the model from the reader
            try
            {
                userObj = new UserModel
                {
                    ID = Convert.ToInt32(reader["id"]),
                    name = Convert.ToString(reader["name"]),
                    surname = Convert.ToString(reader["surname"]),
                    code = Convert.ToString(reader["code"]),
                    groupID = Convert.ToInt32(reader["groupID"]),
                };
            }
            catch(Exception ex)
            {
                MessageBox.Show($"An error occured: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void AssignFromCode(string code)
        {
            // method that assigns the model from the access code he uses
            SQLiteDataReader rdr = dbController.GetUserByCode(code); // select all columns
from user table
```

```
            if(rdr == null)
            {
                // user with the specified code does not exist
                userObj = null;
                return;
            }

            string userCode = Convert.ToString(rdr["code"]);

            if(code == userCode)
            {
                AssignModel(rdr);
                return;
            }

            // code not found -> set user to null
            userObj = null;
        }

        public void AssignFromName(string name, string surname)
        {
            // method that assigns the model from the name and surname
            SQLiteDataReader rdr = dbController.GetUserByName(name, surname); // select
all columns from user table

            if (rdr == null)
            {
                userObj = null;
                return;
            }

            AssignModel(rdr);
        }

        public void CreateUser()
        {
            // method that creates a new user
            dbController.InsertUser(userObj.name, userObj.surname, userObj.code
,userObj.groupID);
        }

        public void UpdateUser(string code, int groupID)
        {
            // method that updates the user
            dbController.UpdateUser(userObj.ID, userObj.name, userObj.surname, code,
groupID);
        }

        public void DeleteUser()
        {
            // method that deletes the user
            dbController.DeleteUser(userObj.ID);
        }
    }
```

```
}
```

GroupModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entra.Model
{
    public class GroupModel
    {
        public int ID { get; set; }
        public string name { get; set; }
    }
}
```

GroupPermissionModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entra.Model
{
    public class GroupPermissionModel
    {
        public int ID { get; set; }
        public int groupID { get; set; }
        public int permissionID { get; set; }
    }
}
```

LogModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entra.Model
{
    public class LogModel
    {
        public int ID { get; set; }
        public int userID { get; set; }
        public int readerID { get; set; }
        public DateTime accessTime { get; set; }
        public string accessResult { get; set; }
    }
}
```

PermissionModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Entra.Model
{
    public class PermissionModel
    {
        public int ID { get; set; }
        public int startDay { get; set; }
        public int endDay { get; set; }
        public string startTime { get; set; }
        public string endTime { get; set; }
        public int readerID { get; set; }
    }
}
```

ReaderModel.cs:

```
using Entra.Controller;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SQLite;

namespace Entra.Model
{
    public class ReaderModel
    {
        public int ID { get; set; }
        public string name { get; set; }
        public string uuid { get; set; } // ideally uuid representing the decoding schema for
regex
    }
}
```

UserModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Policy;
using System.Text;
using System.Threading.Tasks;

namespace Entra.Model
{
    public class UserModel
    {
        public int ID { get; set; }
        public string name { get; set; }
        public string surname { get; set;}
        public string code { get; set; }
        public int groupID { get; set; }
    }
}
```

Odpovědi na otázky:

1. Uveďte dva klíčové parametry, které budete zvažovat při pořizování RFID čipů do již existujícího systému.

Prvním klíčovým parametrem je definitivně kompatibilita s již existujícím systémem. V dnešní době existuje několik frekvencí, na kterých fungují jak čtečky a čipy, tudíž pořizovat do systému pracujícího s frekvencí 125kHz čip pro frekvence 1356Mhz není nejlepší investicí.

Druhým parametrem je určitě bezpečnostní standard karet. Vybírat je ideální ten nejvyšší, jelikož se jedná o typ karet, jenž se nedají zkopírovat konvenčními čtečkami a jejich obsah je šifrován. Ovšem bezpečnostní třída RFID čipů souvisí s podporovanou frekvencí, tudíž je důležité vybírat co nejvyšší bezpečnostní standard pro podporované frekvence čteček.

2. Popište základní princip RFID identifikace a pokuste se zamyslet nad zranitelnostmi tohoto systému.

Základní princip RFID identifikace spočítá v konstrukci samotné čtečky a karet, jelikož při přiblížení čipu ke čtečce je v kartě indukované napětí, které nabije interní kondenzátor a ten pak dovolí mikročipu v kartě rádiově vyslat své UID (Unique Identifier), které může mít 32, 56 nebo až 80 bitů.

Identifikace uživatele pak probíhá pomocí vyhledání osoby v databázi za pomocí tohoto UID a následného povolení/odepření přístupu. Ovšem díky tomu, jak RFID čipy fungují a jak se s nimi v dnešní době zacházeno, daly by se z bezpečnostního hlediska přirovnat k méně bezpečné variantě klasického klíče od bytu. Důvod, proč je tento čip slabším klíčem je jeho jednoduchá kopie, ke které stačí pouhé přiblížení se k čipu oběti tak, aby bylo načteno jeho UID. To ovšem platí pouze v případech, kdy se jedná o hackera amatéra co používá komerční čtečky. V případech, kdy hacker opravdu ví, co dělá a použije velkou cívku schovanou do pouzdra kufru či do kabátu, není nutný ani blízký kontakt s čipem, stačí běžný odstup v metru a RFID čip a bezpečnost budovy či sejfu je kompromitována.

3. V rámci našeho modelu je klíčovým prvkem vyhodnocení informace počítač. Nastiňte řešení, které by bylo robustní a odolné i proti výpadkům komunikace při zachování funkce. (tip: představte si řešení např. na úrovni hotelu).

V takovémto případě by bylo ideální nasadit místo počítače mikrokontroler s light-weight operčaním systémem pracujícím pouze na úrovni příkazové řádky, přičemž jedinou aplikací, která by neustále běžela by byl back-end přístupového systému společně s webserverem (apache2 či nginx), sloužícím jako konfigurační rozhraní pro admina. Kvůli tomu, aby se systém stal 100% odolným proti výpadkům bych k mikrokontroleru připojil UPS stanici a nakonfiguroval ho tak, že pokud není v dosahu jeho mateřská síť (vypadl proud), vytvoří si svou síť pro administraci. Předpokladem tohoto řešení je samozřejmě i to, že na UPS stanici budou připojeny zámky a čtečky v systému. Jakákoliv komunikace mezi řídící jednotkou a samotnými čtečkami by pak probíhala na STP kabelu.

Závěr:

Výsledkem řešení této úlohy je naprosto funkční program pro vyhodnocování oprávnění přístupu u uživatelů a z 90% dodělaná část administrátorského panelu, přičemž zbývajících 10% je pouze debugging určitých neočekávaných eventů jež nastávají v průběhu běhu, tudíž by byla aplikace po investici cca 3 hodin intenzivního debuggingu naprosto funkční a schopna být nasazena v produkčním prostředí. Důvodem proč není řešení kompletní ze sta procent je špatné časové rozložení práce na úloze z mé strany, jelikož jsem investoval přespříliš času do získávání dat ze čteček a jich následnému dekódování.