PRAXE

Datum: 17.10. 2023	Střední průmyslová škola, Chomutov, Školní 50, příspěvková organizace	Třída: V4
Číslo úlohy: 8.	Vstupenkový Systém	Jméno: Tomáš Bartoš

Zadání:

S využitím periferie čtečky čárového kódu sestavte C# aplikaci, která ve spojením s SQLite databází bude fungovat jako prodejní a kontrolní systém pro použití na kulturních akcích (např. divadelní a filmová představení). Každá akce je definována maximální kapacitou, názvem a termínem konání akce spolu s cenou vstupenky. Při prodeji načte obsluha svůj kód jako identifikaci prodejce a následně načítá čárové kódy kterými je každá akce identifikována. V rámci jednoho nákupu je možné nakoupit nezávisle libovolné množství vstupenek na libovolnou kombinaci akcí. Po prodeji a zaplacení s podporou dopočtení částky k vrácení obsluze je do souboru vygenerována vstupenka pro každého návštěvníka s identifikací akce a kódem pro načtení u vstupu. Zároveň jsou tyto informace uloženy do databáze systému.

Výsledné řešení musí obsahovat následující funkcionalitu:

- Jednoduché zadání nákupu s pomocí čtečky čárového kódu a pouze numerického bloku klávesnice bez myši. Zvažte efektivní řešení nákupu např. deseti stejných vstupenek.
- Korektní reakci na chybný vstup (neznámý kód akce, neplatnou vstupenku při kontrole u vstupu apod.)
- V režimu kontroly vstupu výběr kontrolované akce ze seznamu platného pro aktuální den spolu se zobrazením aktuálního počtu odbavených návštěvníků.
- Vstupenky generované do vybraného typu souboru vč. čárového kódu pro načtení (alespoň jako obrázek).
- Hodnoceno je mj. uživatelské prostředí a použitelnost aplikace, odolnost proti chybám a dodržení alespoň základních principů pro korektní návrh struktury a vazby tabulek databáze.

Teorie:

Řešení úlohy vstupenkového systému je postaveno na 2 hlavních pilířích. Prvním z nich je hardwarové hledisko řešení, mající sice roli důležitou, ale z kvantitativního hlediska menšinovou. Druhá část, tedy software samotného řešení, pak převládá a ní je stavěna většina mého řešení.

Čtečka čárových kódů, použitá v řešení, obsahuje již od výrobce jistou abstrakční vrstvu, jíž je převod analogových dat (čar na papíře) na data digitální, jimiž jsou znaky stisknuté na virtuální klávesnici. Čtečka se tak chová jako HID zařízení připojené přes USB rozhraní do počítače, přičemž ke zrychlení práce je odezva mezi stisky jednotlivých kláves minimální.

Čtení čárových kódů funguje na principu převodu analogového signálu přečteného CCD snímačem na digitální data, která mohou být dále interpretována. Když je třeba přečíst kód, laser nebo led diody osvítí povrch těsně před hlavou čtečky. Paprsek světla dopadající na tmavé části kódu je absorbován, zatímco plochy bílé ho odráží zpět do hlavy čtečky, kde ho jednotlivé CCD senzory čtou jako napěťovou hodnotu, která je nadále poslána do řídícího čipu čtečky pro následný post processing.

Po postprocessingu jsou stisky kláves zmáčknuty v tomto pořadí:

- Podržena klávesa SHIFT + zmáčknuta „numerická“ klávesa umístěna o řádek výše než znaky qwertz….

Je proto třeba při zpracovávání vstupů v programu vyfiltrovat daný SHIFT a číst pouze scan kódy číslic, popřípadě znaků.

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:

Okno navigace:

Po spuštění samotného programu se vytváří nová instance třídy Form1, sloužící jako rozcestník mezi módem čtení lístků a prodejem lístků. Jediný přijímaný input od uživatele je v tomto okně klávesa 1 a 3 na numerickém bloku klávesnice. V případě, že je stisknuta jakákoliv jiná klávesa, zobrazí se uživateli panel indikující nesprávně zmáčknutou klávesu. Ke zvýšení interaktivity je panel propojen s interním trimerem, který ho schová po uplynutí 500ms od jeho zobrazení.

Okno prodeje:

V případě, že uživatel stiskne klávesu 1 vyvolá se instance třídy SellerView, jejíž primární funkcí je obsluha zákazníka na úrovni kasy. Ihned po zavolání probíhá přiřazení objektům ke globálním proměnným a inicializace zobrazení, které v moment spuštění očekává sken čárového kódu prodejce. Mezi objekty, jež se inicializují je i objekt sloužící pro přístup do databáze (DatabaseController), který na základě existence databáze buďto vytvoří nový soubor a inicializuje tabulky, nebo se jen připojí k existujícímu souboru.

Po počáteční inicializace je očekáván vstup ve formě čárového kódu uživatele a filmu, ke kterému se bude vztahovat daný prodej. Obě tyto funkce jsou na počátku obsluhovány za pomocí metody keyDown, která na základě aktuálního stavu proměnných readingQuantity a readingReceived rozhoduje o funkci, jež má vyhodnotit aktuálně stisknutou klávesu.

Jelikož se obě tyto proměnné nastavují až v pozdější fázi běhu programu, zpracovává se kód prodejce a filmu pomocí metody HandleBarCodeInput. Ta po zavolání porovnává hodnotu scan kódu přidělenou v parametru s hodnotou klávesy enter. Pokud nebyla aktuální klávesa enter je do globálního pole s názvem scannedCode přidána ASCII hodnota scan kódu. V případě, že naopak byla klávesa stisknuta, je vyvolána metoda HandleSellerCode.

HandleSellerCode vyhodnotí pomocí objektů prodejce a akce fázi prodeje a v případě, kdy je zadáván kód prodejce je vyvolána metoda HandleSellerCode, zatímco v případě čtení kódu filmu HandleFilmCode. Obě z těchto metod mají pak velmi podobnou funkcionalitu: za pomocí objektu DatabaseController si zjistí, zda v databázi existuje záznam o filmu či prodejci a poté zpřístupní příslušné prvky v zobrazení formuláře tak, aby mohl prodej plynule pokračovat.

Po úspěšném zadání kódu filmu je zaktivněn vstup pro zadávání počtu vstupenek. V této fázi programu je sice stále použit event handler keyDown, avšak v potaz jsou brány pouze klávesy enter, důvodem je možnost číst vstup uživatele přímo z numeric up down pole.

V moment, kdy uživatel klikne na klávesu enter, je předpokládáno pokračování v nákupu lístků, tudíž se pouze vygeneruje specifický počet lístků do RAM paměti a program je přiveden do fáze čtení kódu filmu. U generování lístků je důležité poukázat na funkci generování kódu lístků, který musí být vždy unikátní. Ideální je použít k vygenerování více unikátních faktorů jako je třeba aktuální čas v milisekundách, MAC adresa zařízení, náhodně vygenerované číslo, náhodně vygenerované pole bitů a další. Čím více faktorů je použito, tím vyšší záruka na unikátnost kódu je.

Jakmile je třeba ukončit aktuální prodej a prodejce stiskl klávesu plus, počne proces permanentního ukládání dat jak do souboru, tak do databáze. Prvotně se lístky uloží ve formátu PDF do souboru. Toho je docíleno iterací skrz všechny objekty lístků uložených v poli s názvem tickets a následnou tvorbou PDF dokumentu a obrázku unikátního kódu za užití knihoven: PdfSharp a ZXing. Následně se pak všechny lístky z paměti uloží do databáze a zobrazí se informace o celkové ceně nákupu s numeric up down, kam prodejce zadá obdrženou částku, načež je mu buďto oznámena chyba (obdržel málo) či je ukázána částka pro vrácení. Jakmile prodejce peníze vrátil, či oznámil chybu, metoda HandleReceivedInput po něm očekává ještě jeden stisk pro plné ukončení transakce, kdy se z paměti RAM smaže pole s lístky a program přejde do fáze čtení kódu filmu.

Okno kontroly:

Pokud byla v okně navigace stisknuta klávesa 3 na numerickém bloku, vytvořila se instance zobrazení CheckView, která svou tvorbou automaticky otevřela nové okno, tentokrát zaměřené na kontrolu lístků před vstupem do sálu.

Ihned po tvorbě instance objektu se na pozadí inicializují globální objekty, timer sloužící pro interaktivní poskytnutí zpětné vazby a prvky v zobrazení. Důležitou součástí inicializace prvků v zobrazení je pak metoda FillEventData, která za pomocí metody GetEventsToDate z objektu DatabaseController získává data o akcích v daný den a následně je zobrazí v tabulce reprezentované prvkem DataGridView společně s přidáním jich do globálního slovníku validEventDict;

Jakmile jsou všechny aspekty zobrazení zinicializované, zaktivní se prvek NumericUpDown a pomocí Event handleru je zjišťován stisk klávesy enter. Jakmile stisk nastane, zkontroluje se platnost dat vůči slovníku validEventDict. Pokud by bylo ID v nesprávném rozsahu, je o tom uživatel informován zobrazením MessageBoxu s chybovou hláškou a smazáním neplatného ID. V opačném případě se za pomocí funkce AssignWithID přiřadí globálnímu objektu eventController data z databáze o aktuální akci a program se přesouvá na fázi samotné kontroly lístků.

Při přepnutí do fáze kontroly lístků se zablokují prvky pro zadávání ID akce a zaktivují se prvky pro příjem naskenovaného kódu a poskytnutí zpětné vazby obsluze. Provede se taky aktualizace počtu odbavených klientů pro aktuální akci za užití objektu DatabaseController a metody GetUsedTickets.

EventHandler pak nereaguje pouze na stisk, ale díky nastavení globální proměnné readingEvent na false při přepnutí fází jsou nyní čteny stisky číslic na klávesy a enteru. Funkce zodpovědná za práci s kódy je potom HandleBarcodeInput, která dokud není stisknutý enter vypočítá reálnou hodnotu klávesy pomocí odečtení čísla 48 od scan kódu a následným porovnáním, zda se nachází v rozsahu od 0 do 9 včetně buďto přidá spočtenou hodnotu do globálního pole scannedCode, či ne. V moment stisku enteru se volá metoda HandleScannedCode.

HandleScannedCode pomocí objektu DatabaseController a metody CheckTicketValidity zjistí, zda je lístek platný a na základě toho zavolá metodu ShowInvalidTicket nebo ShowValidTicket, obě, z nichž zobrazí na 750ms panel značící úspěšnost čtení.

Pokud byl lístek platný nastaví se pomocí objektu DatabaseController a metody SetTicketUsed lístek na použitý a proběhne restart vstupu pro kód lístku tak, aby bylo možné odbavit dalšího klienta.

Rozbor Proměnných a metod:

Metody:

Třída Form1			
Typ	**Název**	**Argumenty**	**Účel**
void	hideError	object sender, ElapsedEventArgs e	Skrití panelu erroru
void	keyDown	object sender, KeyEventArgs e	Zpracování stisknutí kláves

Třída SellerView			
Typ	**Název**	**Argumenty**	**Účel**
void	HandleReceivedInput	int keyVal	Zpracování vtupů obdržené částky
void	HandleBarcodeInput	int keyVal	Zpracování vstupu ze čtečky
void	HandleQuantityInput	int keyVal	Zpracování vstupu počtu lístků
void	HandleFilmCode	void	Metoda pro zpracování kódu dilmu

Třída CheckView			
Typ	**Název**	**Argumenty**	**Účel**
void	FillEventData	void	Metoda pro naplnění
void	HandleEventInput	int keyVal	Metoda pro zpracování zdaného ID akce
void	HandleBarcodeInput	int keyVal	Metoda pro zpracování vstupů při skenu čárových kódů
void	HandleScannedCode	void	Metoda rozhodující o dalším postupu programu na základě platnosti lístku

Třída DatabaseController			
Typ	**Název**	**Argumenty**	**Účel**
void	InitDatabase	void	Metoda pro inicializaci databáze.
void	CreateTables	void	Metoda pro vytvoření tabulek, pokud již neexistují
SQLiteDataReader	GetModelReader	string table, string column, string code	Univerzální Metoda pro získání objektu obsahujícího data z databáze.
int	Entries	string table	Metoda pro získání počtu záznamů v dané tabulce.
void	SaveSale	SaleController saleController	Metoda sloužící k uložení dat o prodeji do databáze.
void	SaveTickets	TicketController ticketController	Metoda pro uložení aktuálních lístků do databáze.
EventModel	GetEventFromTicket	TicketModel ticket	Metoda pro získání dat za užití kódu aktuálně naskenovaného lístku.
int	GetMaxTicketID	int eventID	Metoda pro získání maximálního ID lístku z databáze.
Dictionary<int, String>	GetEventsToDate	DateTime currentDate	Metoda pro získání akcí platných pro aktuální datum.
bool	CheckTicketValidity	int eventID, string ticketCode	Metoda pro zjištění platnosti lístku za pomocí jeho kódu.
void	SetTicketUsed	string ticketCode	Metoda aktualizující stav lístku na použitý.
int	GetUsedTickets	int eventID	Metoda pro získání počtu nepoužitých lístků.

Třída EventController			
Typ	**Název**	**Argumenty**	**Účel**
void	AssignModel	SQLiteDataReader reader	Metoda pro přiřazení dat k internímu objektu akce z dat získaných z databáze.
int	GetFreeCapacity	void	Metoda pro získání počtu volných lístků akce.

Třída PrintController			
Typ	**Název**	**Argumenty**	**Účel**
void	GenerateTicketPDF	string ticketPath, string ticketCode, string eventName, DateTime eventDate, float ticketPrice	Metoda generuje soubor typu pdf s informacemi o lístku a akce.
XImage	GenerateBarcode	string text	Metoda pro vygenerování objektu Ximage obshaující čárový kód lístku.

Třída SaleController			
Typ	**Název**	**Argumenty**	**Účel**
void	CreateSale	SellerModel saleSeller	Metoda pro vytvoření objektu SaleModel a přiřazení mu data.

Třída SellerController			
Typ	**Název**	**Argumenty**	**Účel**
void	AssignModel	SQLiteDataReader reader	Metoda pro přiřazení dat k internímu ibjektu prodejce z dat získaných z databáze.

Třída TicketController			
Typ	**Název**	**Argumenty**	**Účel**
void	GenerateTickets	int n, EventModel ticketEvent, SaleModel sale	Metoda generuje specifikovaný počet lístků a uloží je do poaměti jako objekty.
void	GetTicketCode	int ticketID	Metoda generuje unikátní kód lístku.
void	SaveTicketPdfs	void	Metoda iteruje skrz objekty lístků v paměti a vyvolá metodu k uložení lístků do paměti.
int	GetTicketID	TicketModel ticket	Funkce získá nejmenší použitelné unikátní ID lístku.

Proměnné:

Třída Form1		
Typ	**Název**	**Účel**
System.Timers.Timer	error_timer	Časovač pro zobrazování chybové hlášky.

Třída SellerView		
Typ	**Název**	**Účel**
DatabaseController	dbController	Objekt kontroleru pro operace s databází.
SellerController	sellerController	Objekt kontroleru pro manipulaci s prodejci.
EventController	eventController	Objekt kontroleru pro manipulaci s akcemi.
TicketController	ticketController	Objekt kontroleru pro manipulaci s vstupenkami.
SaleController	saleController	Objekt kontroleru pro manipulaci se prodeji.
string	scannedCode	Ukládá charaktery načteného kódu.
bool	readingQuantity	Označuje, zda je program v režimu čtení množství.
bool	readingReceived	Označuje, zda je program v režimu čtení obdržené částky.
int	receivedAcknowledged	Sleduje stav potvrzení obdržené sumy.

Třída CheckView		
Typ	**Název**	**Účel**
DatabaseController	dbController	Objekt kontroleru pro operace s databází.
EventController	eventController	Objekt kontroleru pro manipulaci s akcemi.
System.Timers.Timer	notificationTimer	Časovač pro zobrazení panelu s validitou nebo nevaliditou lístku.
string	scannedCode	Ukládá čísla načteného kódu z čárového kódu.
bool	readingEvent	Určuje, zda je program v režimu čtení ID události.
bool	errorNotification	Uchovává aktuální stav validity lístku.
Dictionary<int, String>	validEventDict	Slovník obsahující platné ID událostí a jejich názvy.

Třída DatabaseControler		
Typ	**Název**	**Účel**
string	path	Řetězec pro ukládání cesty k souboru databáze.
SQLiteConnection	connection	Objekt obsahující informace o aktuálním spojení s databází.
SQLiteCommand	command	Objekt sloužící k vykonávání SQL příkazů.

Třída EventController		
Typ	**Název**	**Účel**
EventModel	eventObj	Obsahuje objekt aktuální akce.
DatabaseController	dbController	Instance třídy pro operace s databází.

Třída PrintController		
Typ	**Název**	**Účel**
PdfDocument	document	Instance pomocné třídy pro tvorbu PDF dokumentu.
XGraphics	gfx	Instance pomocné třídy pro přidání grafických prvků do PDF dokumentu.
int	xOffset	Hodnota odsazení z levé strany lístku.
int	yOffset	Hodnota odsazení od shora lístku.
int	lineHeight	Výška řádku při zápisu více řádků na lístek.
BarcodeWriter	barcodeWriter	Instnace pomocné třídy pro tvorbu čárového kódu.

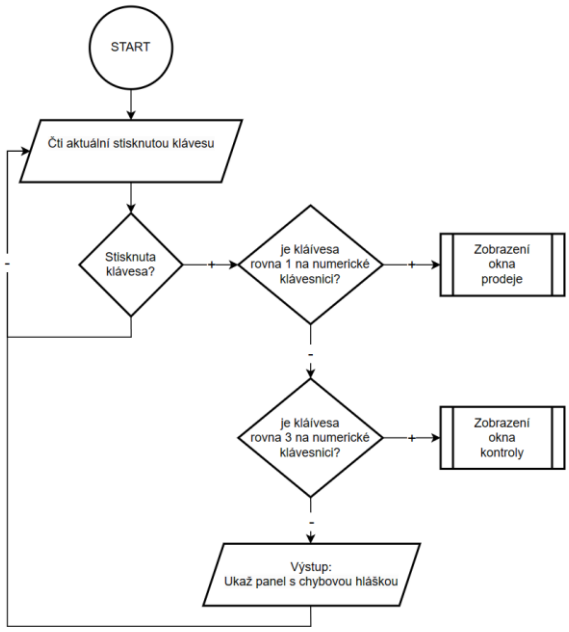
Třída SaleController		
Typ	**Název**	**Účel**
DatabaseController	dbController	Objekt kontroleru pro operace s databází.
SaleModel	saleObj	Obsahuje objekt aktuální akce.

Třída SellerController		
Typ	**Název**	**Účel**
SellerModel	sellerObj	Obsahuje objekt aktuálním prodejci.

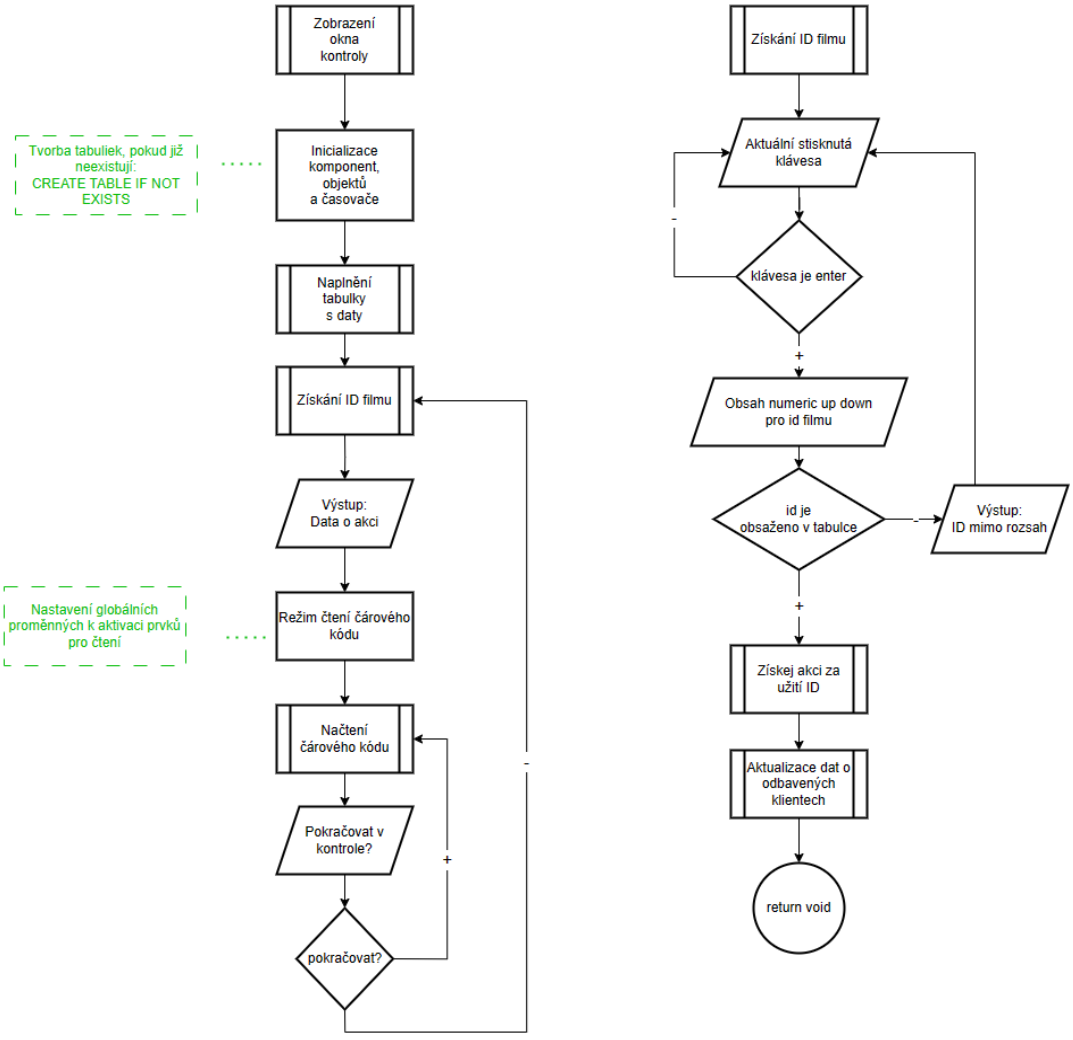
Třída TicketController		
Typ	**Název**	**Účel**
DatabaseController	dbController	Objekt kontroleru pro operace s databází.
PrintController	printController	Objekt kontroleru pro tickové operace.
float	totalTicketPrice	Číslo reprezentující celkovou cenu lístků pro jeden prodej.
List<TicketModel>	tickets	List obsahující všechny lístky pro aktuální prodej.
Random	random	Instance třídy Radnom sloužící pro vygenerování unikátního kódu lístku.

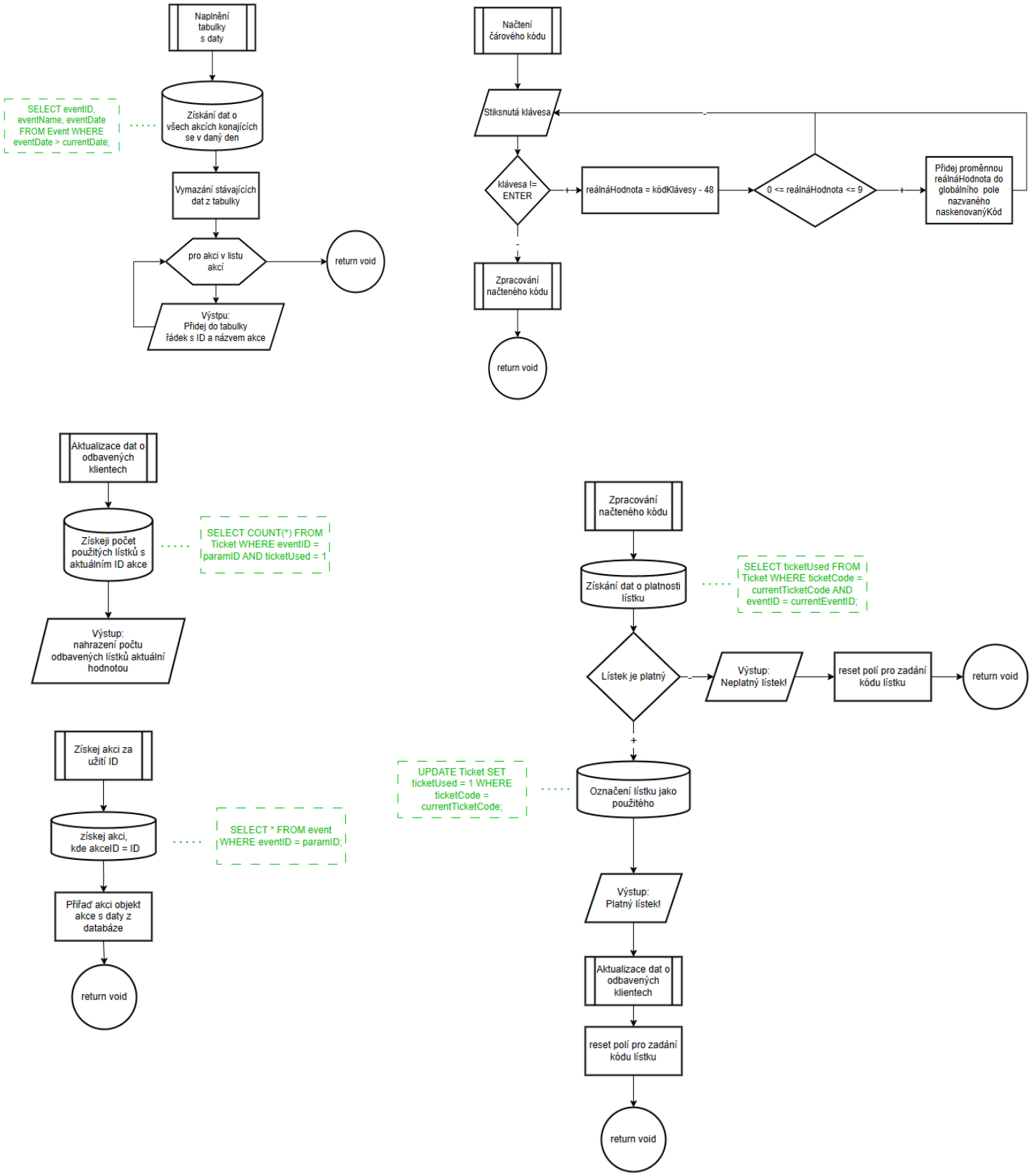
Vývojový diagram:

a. Vývojový diagram pro výběr módu:

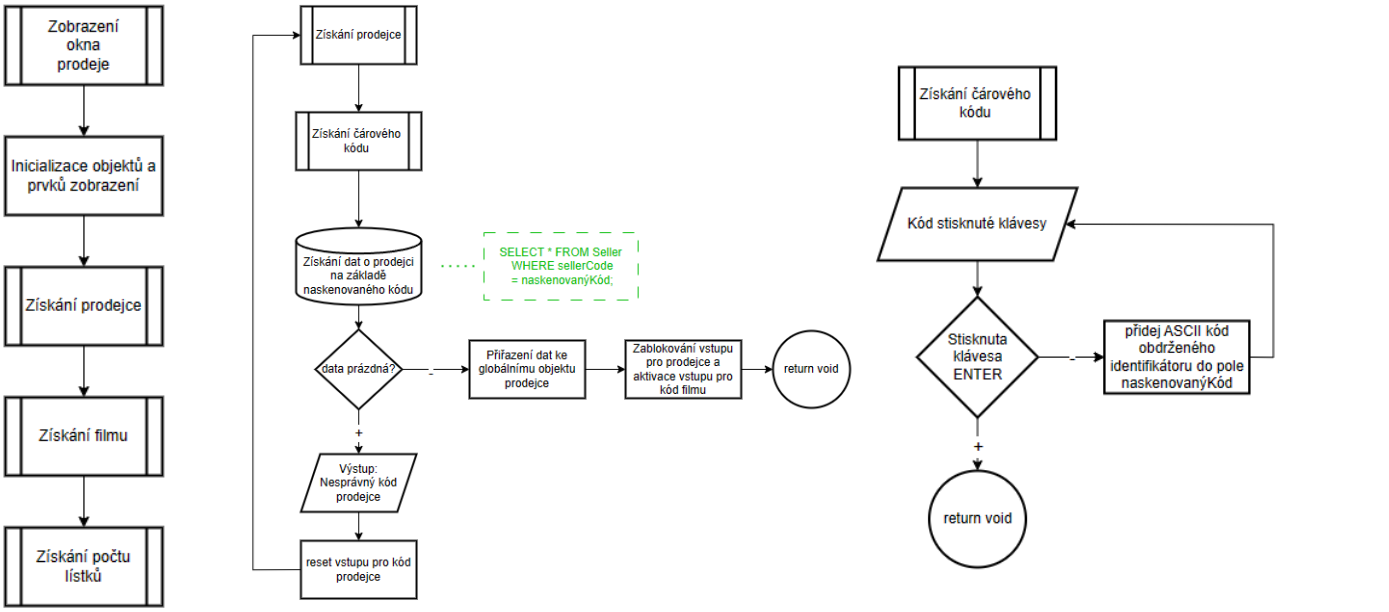


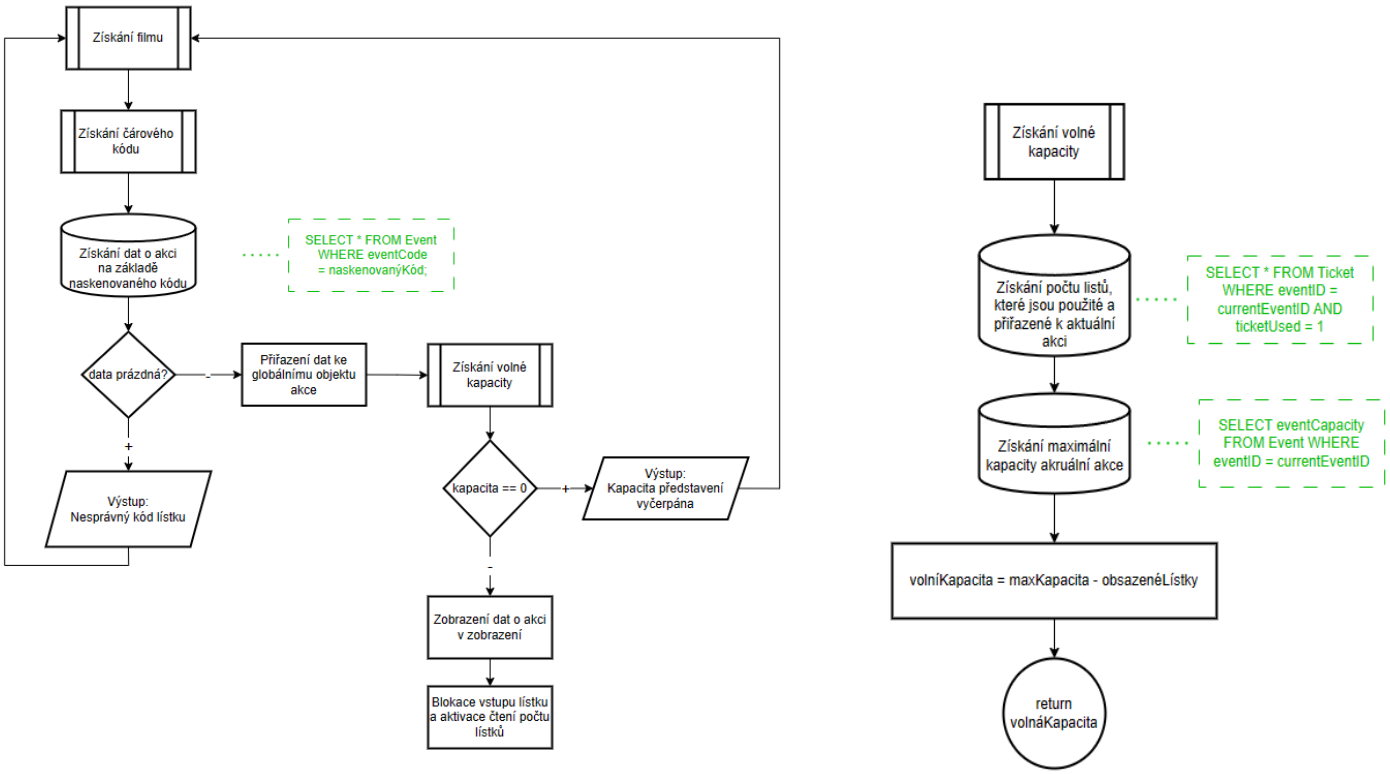
b. Vývojový diagram pro zobrazení kontroly:

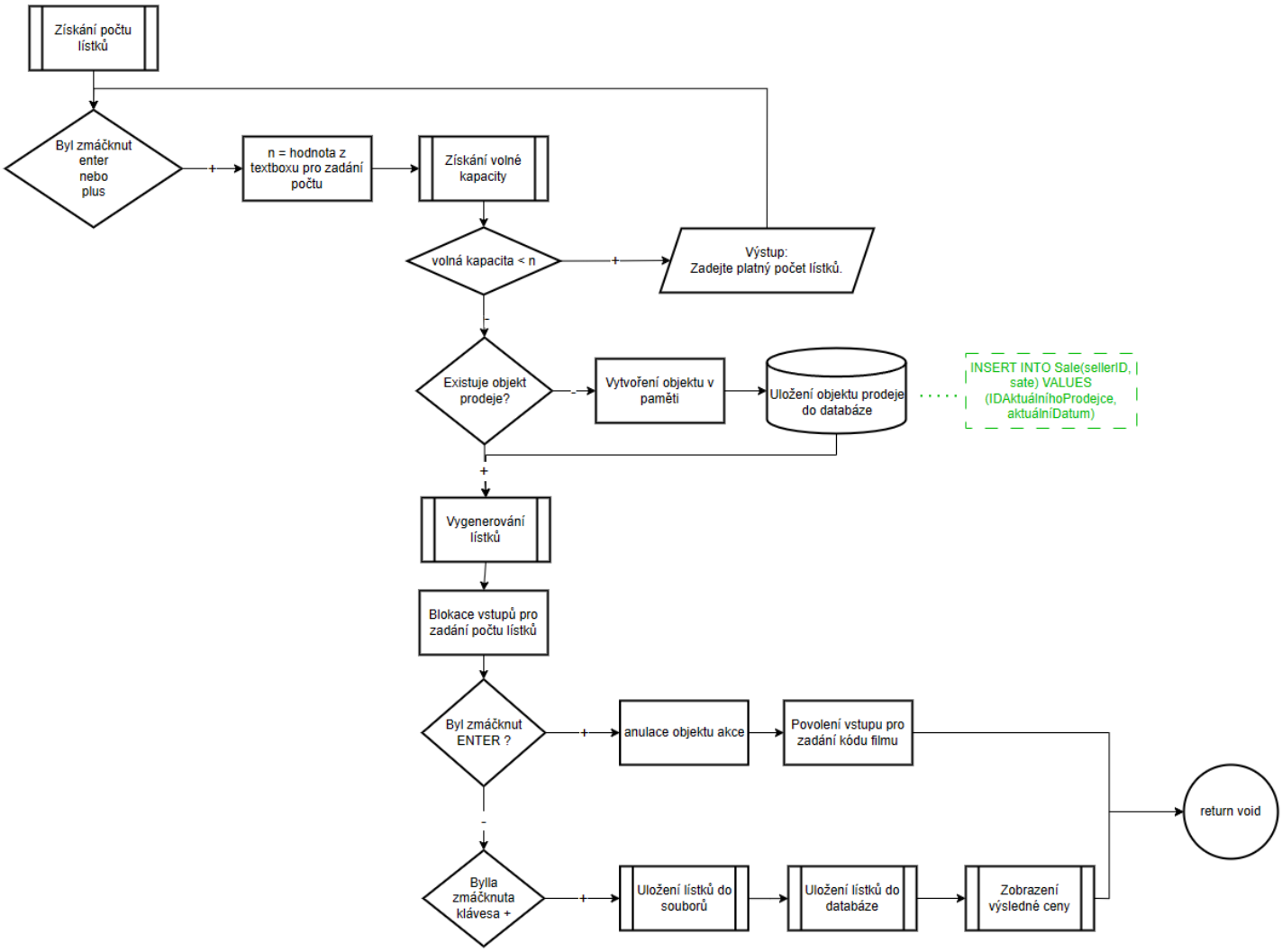


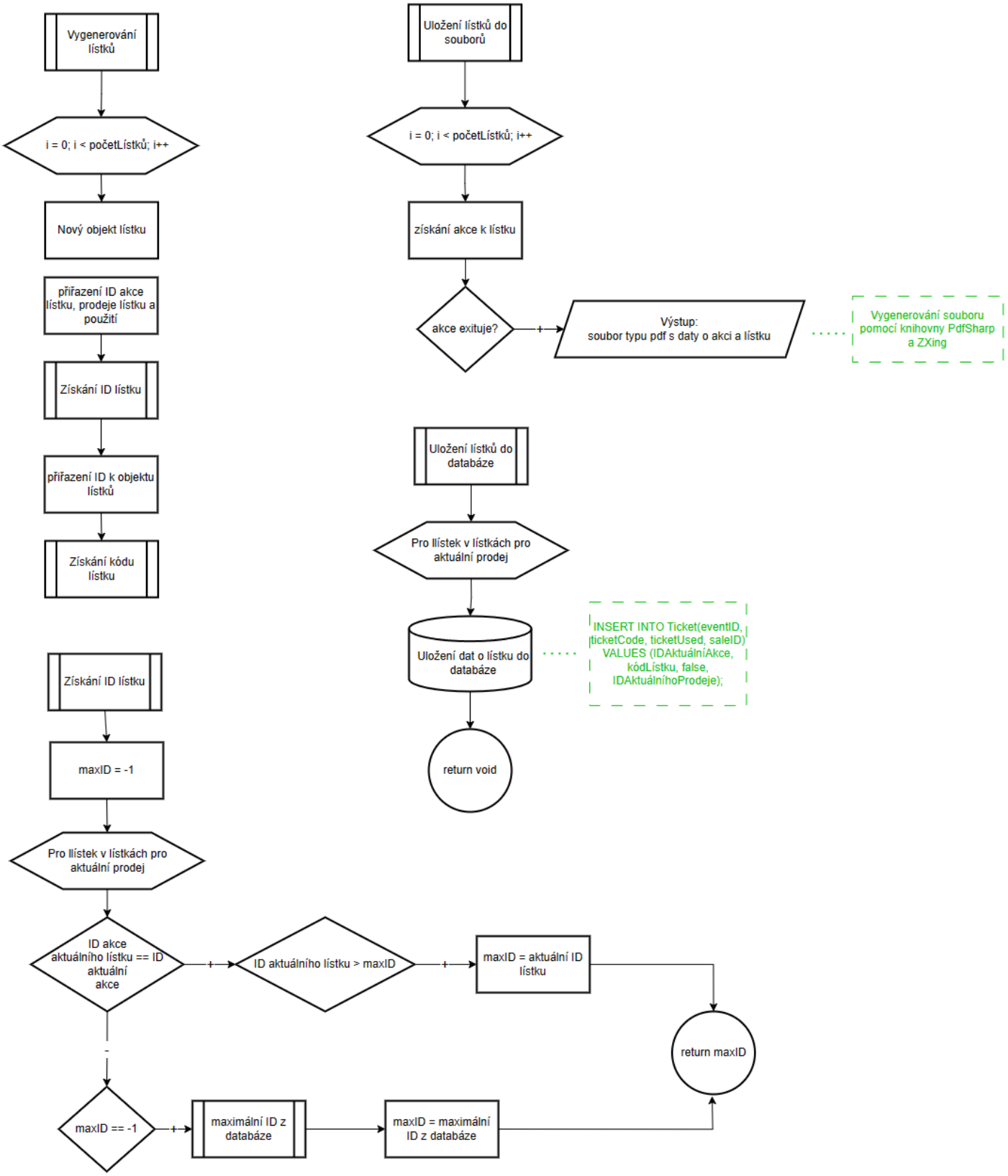

Naplnění tabulky s daty
SELECT eventID, eventName, eventDate FROM Event WHERE eventDate > currentDate;
Získání dat o všech akcích konajících se v daný den
Vymazání stávajících dat z tabulky
pro akci v listu akcí
return void
Výstpu: Přidej do tabulky řádek s ID a názvem akce
Načtení čárového kódu
Stiksnutá klávesa
klávesa != ENTER
reálnáHodnota = kódKlávesy - 48
0 <= reálnáHodnota <= 9
Přidej proměnnou reálnáHodnota do globálního pole nazvaného naskenovanýKód
Zpracování načteného kódu
return void
Aktualizace dat o odbavených klientech
Získeji počet použitých lístků s aktuálním ID akce
SELECT COUNT(*) FROM Ticket WHERE eventID = paramID AND ticketUsed = 1
Výstup: nahrazení počtu odbavených lístků aktuální hodnotou
Získej akci za užití ID
získej akci, kde akceID = ID
SELECT * FROM event WHERE eventID = paramID;
Přiřaď akci objekt akce s daty z databáze
return void
Zpracování načteného kódu
Získání dat o platnosti lístku
SELECT ticketUsed FROM Ticket WHERE ticketCode = currentTicketCode AND eventID = currentEventID;
Lístek je platný
Výstup: Neplatný lístek!
reset polí pro zadání kódu lístku
return void
UPDATE Ticket SET ticketUsed = 1 WHERE ticketCode = currentTicketCode;
Označení lístku jako použitého
Výstup: Platný lístek!
Aktualizace dat o odbavených klientech
reset polí pro zadání kódu lístku
return void

c. Vývojový diagram pro zobrazení prodeje:






Získání počtu lístků
Byl zmáčknut enter nebo plus
+
n = hodnota z textboxu pro zadání počtu
Získání volné kapacity
volná kapacita < n
+
Výstup: Zadejte platný počet lístků.
-
Existuje objekt prodeje?
-
Vytvoření objektu v paměti
Uložení objektu prodeje do databáze
INSERT INTO Sale(sellerID, sate) VALUES (IDAktuálníhoProdejce, aktuálníDatum)
+
Vygenerování lístků
Blokace vstupů pro zadání počtu lístků
Byl zmáčknut ENTER ?
+
anulace objektu akce
Povolení vstupu pro zadání kódu filmu
-
Bylla zmáčknuta klávesa +
+
Uložení lístků do souborů
Uložení lístků do databáze
Zobrazení výsledné ceny
return void


Vygenerování lístků
i = 0; i < početLístků; i++
Nový objekt lístku
přiřazení ID akce lístku, prodeje lístku a použití
Získání ID lístku
přiřazení ID k objektu lístků
Získání kódu lístku
Uložení lístků do souborů
i = 0; i < početLístků; i++
získání akce k lístku
akce exituje?
+
Výstup:
soubor typu pdf s daty o akci a lístku
Vygenerování souboru pomocí knihovny PdfSharp a ZXing
Uložení lístků do databáze
Pro lístek v lístkách pro aktuální prodej
Uložení dat o lístku do databáze
INSERT INTO Ticket(eventID, ticketCode, ticketUsed, saleID) VALUES (IDAktuálníAkce, kódLístku, false, IDAktuálníhoProdeje);
return void
Získání ID lístku
maxID = -1
Pro lístek v lístkách pro aktuální prodej
ID akce aktuálního lístku == ID aktuální akce
+
ID aktuálního lístku > maxID
+
maxID = aktuální ID lístku
return maxID
-
maxID == -1
+
maximální ID z databáze
maxID = maximální ID z databáze

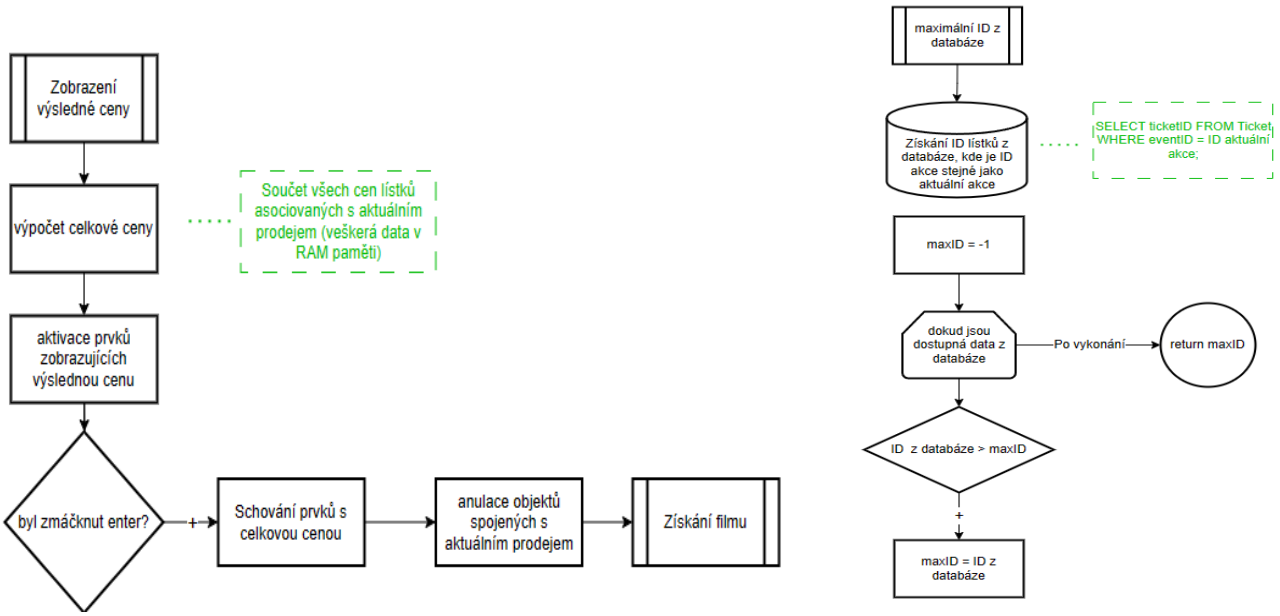
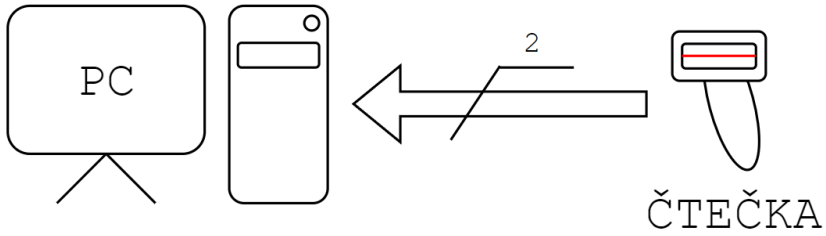
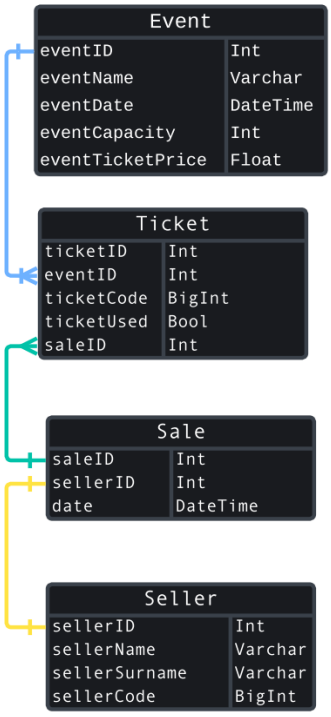


Schéma zapojení:

a. Schéma fyzického zapojení:



b. Schéma relací v databázi:



Komentovaný výpis program:

Form1.cs

```
using CodeScanner.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;

namespace CodeScanner
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private System.Timers.Timer error_timer; // timer used for displaying an error panel
        private void Form1_Load(object sender, EventArgs e)
        {
            this.KeyPreview = true; // enable handling key presses

            // view setup
            error_panel.Visible = false;

            // timer setup
            error_timer = new System.Timers.Timer();
            error_timer.Interval = 500;
            error_timer.Elapsed += hideError;
            error_timer.AutoReset = false;
        }

        private void hideError(object sender, ElapsedEventArgs e)
        {
            // hide the error message when timer stops

            Invoke(new Action(() => { error_panel.Visible = false; })); // perform a cross
thread operation to edit the form view
        }

        private void keyDown(object sender, KeyEventArgs e)
        {
            if(e.KeyCode == Keys.NumPad1)
            {
                // open sell mode form
                film_price_label sellerView = new film_price_label();
                sellerView.Show();
                return;
            }
            else if(e.KeyCode == Keys.NumPad3)
            {
                // open check mode form
                CheckView checkView = new CheckView();
                checkView.Show();
                return;
            }
            else
            {
                // show erroe -> user clicked on a different key
                error_panel.Visible = true;
                error_timer.Start();
            }
        }
    }
}
```

SellerView.cs

```
using CodeScanner.Controller;
using CodeScanner.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.Windows.Forms;
using System.Data.SQLite;
using System.Diagnostics.Eventing.Reader;
using System.Security.AccessControl;

namespace CodeScanner.View
{
    public partial class film_price_label : Form
    {
        // CONTROLLERS
        DatabaseController dbController { get; set; }
        SellerController sellerController { get; set; }
        EventController eventController { get; set; }
        TicketController ticketController { get; set; }
        SaleController saleController { get; set; }

        // GLOBAL VARIABLES
        string scannedCode = "";
        public bool readingQuantity = false;
        public bool readingReceived = false;
        public int recievedAcknowledged = -1;


        public film_price_label()
        {
            InitializeComponent();
        }

        private void SellerView_Load(object sender, EventArgs e)
        {
            GlobalMethodsSetup();
            ViewSetup();
        }

        private void GlobalMethodsSetup()
        {
            // function for setting up the Controller objects
            dbController = new DatabaseController();
            sellerController = new SellerController();
            eventController = new EventController(dbController);
            ticketController = new TicketController(dbController);
            saleController = new SaleController(dbController);
        }
```

```
        private void ViewSetup()
        {
            // function for setting up the view parameters on startup

            this.KeyPreview = true; // read input from keyboard

            // seller box init
            seller_name_input.Enabled = false;
            seller_surname_input.Visible = false;

            seller_surname_input.Enabled = false;
            seller_surname_input.Visible = false;

            seller_code_input.Focus();

            //film box init
            film_code_group.Enabled = false;

            // ticket box init
            ticket_input_box.Enabled = false;
            ticket_input.Minimum = 1;

            // price box init
            price_box.Visible = false;
            price_received_input.Minimum = 0;
            price_received_input.Maximum = decimal.MaxValue;
        }

        private void keyDown(object sender, KeyEventArgs e)
        {
            // function for handling the key press

            if (readingQuantity)
            {
                HandleQuantityInput(e.KeyValue);
            }
            else if (readingReceived)
            {
                HandleReceivedInput(e.KeyValue);
            }
            else
            {
                HandleBarcodeInput(e.KeyValue);
            }
        }

        private void HandleReceivedInput(int keyVal)
        {
            if(keyVal != 13)
            {
                // if enter received -> ignore it
                return;
            }

            if(recievedAcknowledged == 0)
```

```
            {
                // if user has ackowledged the return sum -> switch to film-reading mode
                recievedAcknowledged = -1;
                readingQuantity = false;
                readingReceived = false;
                ResetSaleData();
                DisableTotal();
                EnableFilmInput();
                return;
            }

            int received = (int)price_received_input.Value;
            float toBeReturned = received - ticketController.totalTicketPrice; // calculating
the return sum

            if (toBeReturned < 0)
            {
                // if return sum incalid -> show error and clear input
                ShowError("Příliš nízká obdržená částka!");
                price_received_input.Value = 0;
                price_received_input.Select(0, 1);
                return;
            }

            // show the sum to be returned and wait for acknowledge
            price_received_input.Enabled = false;
            price_return_text.Text = $"{toBeReturned} Kč";
            recievedAcknowledged++;
        }


        private void HandleBarcodeInput(int keyVal)
        {
            // function for handeling barcode input froom the reader
            if (keyVal != 13)
            {
                // if enter not pressed -> add the ASCII code to the scannedCode string
                scannedCode += keyVal.ToString();
            }
            else
            {
                // enter pressed -> check the entered code
                HandleScannedCode();
            }
        }

        private void HandleQuantityInput(int keyVal)
        {
            // function for handeling the quantity input
            if (keyVal == 13 || keyVal == 107)
            {
                // if plus or enter key were pressed, print the ticket or continue in the
sale

                int n = (int)ticket_input.Value;
```

```
                if(eventController.GetFreeCapacity() < n)
                {
                    // not enough tickets -> show error
                    ShowError("Příliš velké číslo zadaných vstupenek.");
                    ticket_input.Value = 1;
                    return;
                }

                if (saleController.saleObj == null)
                {
                    // if no sale existed, add it to memory and create a record in db
                    saleController.CreateSale(sellerController.sellerObj);
                    dbController.SaveSale(saleController);
                }

                // generate the tickets and save them to RAM
                ticketController.GenerateTickets(n, eventController.eventObj,
saleController.saleObj);
                DisbaleTicketInput();
            }

            if(keyVal == 13)
            {
                // if enter clicked -> continue in the sale
                eventController.eventObj = null;
                readingQuantity = false;
                EnableFilmInput();
            }

            if (keyVal == 107)
            {
                // save the tickets and reset the controllers to accept further data
                readingReceived = true;
                readingQuantity = false;
                ticketController.SaveTicketPdfs();
                dbController.SaveTickets(ticketController);
                EnableTotal();
            }
        }

        public void ResetSaleData()
        {
            // reset the controller objects so new data could be stored
            ticketController.tickets = new List<TicketModel>();
            eventController.eventObj = null;
            saleController.saleObj = null;
        }

        public void HandleScannedCode()
        {
            // function for deciding which handler function should be used based on the state
of the controllers
            if (sellerController.sellerObj == null)
            {
```

```
                HandleSellerCode();
            }
            else if (eventController.eventObj == null)
            {
                HandleFilmCode();
            }
            scannedCode = "";
        }

        private void HandleSellerCode()
        {
            // function for handling the barcode of a seller

            SQLiteDataReader sellerReader = dbController.GetModelReader("Seller",
"sellerCode", scannedCode);
            if (sellerReader != null)
            {
                // seller exists -> convert it to model
                sellerController.AssignModel(sellerReader);
                DisableSellerInput();
                EnableFilmInput();
            }
            else
            {
                // seller is not valid -> show error and clear input
                seller_code_input.Text = "";
                ShowError("Nesprávný kód prodejce...");
            }
        }

        private void HandleFilmCode()
        {
            // function for handeling film input
            SQLiteDataReader eventReader = dbController.GetModelReader("Event", "eventCode",
scannedCode);
            if (eventReader != null)
            {
                // event exists
                eventController = new EventController(dbController);
                eventController.AssignModel(eventReader);

                if(eventController.GetFreeCapacity() == 0)
                {
                    // event capacity exceeded -> show error clear input
                    ShowError("Kapacita představení vyčeprána!");
                    film_code_input.Text = "";
                    eventController.eventObj = null;
                    return;
                }

                ShowEventData();
                EnableTicketInput();
            }
            else
            {
```

```
            // event does not exist -> clear input and show error
            film_code_input.Text = "";
            ShowError("Nesprávný kód filmu...");
        }
    }
    private void EnableTotal()
    {
        // enable the total view elements
        price_box.Visible = true;
        price_received_input.Enabled = true;
        price_received_input.Value = 0;
        price_total_text.Text = $"{Math.Round(ticketController.totalTicketPrice, 2)} Kč";
        price_return_text.Text = "";
        price_received_input.Focus();
        price_received_input.Select(0, 1);
    }

    private void DisableTotal()
    {
        // function for disablig the total
        price_box.Visible = false;
        price_total_text.Text = $"0 Kč";
    }

    private void DisableSellerInput()
    {
        // show the seller related inputs
        seller_name_input.Visible = true;
        seller_surname_input.Visible = true;

        // set seller informatuin
        seller_name_input.Text = sellerController.sellerObj.sellerName;
        seller_surname_input.Text = sellerController.sellerObj.sellerSurname;
        seller_code_input.Enabled = false;
    }

    private void EnableFilmInput()
    {
        // function for enabling the film inputs in the view
        film_name_input.Text = "";
        film_price_input.Text = "";
        free_input.Text = "";
        occupied_input.Text = "";
        film_code_input.Text = "";
        film_code_input.Enabled = true;
        film_code_group.Enabled = true;
        film_code_input.Focus();
    }


    private void EnableTicketInput()
    {
        // function for enabling the ticket inputs in the view
        readingQuantity = true;
        ticket_input_box.Enabled = true;
```

```
            ticket_input.Enabled = true;
            ticket_input.Focus();
            ticket_input.Select(0, 1);
        }
        private void DisbaleTicketInput()
        {
            // function for disabling the ticket input in the view
            ticket_input.Value = 1;
            ticket_input_box.Enabled = false;
        }
        private void ShowEventData()
        {
            // function for filling the textboxes with event related values
            film_code_input.Enabled = false;
            film_name_input.Text = eventController.eventObj.eventName;
            free_input.Text = eventController.GetFreeCapacity().ToString();
            occupied_input.Text = eventController.eventObj.eventCapacity.ToString();
            film_price_input.Text = eventController.eventObj.eventTicketPrice.ToString();
        }


        public void ShowError(string msg)
        {
            // function for showing erros as message boxes
            MessageBox.Show($"{msg}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

    }
}
```

CheckView.cs

```
using CodeScanner.Controller;
using iText.StyledXmlParser.Jsoup.Helper;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Timers;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace CodeScanner.View
{
    public partial class CheckView : Form
    {
        // CONTROLLERS
        DatabaseController dbController { get; set; }
        EventController eventController { get; set; }

        // GLOBAL VARIABLES
        private System.Timers.Timer notificationTimer;
        private string scannedCode = "";
        private bool readingEvent = false;
        private bool errorNotification = false;
        Dictionary<int, String> validEventDict;


        public CheckView()
```

```
        {
            this.KeyPreview = true; // enable keyboard input
            InitializeComponent();
            GlobalMethodsSetup();
            TimerSetup();
            ViewSetup();
        }

        private void ViewSetup()
        {
            // event input settings
            event_input.Minimum = 0;

            // panel settings
            error_panel.Visible = false;
            success_panel.Visible = false;

            // other settings
            DataGridSetup();
            FillEventData();
            EnableEventInput();
        }

        private void TimerSetup()
        {
            // function that sets basic parameters of the notificationTimer
            // notification timer is used to show valid/invalid panels
            notificationTimer = new System.Timers.Timer();
            notificationTimer.Interval = 750;
            notificationTimer.Elapsed += HideNotification;
            notificationTimer.AutoReset = false;
        }

        private void DataGridSetup()
        {
            // function for the DataGridVivew component setup

            // set the column names
            event_list.Columns.Add("ID", "ID");
            event_list.Columns.Add("eventName", "Event Name");

            // read-only grid
            event_list.ReadOnly = true;

            // omit the default index column
            event_list.RowHeadersVisible = false;
        }
        private void GlobalMethodsSetup()
        {
            // function for inicialization of the global controller variables
            dbController = new DatabaseController();
            eventController = new EventController(dbController);
        }

        private void FillEventData()
        {
            // function for filling the DataGridView component with data from the database

            validEventDict = dbController.GetEventsToDate(DateTime.Now); // get events from db,
that happen today or in the futurue

            if(validEventDict.Count == 0)
            {
                // if no events today -> return void;
                return;
            }

            event_list.Rows.Clear();

            foreach(var eventDataEntry in validEventDict)
            {
```

```
                // for each entry in the global dictonary validEventDict containing data for the
DataGridView -> add it to the component
                int eventID = eventDataEntry.Key;
                string eventName = eventDataEntry.Value;

                event_list.Rows.Add(eventID, eventName);
            }

        }

        private void keyDown(object sender, KeyEventArgs e)
        {
            // function for handling keydown event
            if (readingEvent)
            {
                HandleEventInput(e.KeyValue);
            }
            else
            {
                HandleBarcodeInput(e.KeyValue);
            }
        }

        private void EnableEventInput()
        {
            // function for switching the programme to event input phase
            readingEvent = true;
            event_box.Enabled = true;
            event_input.Value = 0;
            event_input.Select(0, 1);
        }

        private void DisableEventInput()
        {
            // event for blocking all of the event input related elements
            readingEvent = false;
            event_box.Enabled = false;
            event_input.Enabled = false;
        }

        private void EnableTicketCodeInput()
        {
            // function for switching the programme to ticket code input phase
            ticket_box.Enabled = true;
            ticket_input.Enabled = true;
            ticket_input.Text = "";
            ticket_input.Focus();
        }

        private void HandleEventInput(int keyVal)
        {
            // function for handling the ID of event input

            if(keyVal != 13)
            {
                // if user did not click on enter -> ignore the keypress
                return;
            }

            int selectedID = (int) event_input.Value;

            if (!validEventDict.Keys.Contains(selectedID))
            {
                // if the ID is out of range -> show error and clear out the input
                EnableTicketCodeInput();
                ShowError("ID je mimo podporavaný rozsah představení.");
                return;
            }

            eventController.AssignWithID(selectedID);
```

```
            if(eventController.eventObj == null)
            {
                // if the assignment wwas not successfull -> show error
                ShowError("Interní chyba: nezdařilé získání představní z indetifikátoru.");
                return;
            }

            UpdateCheckedClients();
            DisableEventInput();
            EnableTicketCodeInput();
        }

        private void HandleBarcodeInput(int keyVal)
        {
            // function for handling ticket barcode input

            if (keyVal != 13)
            {
                // enter was not pressed
                int scannedInt = keyVal - 48; // calculate the actual value of the pressed key
                if(0 <= scannedInt && scannedInt <= 9)
                {
                    // if the actual value is in the range 0-9 -> add it to the scannedCode
global var
                    scannedCode += scannedInt;
                }
            }
            else
            {
                // enter was pressed -> assess its validity
                HandleScannedCode();
            }
        }

        private void HandleScannedCode()
        {
            // check db for entries
            // if entry exists and the ticket is not used, use it and display a proper message
            // if entry is used or does not exist show error

            if(!dbController.CheckTicketValidity(eventController.eventObj.eventID, scannedCode))
            {
                ShowInvalidTicket();
                EnableTicketCodeInput();
                return;
            }

            dbController.SetTicketUsed(scannedCode);
            ShowValidTicket();
            UpdateCheckedClients();
            EnableTicketCodeInput(); // reset inputs
        }

        private void UpdateCheckedClients()
        {
            // function for updating the number of checked clients
            int checkedClients = dbController.GetUsedTickets(eventController.eventObj.eventID);

            checked_text.Text = $"{checkedClients} Klientů";
        }

        private void ShowInvalidTicket()
        {
            // function for showing the invalid ticket panel
            errorNotification = true;
            error_panel.Visible = true;
            notificationTimer.Start();
        }

        private void ShowValidTicket()
        {
```

```
            // function for showing the valid ticket panel
            errorNotification = false;
            success_panel.Visible = true;
            notificationTimer.Start();
        }

        private void HideNotification(object sender, ElapsedEventArgs e)
        {
            // function for hiding the valid or invalid info panel

            Invoke(new Action(() => {
                // perform a cross-thread operation to edit the form view
                if (errorNotification)
                {
                    error_panel.Visible = false;
                }
                else
                {
                    // hide success panel
                    success_panel.Visible = false;
                }
            }));
        }

        private void ShowError(string msg)
        {
            // function for showing an error message in the form of message box
            MessageBox.Show($"{msg}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

DatabaseController.cs

```
using CodeScanner.Model;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.Entity;
using System.Data.SQLite;
using System.Deployment.Application;
using System.IO;
using System.Security.Cryptography;
using System.Windows.Forms;

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


        public DatabaseController(string userPath = "collosus.db")
        {
            path = userPath;
            InitDatabase();
            CreateTables();
        }

        private void InitDatabase()
```

```
        {
            // function for initialisation of the database
            if (!File.Exists(path))
            {
                try
                {
                    SQLiteConnection.CreateFile(path); // create a db file to connect to
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Nastala chyba: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

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

            // create a new SQLITE command with the connection
            command = new SQLiteCommand(connection);
        }

        private void CreateTables()
        {
            // create event table
            try
            {
                command.CommandText = @"
                    CREATE TABLE IF NOT EXISTS Event (
                        eventID INTEGER PRIMARY KEY,
                        eventName TEXT,
                        eventDate DATETIME,
                        eventCapacity INTEGER,
                        eventTicketPrice REAL,
                        eventCode TEXT
                    )";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"Chyba SQLite při tvorbě tabulky události: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
```

```
                // handle normal exception
                MessageBox.Show($"Nastala chyba: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            // create ticket table
            try
            {
                command.CommandText = @"
                    CREATE TABLE IF NOT EXISTS Ticket (
                        ticketID INTEGER PRIMARY KEY,
                        eventID INTEGER,
                        ticketCode VARCHAR,
                        ticketUsed BOOL,
                        saleID INTEGER,
                        FOREIGN KEY (eventID) REFERENCES Event(eventID),
                        FOREIGN KEY (saleID) REFERENCES Sale(saleID)

                    )";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"Chyba SQLite při tvorbě tabulky lístku: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"Nastala chyba: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            // create a seller table
            try
            {
                command.CommandText = @"
                    CREATE TABLE IF NOT EXISTS Seller (
                        sellerID INTEGER PRIMARY KEY,
                        sellerName VARCHAR,
                        sellerSurname VARCHAR,
                        sellerCode VARCHAR
                    )";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"Chyba SQLite při tvorbš tabulky prodejce: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
```

```
                MessageBox.Show($"Nastala chyba: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            // create a sale table
            try
            {
                command.CommandText = @"
                    CREATE TABLE IF NOT EXISTS Sale (
                        saleID INTEGER PRIMARY KEY,
                        sellerID VARCHAR,
                        date DATETIME,
                        FOREIGN KEY (sellerID) REFERENCES Seller(sellerID)
                    )";
                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"Chyba SQLite při tvorbě tabulky prodeje: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"Nastala chyba: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public SQLiteDataReader GetModelReader(string table, string column, string code)
        {
            // function for getting an SQLiteDataReader from a specified, table, column and
code
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = $"SELECT * FROM {table} WHERE {column} = '{code}';";

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    // if has data -> return the reader
                    return reader;
                }
            }
            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 null; // no data  =  null
        }

        public int Entries(string table)
        {
            // function for returning the number of entries in a table

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

                SQLiteDataReader reader = command.ExecuteReader();
                if (reader.Read())
                {
                    while(reader.Read())
                    {
                        // while has data -> increase the index count
                        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;
        }
        internal void SaveSale(SaleController saleController)
        {
            // function for saving data about a sale to a database
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = "INSERT INTO Sale(sellerID, sate) VALUES (@sellerID,
@date)"; // a typo in date -> change and add data later
                command.Parameters.Add("@sellerID", DbType.Int32).Value =
saleController.saleObj.sellerID;
                command.Parameters.Add("@date", DbType.DateTime).Value =
saleController.saleObj.date;
                command.ExecuteNonQuery();
```

```
            }catch(SQLiteException ex)
            {
                // handle sqlite exceptions
                MessageBox.Show($"Chyba SQLite při ukládání informací o prodeji:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch(Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"Nastala chyba systému při ukládání informací o prodeji:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        internal void SaveTickets(TicketController ticketController)
        {
            // function for saving all of the tickets to the database
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = @"INSERT INTO Ticket(eventID, ticketCode, ticketUsed,
saleID) VALUES
                                                                (@eventID, @ticketCode,
@ticketUsed, @saleID);";
            }
            catch(SQLiteException ex)
            {
                MessageBox.Show($"Chyba SQLite při tvorbě SQLite příkazu: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"Nastala chyba systému při SQLite tvorbě příkaz:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            foreach (TicketModel ticket in ticketController.tickets)
            {
                // for every ticket in the memory...
                try
                {
                    command.Parameters.Clear(); // Clear previous parameters

                    // add the necessary parameters
                    command.Parameters.Add("@eventID", DbType.Int32).Value = ticket.eventID;
                    command.Parameters.Add("@ticketCode", DbType.String).Value =
ticket.ticketCode;
                    command.Parameters.Add("@ticketUsed", DbType.Boolean).Value =
ticket.used;
                    command.Parameters.Add("@saleID", DbType.Int32).Value = ticket.saleID;

                    command.ExecuteNonQuery();
                }
                catch(SQLiteException ex)
```

```
                {
                    // handle sqlite exceptions
                    MessageBox.Show($"Chyba SQLite při přidávání {ticket.ticketID}. lístku:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                catch (Exception ex)
                {
                    // handle normal exception
                    MessageBox.Show($"Nastala chyba systému při ukládání lístku:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        internal EventModel GetEventFromTicket(TicketModel ticket)
        {
            // fuction for getting an EventModel from the TicketModel
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = "SELECT * FROM Event WHERE eventID = @ticketEventID;";
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show($"Chyba SQLite při získávání dat o lístku: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Nastala chyba systému při získávání dat o lístku:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            try
            {
                // add the ticket event parameter
                command.Parameters.Add("@ticketEventID", DbType.Int32).Value =
ticket.eventID;
                SQLiteDataReader reader = command.ExecuteReader();

                if(reader != null )
                {
                    while(reader.Read())
                    {
                        // while the reader has data -> get the EventModel from the
EventController and return it
                        EventController tmpEventController = new EventController(new
DatabaseController());
                        tmpEventController.AssignModel(reader);
                        return tmpEventController.eventObj;
                    }
                }
            }
            catch (SQLiteException ex)
            {
                // handle sqlite exceptions
```

```
                MessageBox.Show($"Chyba SQLite  při čtení dat o představení {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // handle normal exception
                MessageBox.Show($"Nastala chyba systému při čtení dat o představení:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return null;
        }

        internal int GetMaxTicketID(int eventID)
        {
            // function for getting the maximum ticket ID from databse
            int maxID = 0;

            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = "SELECT ticketID FROM Ticket WHERE eventID =
@ticketEventID;";
                command.Parameters.Add("@ticketEventID", DbType.Int32).Value = eventID;

                using (SQLiteDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        // while the reader has data

                        int entryID = reader.GetInt32(0);
                        if (entryID > maxID)
                        {
                            maxID = entryID;
                        }
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show($"Chyba SQLite při získávání dat o lístku: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Nastala chyba systému při získávání dat o lístku:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return maxID;
        }

        internal Dictionary<int, String> GetEventsToDate(DateTime currentDate)
        {
            // get a dictonary with ids and names of events for the current date
            Dictionary<int, string> upcomingEvents = new Dictionary<int, string>();
```

```
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = "SELECT eventID, eventName, eventDate FROM Event WHERE
eventDate > @currentDate;";
                command.Parameters.Add("@currentDate", DbType.DateTime).Value = currentDate;

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader != null)
                {
                    while (reader.Read())
                    {
                        // while have data
                        int eventID = reader.GetInt32(0);
                        string eventName = reader.GetString(1);

                        // Add to the dictionary
                        upcomingEvents.Add(eventID, $"{eventName}");
                    }
                }
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"Chyba SQLite při získávání dat o představení:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"Nastala chyba systému při získávání dat o představení:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            return upcomingEvents;
        }

        internal bool CheckTicketValidity(int eventID, string ticketCode)
        {
            // function for getting the data related to ticket validity from a database
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = "SELECT ticketUsed FROM Ticket WHERE ticketCode =
@ticketCode AND eventID = @eventID;";
                command.Parameters.Add("@ticketCode", DbType.String).Value = ticketCode;
                command.Parameters.Add("@eventID", DbType.Int32).Value = eventID;

                SQLiteDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    bool ticketUsed = reader.GetBoolean(0);
```

```
                    if (ticketUsed)
                    {
                        return false;
                    }
                    else
                    {
                        return true;
                    }
                }
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"Chyba SQLite při ověřování lístku: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"Nastala chyba systému při ověřování lístku: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return false;
        }

        internal void SetTicketUsed(string ticketCode)
        {
            // function for setting a ticket as used based on the ticket code
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = "UPDATE Ticket SET ticketUsed = 1 WHERE ticketCode =
@ticketCode;";
                command.Parameters.Add("@ticketCode", DbType.String).Value = ticketCode;

                command.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"Chyba SQLite při označování lístku jako použitého:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"Nastala chyba systému při označování lístku jako použitého:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        internal int GetUsedTickets(int eventID)
        {
            // function for getting the numer of used tickets based on the eventID provided
            int used = 0;
```

```
            try
            {
                command = new SQLiteCommand(connection);
                command.CommandText = "SELECT * FROM Ticket WHERE eventID = @eventID AND
ticketUsed = 1;";
                command.Parameters.Add("@eventID", DbType.Int32).Value = eventID;

                SQLiteDataReader reader = command.ExecuteReader();

                if(reader != null )
                {
                    while(reader.Read())
                    {
                        // while reader has data -> increase the used int variable
                        used++;
                    }
                }
            }
            catch (SQLiteException ex)
            {
                // Handle SQLite exceptions
                MessageBox.Show($"Chyba SQLite při získávání použitých lístků: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle other exceptions
                MessageBox.Show($"Nastala chyba systému při získávání použitých lístků:
{ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            return used;
        }
    }
}
```

EventController.cs

```
using CodeScanner.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 CodeScanner.Controller
{
    internal class EventController
    {
        // Holds the current event object
        public EventModel eventObj { get; set; }

        // Database controller instance for handling database operations
        private DatabaseController dbController { get; set; }

        // Constructor to initialize the EventController with a DatabaseController instance
```

```
        public EventController(DatabaseController databaseController)
        {
            dbController = databaseController;
        }

        public void AssignModel(SQLiteDataReader reader)
        {
            // function that assigns values from a SQLiteDataReader to the eventObj property
            try
            {
                // Create a new EventModel and populate its properties with data from the
reader
                eventObj = new EventModel
                {
                    eventID = Convert.ToInt32(reader["eventID"]),
                    eventName = Convert.ToString(reader["eventName"]),
                    eventDate = Convert.ToDateTime(reader["eventDate"]),
                    eventCapacity = Convert.ToInt32(reader["eventCapacity"]),
                    eventTicketPrice = Convert.ToInt32(reader["eventTicketPrice"]),
                    eventCode = Convert.ToString(reader["eventCode"])
                };
            }
            catch (Exception ex)
            {
                // Display an error message if there's an issue processing the data
                MessageBox.Show($"Chyba při zpracovávání dat prodejce: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public int GetFreeCapacity()
        {
            // function that retrieves the free capacity of the associated event
            SQLiteDataReader reader = dbController.GetModelReader("Ticket", "eventID",
eventObj.eventID.ToString());

            if (reader == null)
            {
                // If there are no tickets for the event, return the full capacity
                return eventObj.eventCapacity;
            }

            int eventTickets = 1; // Start at 1 to account for the current ticket
            while (reader.Read())
            {
                eventTickets++;
            }

            // Return the remaining capacity after considering the sold tickets
            return eventObj.eventCapacity - eventTickets;
        }

        public void AssignWithID(int id)
        {
            // funtion that assigns event details based on the provided event ID
```

```
            // Retrieve event data from the database using the provided event ID
            SQLiteDataReader reader = dbController.GetModelReader("Event", "eventID",
id.ToString());
            // Assign the retrieved data to the eventObj property
            AssignModel(reader);
        }
    }
}
```

PrintController.cs

```
using CodeScanner.Model;
using System;
using System.Collections.Generic;
using System.Drawing.Printing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using PdfSharp.Drawing;
using PdfSharp.Pdf;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
using ZXing;


namespace CodeScanner.Controller
{
    internal class PrintController
    {
        XFont fontBold = new XFont("Courier New", 16, XFontStyle.Bold);
        XFont fontNormal = new XFont("Courier New", 14);

        public void GenerateTicketPDF(string ticketPath, string ticketCode, string eventName,
DateTime eventDate, float ticketPrice)
        {
            // function for generating PDF files of tickets

            // Document setup
            int xOffset = 10, yOffset = 25, lineHeight = 20;
            PdfDocument document = new PdfDocument();
            PdfPage page = document.AddPage();
            page.Height = XUnit.FromInch(5.63);
            page.Width = XUnit.FromInch(1.97);
            page.Orientation = PdfSharp.PageOrientation.Landscape;
            XGraphics gfx = XGraphics.FromPdfPage(page);

            // Add event details to the document
            gfx.DrawString($"Collosus -> {eventName}", fontBold, XBrushes.Black, xOffset,
yOffset);
            yOffset += lineHeight * 2;
            gfx.DrawString($"Čas:
{eventDate.Hour.ToString("00")}:{eventDate.Minute.ToString("00")}", fontNormal,
XBrushes.Black, xOffset, yOffset);
            yOffset += lineHeight;
            gfx.DrawString($"Datum: {eventDate.Day}.{eventDate.Month} {eventDate.Year}",
fontNormal, XBrushes.Black, xOffset, yOffset);
```

```
            yOffset += lineHeight;
            gfx.DrawString($"Cena za lístek: {ticketPrice} Kč", fontNormal, XBrushes.Black,
xOffset, yOffset);
            yOffset += lineHeight;

            // Add the generated barcode to the document
            gfx.DrawImage(GenerateBarcode(ticketCode), 200, 40);

            // Save the document
            document.Save(ticketPath);
        }

        static XImage GenerateBarcode(string text)
        {
            // function for generating a XImage object with the ticket code

            BarcodeWriter barcodeWriter = new BarcodeWriter();
            barcodeWriter.Format = BarcodeFormat.CODE_128;
            barcodeWriter.Options = new ZXing.Common.EncodingOptions { Width = 300, Height =
100 };

            // Convert barcode to XImage
            Bitmap barcodeBitmap = barcodeWriter.Write(text);
            MemoryStream memoryStream = new MemoryStream();
            barcodeBitmap.Save(memoryStream, System.Drawing.Imaging.ImageFormat.Png);
            return XImage.FromStream(memoryStream);
        }
    }
}
```

SaleController.cs

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

namespace CodeScanner.Controller
{
    internal class SaleController
    {
        DatabaseController dbController { get; set; }
        public SaleModel saleObj { get; set; }

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

        public void CreateSale(SellerModel saleSeller)
        {
            // function for creating SaleModel object, assigning it some data and saving it
to memory
            saleObj = new SaleModel();
```

```
            saleObj.sellerID = saleSeller.sellerID;
            saleObj.date = DateTime.Now;
            saleObj.saleID = dbController.Entries("Sale") + 1; // +1 due to indexing
        }
    }
}
```

SellerController.cs

```
using CodeScanner.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 CodeScanner.Controller
{
    internal class SellerController
    {
        public SellerModel sellerObj { get; set; }

        public void AssignModel(SQLiteDataReader reader)
        {
            // function that accepts an SQL reader object as an argument and maps its data to
internal seller object
            try
            {
                sellerObj = new SellerModel
                {
                    sellerID = Convert.ToInt32(reader["sellerID"]),
                    sellerName = Convert.ToString(reader["sellerName"]),
                    sellerSurname = Convert.ToString(reader["sellerSurname"]),
                    sellerCode = Convert.ToString(reader["sellerCode"]),
                };
            }
            catch(Exception ex)
            {
                MessageBox.Show($"Chyba při zpracovávání dat prodejce: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

    }
}
```

TicketController.cs

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

```
namespace CodeScanner.Controller
{
    internal class TicketController
    {
        private DatabaseController dbController { get; set; }
        private PrintController printController { get; set; }

        public float totalTicketPrice = 0;
        public List<TicketModel> tickets = new List<TicketModel>();
        internal Random random = new Random();


        public TicketController(DatabaseController databaseController)
        {
            printController = new PrintController();
            dbController = databaseController;
        }

        public void GenerateTickets(int n, EventModel ticketEvent, SaleModel sale)
        {
            // function for generating tickets and saving them into the memory as models
            for(int i = 0; i < n; i++)
            {
                TicketModel ticket = new TicketModel();
                ticket.eventID = ticketEvent.eventID;
                ticket.saleID = sale.saleID;
                ticket.used = false;
                totalTicketPrice += ticketEvent.eventTicketPrice;
                ticket.ticketID = GetTicketID(ticket);
                ticket.ticketCode = GetTicketCode(ticket.ticketID);
                tickets.Add(ticket);
            }
        }

        private string GetTicketCode(int ticketID)
        {
            // function fore generating a unique ticket code

            string code = (random.Next(100_000, 999_999) | ticketID).ToString(); // generate
a random number from <100_000, 999_999> and OR it with the ticketID
            string final = "";

            foreach(char c in code)
            {
                final += ((int)c).ToString(); // add a ASCII value of each digit in the code
int into the final string
            }

            return final;
        }

        public void SaveTicketPdfs()
        {
            // function that iterates through all of the tickets in memory and saves them as
PDF files
```

```
            for(int i=0; i< tickets.Count; i++)
            {
                TicketModel ticket = tickets[i];
                EventModel ticketEvent = dbController.GetEventFromTicket(ticket);

                if(ticketEvent != null)
                {
                    printController.GenerateTicketPDF($"{ticketEvent.eventName}_{ticket.ticke
tID}.pdf",ticket.ticketCode, ticketEvent.eventName, ticketEvent.eventDate,
ticketEvent.eventTicketPrice);
                }
            }
        }

        private int GetTicketID(TicketModel ticket)
        {
            // function for getting the next possible unique ID

            int maxID = -1;
            foreach(TicketModel loopTicket in tickets)
            {
                // query the current list of tickets
                if(loopTicket.eventID == ticket.eventID)
                {
                    if(loopTicket.ticketID > maxID)
                    {
                        maxID = loopTicket.ticketID;
                    }
                }
            }

            if(maxID == -1)
            {
                // no tickets in memory -> get the ID from database
                return dbController.GetMaxTicketID(ticket.eventID) + 1;
            }

            // return the next USABLE ID -> that's why there is a + 1
            return maxID + 1;
        }
    }
}
```

Odpovědi na otázky:

1. V rámci téměř každého systému je nutné zajistit spolehlivou unikátnost nějaké informace. Jak byste řešili tento problém u rozsáhlého systému, pokud byste se nemohli spolehnout na kontrolu existence informace např. dotazem nad DB?

   Problém unikátnosti kódu v okamžiku tvorby se dá řešit pomocí čtení počtu uběhlých milisekund od předem specifikovaného data zkombinovaným společně s polem randomizovaných hodnot. Kombinace by pak probíhala za pomocí procházení pole s uplynulým časem od konce, přičemž v potaz by se bralo pouze posledních 128 bitů. Na hodnotu času se pak pomocí funkce XOR použije bit z randomizovaného pole. Výsledná hodnota se pak zapisuje do finálního pole o velikost 128 bitů.

2. U tohoto typu aplikací se v podstatě vždy jedná o nějaké manipulace s modely reálných entit (osoba, vstupenka). Pro ulehčení této práce slouží technika object-relational mapping. Pokuste se vysvětlit základní principy této technologie a naznačte změnu, kterou by přineslo použití ORM v kódu vaší aplikace.

   ORM je technologie používaná převážně v asociaci s objektově orientovaným programování, přičemž slouží k ulehčení práce s relačními databázemi za pomocí konverze modelů (model = objekt reprezentující tabulku v databázi) a vztahů mezi nimi na klasický databázový systém. Tato technologie tak ve zkratce slouží jako imaginární databázový expert, který pouze konvertuje kód psaný back-end vývojářem na relační databázový systém.

   Základní principy[1] jsou:

   - Mapování objektů na tabulky
   - Podpora CRUD operací
   - Možnost získávání dat z databáze za pomocí dotazů
   - Podpora asociací (relací mezi tabulkami)

   Dopadem použití této technologie pro mé konkrétní řešení by byla výrazná redukce kódu, co se databázového přístupu týče. V případě migrace na tuto technologii by stačilo smazat celý soubor pro práci s databází, nainstalovat knihovnu pro ORM, jako například Entity, a po úpravě kódu jen vytvořit prvotní migraci dat.

   Celková čitelnost kódu by se pak po implementaci ORM výrazně zlepšila a nebylo by třeba mít takové množství pomocných funkcí.

3. Do své dokumentace v použitelné podobě vygenerujte kód, který bude obsahovat vaše jméno, email a datum hodnocení úlohy.

Závěr:

Výsledkem řešení této úlohy je plně funkční vstupenkový systém využitelný v jakémkoliv prostředí, kde je třeba prodávat a následně zpracovávat vstupenky. Jedinou úpravou, která by celý systém pozdvihla na vyšší úroveň by byla optimalizace uživatelské přívětivosti. Na pár místech v programu se mohou úkony vyžadované po obsluze zdát trochu neintuitivní, a proto by bylo opatření formulářů vysvětlivkami a úpravou rozfázovanosti programu vyústilo ve skoro perfektním řešení.

[1] https://www.tutorchase.com/answers/a-level/computer-science/what-are-the-basic-principles-of-object-relational-mapping--orm-