Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Aplikace pro emulaci docházkového terminálu

Bc. Petr Němec

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Petr Němec

Informatika

Název práce

Aplikace pro emulaci docházkového terminálu

Název anglicky

Application of the job attendance terminal

Cíle práce

Cílem diplomové práce je navrhnout a implementovat softwarovou aplikaci pro evidenci docházky, rozšířenou o možnost zaznamenání výkonu práce na konkrétní zakázce. Aplikace bude navržena podle potřeb zákazníka a pořízené záznamy musí být předávány docházkovému systému firmy Ivar a.s.

Metodika

Diplomová práce se bude skládat ze dvou částí, teoretické a praktické.

Metodika zpracování teoretické části práce se bude věnovat studiu a analýze odborných informačních zdrojů použitých v praktické části práce. Praktická část bude zaměřena na analýzu požadavků zákazníka, návrh a implementaci. Aplikace bude vyvíjena v jazyce Delphi v prostředí Delphi 10.4 Sydney. V závěru práce bude aplikace otestována včetně navržení dalších možností vývoje.

Doporučený rozsah práce

60-80 stran

Klíčová slova

docházkový terminál; Delphi 10.4. Sydney; softwarová emulace

Doporučené zdroje informací

Arlow, Jim. a Neustadt, Ila. UML 2 a unifikovaný proces vývoje aplikací. Brno: Computer Press, a.s., 2011. ISBN 978-80-251-1503-9

CANTU, Marco. Myslíme v jazyku Delphi 7. 1. vyd. Praha: Grada, 2003. ISBN 80-247-0694-6

Fialová, Eva. Bezkontaktní čipy a ochrana soukromí. Praha: Legos, 2017. ISBN 978-80-7502-150-2

WIEGERS, Karl Eugene. Požadavky na software. Brno: Computer Press, 2008. ISBN 978-80-251-1877-1

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 4. 11. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 12. 03. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Aplikace pro emulaci docházkového terminálu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.11.2023 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Vojtěchu Merunkovi, Ph.D. za podnětné připomínky, rady a ochotu při vedení této diplomové práce.

Aplikace pro emulaci docházkového terminálu

Abstrakt

Tato diplomová práce se zaměřuje na vývoj a implementaci aplikace pro emulaci docházkového terminálu s rozšířenou funkcionalitou. Hlavním cílem bylo navrhnout a vytvořit uživatelsky přívětivou aplikaci, pro evidenci docházky rozšířenou o možnost zaznamenání výkonu práce na konkrétní zakázce. Práce je rozdělena do teoretické a praktické části.

Teoretická část se věnuje studiu metod softwarového inženýrství, analýze uživatelských požadavků, návrh a modelování aplikace v *Unified Modeling Language*. Dále se zabývá programování v jazyce Delphi, identifikací pomocí RFID technologie a dalšími klíčovými aspekty vývoje aplikace.

Praktická část začíná sběrem a analýzou požadavků, na základě, kterých je aplikace modelována pomocí případů užití, diagramů tříd a stavových diagramů. Následně jsou vytvořeny *wireframy*, které slouží jako vizuální podklad pro samotnou implementaci.

Implementace probíhá v prostředí Delphi 10.4 Sydney. Po dokončení návrhu a implementace následuje fáze testování, během které jsou ověřeny klíčové funkce aplikace. V závěrečné části práce jsou prezentovány výsledky testování a diskutovány možnosti dalšího vylepšení aplikace.

Klíčová slova: Docházka; analýza požadavků; UML; RFID; Delphi; wireframe; Ivar; průmyslový počítač; čtečka; softwarová emulace.

Application of the job attendance terminal

Abstract

This diploma thesis focuses on the development and implementation of an application for emulating a time and attendance terminal with extended functionality. The main objective was to design and create a user-friendly application for attendance tracking, expanded to include recording work performance on specific orders. The thesis is divided into theoretical and practical parts.

The theoretical part addresses the study of software engineering methods, user requirements analysis, design, and application modeling using Unified Modeling Language (UML). It also covers programming in Delphi, identification using RFID technology, and other key aspects of application development.

The practical part begins with the collection and analysis of requirements, based on which the application is modeled using use cases, class diagrams, and state diagrams. Subsequently, wireframes are created, serving as visual aids for the actual implementation.

The implementation takes place in the Delphi 10.4 Sydney environment. After completing the design and implementation, the testing phase follows, during which key functions of the application are verified. In the final part of the thesis, the test results are presented, and possibilities for further improvement of the application are discussed.

Keywords: Attendance; requirements analysis; UML; RFID; Delphi; wireframes; Ivar; industrial computer; reader; software emulation.

Obsah

1 Úvod

Software pro správu docházky hraje v dnešním prostředí klíčovou roli, neboť jeho komplexnost a efektivnost mohou zásadním způsobem ovlivnit chod organizace. Vývoj této aplikace se zaměřuje na simulaci chování fyzického docházkového terminálu a umožňuje efektivní evidenci přítomnosti zaměstnanců v pracovním prostředí.

Během desetiletého působení v oboru softwarového vývoje jsem se věnoval náročným projektům v oblasti docházkových systémů, přístupových řešení, pokladních a skříňkových systémů. Dále se firma specializuje na tvorbu aplikací pro plánování směn v nepřetržitých provozech, zejména pro zdravotnický personál.

Inspirací pro tuto práci byla potřeba zlepšit komunikaci mezi klienty a dodavateli softwarových řešení, zejména v případech, kdy projekty vyžadují vývoj nových softwarových produktů nebo úpravy stávajících systémů. Náročnost a rozsah těchto projektů často vyvolávají nedorozumění, která vedou k nárůstu nákladů a zpoždění dodání aplikací.

Cílem této práce je vytvoření aplikace, která využije softwarového inženýrství k eliminaci nedorozumění mezi zainteresovanými stranami a zefektivní vývoj aplikací pro správu docházky. Aplikace byla navržena tak, aby nahradila stávající docházkové terminály a přinesla nové možnosti pro evidenci pracovní doby na konkrétních zakázkách.

Diplomová práce reaguje na specifický požadavek klienta, jenž si přeje docházkový terminál s rozšířenou funkcionalitou pro evidenci odpracovaných hodin v rámci konkrétní zakázky. Vzhledem k tomu, že žádný z dostupných terminálů nevyhovuje těmto specifickým požadavkům, je nezbytné vyvinout vlastní aplikaci pro emulaci docházkového terminálu. Tato aplikace bude provozována na vybraných zařízeních a nahradí standardní terminál.

Tato práce představuje snahu o inovaci a zvýšení kvality softwarového vývoje v oblasti správy docházky a plánování pracovní doby, a může sloužit jako inspirace pro budoucí

projekty a aplikace, které mají potenciál zlepšit řízení zaměstnanců a pracovních procesů v organizacích různého typu a velikosti.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem diplomové práce je navrhnout a implementovat aplikaci pro evidenci docházky, rozšířenou o možnost zaznamenání výkonu práce na konkrétní zakázce.
Aplikace bude navržena podle potřeb zákazníka a pořízené záznamy musí být předávány docházkovému systému firmy Ivar a.s.

2.2 Metodika

Diplomová práce se skládá ze dvou částí, teoretické a praktické. Metodika zpracování teoretické části práce bude spočívat v důkladném studiu a analýze odborných informačních zdrojů. V rámci této části budou rozebírány klíčové metody softwarového inženýrství, které zahrnují analýzu uživatelských požadavků, použití UML pro návrh systému, prototypování, formulování požadavků na vývoj softwaru, a v neposlední řadě identifikaci pomocí RFID technologie.

Praktická část diplomové práce bude zaměřena na analýzu požadavků zákazníka, návrh a implementaci aplikace v jazyce Delphi v prostředí Delphi 10.4 Sydney. Po implementaci bude aplikace testována a zhodnocena včetně návrhů pro další možnosti vývoje.

3 Teoretická východiska

3.1 Metodiky vývoje software

Současný svět si již nelze představit bez software a počítačů. Aniž bychom si to možná uvědomovali, počítače jsou všude kolem nás. Veřejná infrastruktura, rozvodné sítě, průmyslová výroba, distribuce, finanční systémy a mnoho dalších je řízeno počítačem s patřičným softwarem. Existuje mnoho typů softwarových systémů od jednoduchých až po komplexní, ale stejné by měly mít to, že jejich vývoj podléhá pravidlům, která vedou ke spokojenosti koncového uživatele.

Podle (Sommerville, 2013) profesionálně vytvořený software neobsahuje pouze samotný program, ale také dokumentaci a případně konfigurační data ke správnému fungování. Obvykle při vzniku kompletního softwaru není zásluha jednotlivce, ale celého týmu, který je řízen metodikou vývoje, definované softwarovým inženýrstvím.

Softwarové inženýrství má podporovat profesionální vývoj softwaru a podle knihy (Sommerville, 2013) Softwarové inženýrství je definováno takto:

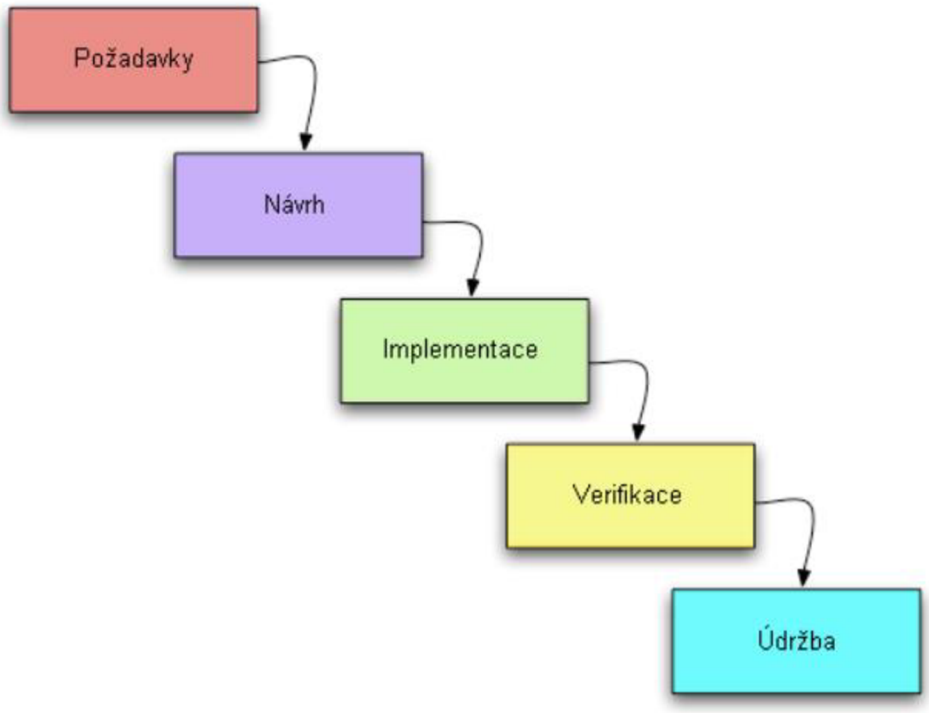
- *„Softwarové inženýrství je technická disciplína, která se zabývá všemi aspekty produkce software od počátečních fází specifikace systému až po údržbu sytému, který se již používá“*
- *„Systematický přístup, který se uplatňuje v softwarovém inženýrství, se někdy označuje jako softwarový proces. Softwarový proces je sekvence aktivit, které vedou k produkci softwarového produktu. Pro všechny softwarové procesy jsou společné čtyři základní aktivity*
    1. *Specifikace software*
    2. *Vývoj software*
    3. *Validace software*
    4. *Evoluce software“*

Existuje mnoho druhů procesních modelů pro splnění různých požadavků. Označujeme je jako modely SDLC (modely životního cyklu vývoje softwaru). Nejoblíbenější a

nejdůležitější SDLC modely uvedené (Sommerville, 2013) jsou následující: vodopádový model, inkrementální (přírůstkový) model, agilní model, spirálový model.
Vzhledem k náročnosti a přiděleným zdrojům dodavatele na vývoj aplikace pro emulaci docházkového terminálu, byl tento projekt koncipován podle vodopádového modelu, který jako jediný z SDLC modelů je popsán v následující kapitole.

3.1.1 Vodopádový model

Vodopádový model byl prvním procesním modelem, který byl představen v roce 1970 Wintson W. Roycem a postupuje formou kaskády jednotlivých fází, zobrazených na obrázku 1. Je také označován jako lineárně sekvenční model životního cyklu. V modelu vodopádu, musí být každá fáze dokončena, než může začít další fáze. Jednotlivé fáze se nepřekrývají.



Obrázek 1 - Vodopádový model (Suchanová, 2021)

Hlavní fáze vodopádového modelu:

1. **Shromažďování a analýza požadavků:** Tato fáze zachycuje všechny možné potřeby systému prokonzultované se zákazníkem, a následně jsou zaneseny do dokumentace a slouží jako specifikace požadavků. Zákazník a vývojář společně pracují na dokumentaci, která obsahuje veškeré požadované funkce a výkony softwaru.

2. **Návrh systému:** Toto je důležitá fáze v modelu. Provádí se analýza dat z první fáze, aby byla použita v následujících fázích kódování. Pomáhá také v celkové architektuře návrhu systému tím, že poskytuje jasný obraz o potřebách hardwaru a softwaru.

3. **Implementace a testování jednotek:** Během této fáze je realizován návrh systému jako sada malých programů neboli jednotek, používané v následných testovacích fázích. Testování jednotek je technika, která ověřuje, jestli je splněna příslušná specifikace.

4. **Integrace a testování systému:** Jednotlivé sady, kousky malých programů jsou sloučeny a testovány jako kompletní systém. Po provedených testech je software předán zákazníkovi.

5. **Provoz a údržba:** Základním aspektem vývojového cyklu produktu je poskytování podpory klientům prostřednictvím údržby a pravidelných kontrol. Údržba zajišťuje opravu chyb, které nebyly odhaleny testováním a také následné zdokonalování software.

Vodopádový model, o kterém je zmiňováno v článku (Suchanová, 2021) Vývoj software je vhodné používat v situacích, kdy jsou požadavky jasně srozumitelné, bez předpokladu, že by se v průběhu vývoje některak měnily. Zpravidla tak bývá u menších projektů. Výhodou tohoto modelu je, že na začátku vývoje se určí přesné požadavky, dle kterých se sestaví harmonogram prací, termíny a rozpočet. Naopak tento model

není tak pružný ke změnovým požadavkům, protože již od začátku je stanoven přesný harmonogram prací.

3.2 Analýza požadavků na vývoj SW

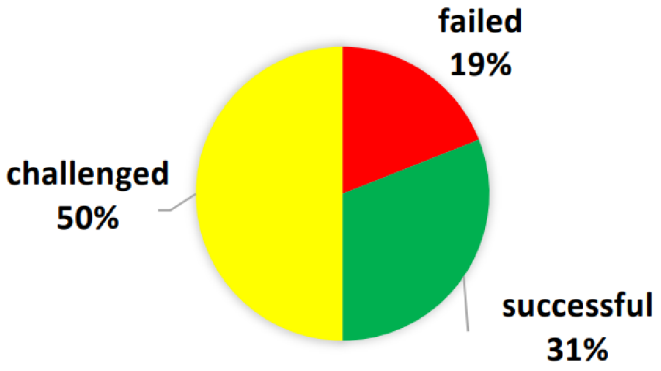
Cílem každého projektu by mělo být dosažení výsledku v požadované kvalitě v dodrženém časovém plánu a rozpočtu. Specifikace a analýza je první fáze vývoje software. Jedná se o pečlivé shromáždění požadavků na projekt od koncových uživatelů nebo zadavatele. Zahrnuje častou komunikaci se zainteresovanými stranami a koncovými uživateli produktu za účelem definování očekávání.

Analýza požadavků tedy zahrnuje týmové úsilí všech klíčových zúčastněných stran: vývojářů software, manažerů, koncových uživatelů, aby bylo dosaženou vzájemného porozumění, co by měl daný produkt dělat. To se vždy provádí v rané fázi projektu, aby bylo zajištěno, že koncový produkt vyhovuje všem požadavkům a předešlo se do budoucna zbytečného předělání projektu z důvodu nepochopení.
Pro požadavky nelze specifikovat přesnou definici, ale pro potřeby této diplomové práce využijeme definování podle (Wiegers, 2008), která požadavek definuje takto:

- *„Podmínku nebo funkci, kterou uživatel potřebuje pro řešení problému nebo dosažení nějakého cíle“*
- *„Podmínku nebo funkci, kterou musí systém nebo jeho část splňovat, aby vyhověl smlouvě, standardu, specifikaci nebo jinému dokumentu, jenž se na něj formálně vztahuje“*
- *„Dokumentovanou podobu některého z předchozích dvou bodů“*

Nedostatečným zpracováním analýzy se zvyšuje riziko dodatečných úprav nebo dodělávek funkcí, které mohou být velmi nákladné jak na peníze i čas. Nastane-li situace, že na konci vývoje byla opomenuta některá funkcionalita z důvodu neuvedení v požadavcích, nejedná se o chybu zhotovitele, ale zákazníka, za kterou bude muset opět zaplatit. Může také nastat situace, že již vývojový tým má přiřazen jiný projekt a původní projekt bude nabírat dlouhé zpoždění, nežli bude dokončen. (Sommerville, 2013)

Společnost *Standish Group*, která se již od roku 1994 zabývá studií úspěchů projektů, které každé dva roky publikuje. Poslední studie společnosti (The Standish Group, 2020) byla vydána v roce 2020 pod názvem *Standish Group 2020 Chaos: Beyond Infinity*. V této studii bylo analyzováno více než 50 000 projektů, ze které vyplynulo, že pouze 31% bylo úspěšných (dodáno v termínu, v rámci rozpočtu s požadovanými vlastnostmi a funkcemi). Druhou nejpočetnější skupinou 50% byly projekty, které byly buď dodány po termínu, přesáhly rozpočet nebo neobsahovaly požadované funkce. Poslední hodnota 19% patří projektům, které se nepodařilo dokončit nebo nebyly nikdy použity.



Graf 1 - Shrnutí zprávy Standish Group CHAOS za rok 2020 (Olivera, 2022)

Závěrem každé analýzy je vytvořit odsouhlasený dokument požadavků, který je součástí smlouvy. Zejména u větších projektů se klade důraz na tvorbu dokumentu „Specifikace požadavků“, které je definován podle normy IEEE 830-1998. Na obrázku 2 je zobrazen průběh vývoje požadavků. První v procesu je sběr dat na systém od potenciálního zákazníka. Po sběru dat následuje jejich analýza. V případě nepochopení probíhá se zákazníkem vyjasnění požadavků, eventuelně může obsahovat modely a prototypy systému pro lepší porozumění. V aktivitě specifikace požadavků, dochází k sepsání analýzy do dokumentu. Nakonec probíhá kontrola realističnosti a úplnosti požadavků. (Wiegers, 2008)

Obrázek 2 - Vývoj požadavků (Szturcová, 2013)

3.2.1 Funkční a mimofunkční požadavky

U požadavků je kladen velký důraz na pečlivost, protože to je prvotní část pro úspěšný projekt. Systémové požadavky lze kategorizovat jako funkční a mimofunkční požadavky.

Funkční požadavky lze definovat „co“ musí systém udělat. Slouží k vyjádření chování systému zadáním vstupních a výstupních podmínek. Specifikují, co musí systém dělat v reakci na různé vstupy a výstupy. Například se může jednat o funkce, výkonnost, výstupy, vstupy, chování a další prvky systému.

Mimofunkční požadavky - popisují systém „jak“ se má chovat. Často se týká systému jako celku, nikoliv jednotlivých funkcí. Mohou se například vztahovat k vlastnostem systému, jako je spolehlivost, estetika, snadné použití, doba odezvy. (Sommerville, 2013)

3.3 UML

Jazyk UML (*Unified Modeling Language*, unifikovaný modelovací jazyk) je univerzální grafický jazyk pro modelování systému, který se využívá při vývoji softwaru. Jedná se o standardizovaný způsob, jak definovat celou softwarovou architekturu. Neslouží pouze jako podpůrný nástroj pro komunikaci mezi vývojáři a zákazníky, ale také mezi vývojářským týmem (programátory, designéry, analytiky atd.).

Historie UML sahá až ke konci 80. let, kde vznikalo mnoho specifikací a standardů, jak objekty a další součásti aplikace zakreslovat. V roce 1994 se autoři metod Grady Booch, Ivan Jacobson a James Rumbaugh spojili ve firmě Rational Corporation, kde pracovali na tvorbě jazyka UML. V roce 1997 bylo sdružením OMG (Object Management Group) přijat jazyk UML jako standard a od té doby na specifikace UML dohlíží. OMG je otevřená instituce, která je zastoupena firmami jako IBM, Microsoft, Oracle, HP a další. První ISO certifikace pro UML označeno ISO/IEC 19501:2005, byla přijata v roce 2005. (Arlow, 2011) Na obrázku 3 je zobrazen přehled všech přijatých standardů. Posledním přijatým standardem je verze 2.5.1 z prosince 2017.

VERSION	ADOPTION DATE	URL
2.5.1	prosince 2017	https://www.omg.org/spec/UML/2.5.1
2.4.1	července 2011	https://www.omg.org/spec/UML/2.4.1
2.3	května 2010	https://www.omg.org/spec/UML/2.3
2.2	ledna 2009	https://www.omg.org/spec/UML/2.2
2.1.2	října 2007	https://www.omg.org/spec/UML/2.1.2
2.0	července 2005	https://www.omg.org/spec/UML/2.0
1.5	března 2003	https://www.omg.org/spec/UML/1.5
1.4	září 2001	https://www.omg.org/spec/UML/1.4
1.3	února 2000	https://www.omg.org/spec/UML/1.3
1.2	července 1999	https://www.omg.org/spec/UML/1.2
1.1	prosince 1997	https://www.omg.org/spec/UML/1.1

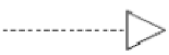
Obrázek 3 – Historie standardů UML (OMG Standarts Development Organization, 2017)

Jak popisuje (Arlow, 2011), využití UML je velice rozsáhlé – od prostředků pro obecný popis systému až po detailní návrh, ze kterého je možné vygenerovat kód v objektově orientovaném jazyce. Jak již bylo zmíněno, používá se převážně pro návrhy objektově orientovaných systémů, ale využití nalézá i v obchodních a podnikatelských procesech. Struktura jazyka je dělena na součásti stavebních bloků, společných mechanizmů a architektury. Společné mechanizmy obsahují způsoby, jak dosáhnou specifických cílů a architektura vyjadřuje perspektivu navrhovaného systému v jazyce UML. Stavební bloky jsou základní prvky modelu a jsou tvořeny předměty, vztahu a diagramy.

Předměty (*things*) jsou samostatné prvky modelu a lze je popsat jako objekt nebo entitu reálného světa. Předměty (věci nebo abstrakce) jsou rozděleny na kategorie:

- **Strukturní abstrakce** - je používána k reprezentaci viditelných prvků, jako jsou, třídy, rozhraní, spolupráce a případy užití.
- **Chování** – slouží k vyjádření fungování systému a zahrnuje slovesa v modelu UML, jako jsou interakce, aktivity a stavové automaty.
- **Seskupení** - balíček, který se používá k seskupení sémanticky souvisejících prvků.
- **Poznámky** - anotace, která může být zapsána do modelu, aby zachytila některé důležité informace.

Relace (relationships) – vyjadřuje vztah vzájemného propojení mezi dvěma předměty.
Na následujícím obrázku 4 je zobrazen přehled relací.

vztah	UML	význam
závislost (dependency)		zdrojový prvek závisí na cílovém a může být ovlivněn změnou jeho stavu
asociace (association)		Popis množiny spojení mezi objekty
agregace (aggregation)		cílový element je součástí zdrojového
kompozice (composition)		silnější forma agregace
generalizace (generalization)		zdrojový prvek je specializace obecnějšího cílového prvku a může jím být zastoupen-nahrazen
realizace (realization)		cílový element realizuje chování, které zdrojový element specifikuje

Obrázek 4 - Typy relací v diagramu tříd UML (Arlow, 2011)

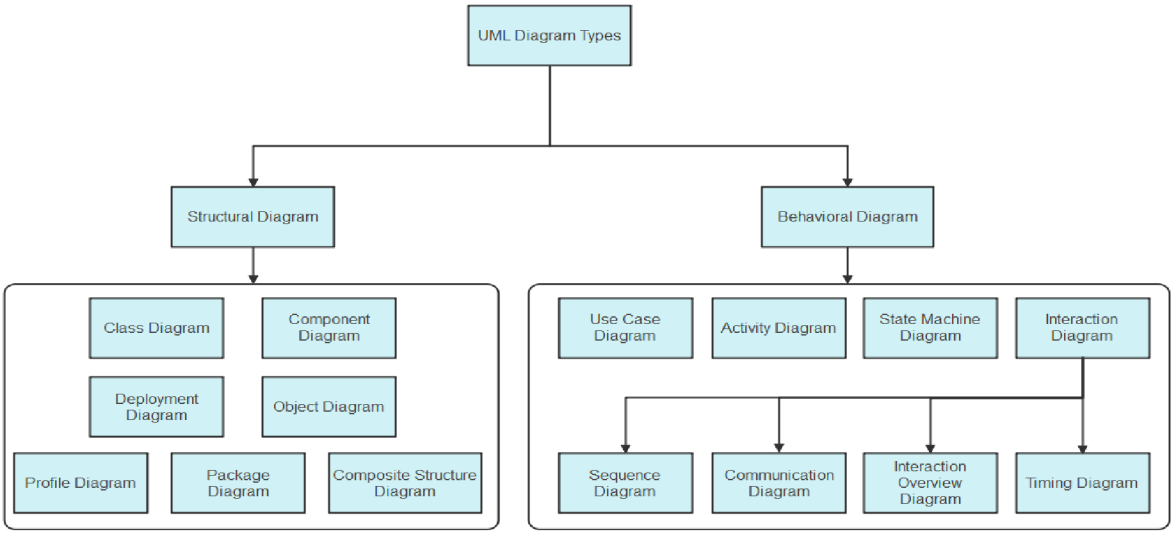
Diagramy (*diagrams*) – pohledy na modely, tedy vizuálními reprezentacemi toho, co a jak systém plánuje dělat. Následující kapitola bude věnována tomuto klíčovému prvku systému.

3.3.1 Diagramy

Za pomoci diagramů umožňuje jazyk UML zachytit systém na různé úrovni abstrakce. Jsou to pohledy na model, které mají vývojářům nebo zákazníkům umožnit porozumět a analyzovat strukturu i chování systému. K popisu systému existuje celkem 14 diagramů, které se dělí do dvou základních skupin. Diagramy struktury a chování jsou zobrazeny na obrázku 5. (Lynch, 2022)

Diagramy struktury znázorňují statický pohled nebo strukturu systému. Jsou velmi používány v dokumentaci softwarové architektury, kde zobrazují různé objekty a jejich vztahy. Zahrnují: Diagram tříd, Diagram komponent, Diagram nasazení, Objektový diagram, Diagram profilů, Diagram balíčků a Diagram složené struktury.

Diagramy chování zobrazují dynamické chování sytému. Popisuje fungování systému. Zahrnuje diagramy případu užití, diagram aktivit, diagram stavového automatu, diagram interakce, který se skládá z diagramu posloupnosti, diagram komunikace, stručný diagram interakce a diagram časování. (Arlow, 2011)



Obrázek 5 - Typy diagramů UML (Lynch, 2022)

V následující části budou popsány tři diagramy, které budou využity pro návrh aplikace emulace docházkového terminálu.

3.3.1.1 Diagram případů užití

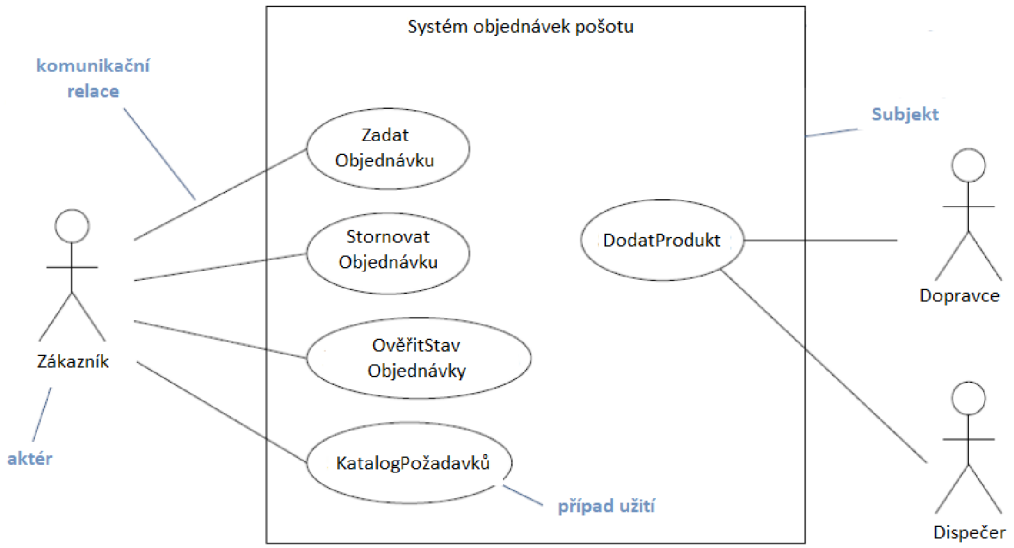
Diagram případů užití (*use case diagram*), patří mezi diagramy chování. Používá se k zobrazení funkcí systému z pohledu uživatele (v UML nazývaný „*actor*“). Tento aktér nemusí být nutně člověkem. Roli lze také přiřadit externímu systému, který přistupuje k jinému systému. Diagram případu užití zobrazuje spojení mezi aktérem a jeho požadavky nebo očekáváními systému, aniž by popisoval probíhající akce nebo je dával do logického pořadí.

Využívá se k jasné prezentaci nejdůležitějších funkcí systému. Při vývoji softwaru nebo plánování nových obchodních procesů je často jedním z prvních kroků vytvoření diagramu případů použití. Poskytuje jednoduchou a jasnou vizualizaci toho, které případy použití je třeba při vývoji zohlednit.

K zobrazení diagramu případu použití se používají standardizované stavební bloky, mezi které patří:

- **Aktér**: Lidé i systémy, reprezentováni jako panáčci vně subjektu.
- **Subjekt**: Systém označený rámečkem, který určuje hranice subjektu
- **Případ užití**: Zobrazen jako elipsa, obvykle obsahuje krátké vyjádření, které pojmenovává proces.

Vztahy (komunikační relace) mezi těmito bloky jsou zobrazeny spojovacími čarami, tzv. asociacemi. Plná čára reprezentuje přiřazení mezi aktérem a případem užití.
Jednotlivý aktéři jsou zobrazovány mimo subjekt. Jednotlivé případy užití, které specifikují chování systému, jsou umístěny uvnitř subjektu. Propojení aktéra s případem užití je modelováno pomocí komunikační relace viz obrázek 7. V některých situacích, je potřeba při modelování zakreslit vztah mezi jednotlivý případy užití. Takovýto vztah, zobrazuje závislost mezi dvěma případy užití a je modelován dvěma způsoby: <<include>> nebo <<extend>>. (Arlow, 2011)



Obrázek 6 - Diagran případu užití (Arlow, 2011)

Relace **include** nastavená mezi případy užití, umožňuje využít chování rozšiřujícího případu použití. Tento typ relace je součástí základního případu a nemůže fungovat samostatně. Hlavním důvodem rozšířeného případu užití je opětovné použití společných akcí ve více případech. Vztah je znázorněn směrovou šipkou s přerušovanou čárou, kde šipka směřuje od základního k rozšiřujícímu případu užití. (Kraval, 2016)

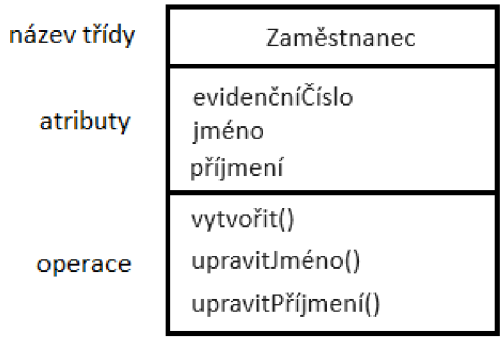
Relace **extend** je vztah mezi případy užití, který rozšiřuje základní případ užití. Základní případ použití, je zcela nezávislý na rozšiřujícím případu. Vztah je znázorněn směrovou šipkou s přerušovanou čárou, která směřuje od rozšiřujícího k základnímu případu užití. (Kraval, 2016)

3.3.1.2 Diagram tříd

Diagram tříd (*class diagram)* popisuje třídy v modelované aplikace spolu s jejich vlastnostmi a vzájemnými vztahy. Poskytuje strukturální a statický pohled na celý systém. Využívá se k vizualizaci, popisu, dokumentaci různých aspektů aplikace. Jako jediný z diagramů je na platformě závislý, tedy charakteristický pro určitý programovací jazyk.

Každá třída v UML je zobrazena v obdélníku, který je rozdělen na tři části. V horní části musí být uveden název třídy, následně atributy, které definují třídu jako objekt a operace

neboli metody, což jsou funkce, které objekt vykonává. Příklad modelu třídy je uveden na obrázku 7. (Arlow, 2011)

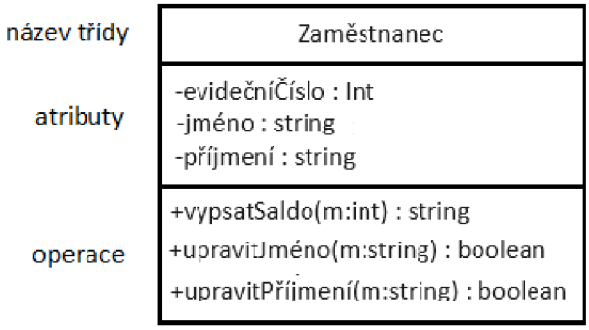


Obrázek 7 – Model třídy (zdroj: vlastní)

Takto navržená třída není ještě kompletní a její využití by bylo možné pouze v analýze. Atributy a operace musí obsahovat datové typy a návratové hodnoty. Datové typy si je možné představit jako faktory viditelnosti a rozlišujeme čtyři základní:

- + (public) – veřejný atribut, přístup z libovolné třídy
- - (private) – přístup pouze ze,stejné třídy, ve které byly deklarovány
- # (protected) – přístup pouze ve třídě a nebo odvozených tříd
- ~ - bez modifikátoru, přístup povolen v rámci stejného balíčku

Příklad třídy o doplněné datové typy a návratové hodnoty uveden na obrázku 10.



Obrázek 8 - Doplněný model tříd (zdroj: vlastní)

K celistvosti diagramu nám nestačí pouze namodelované jednotlivé třídy, ale je potřeba specifikovat vzájemné vztahy mezi sebou. Vztahů mezi třídami, které lze využít je několik. Mezi nezákladnější patří:

Asociace popisuje statické nebo fyzické spojení mezi dvěma nebo více objekty. Zobrazuje, kolik předmětů je ve vztahu. Na diagramu je znázorněna plnou čárou mezi třídami.

Agregace je speciální forma asociace, která specifikuje vztah mezi částí a celkem. Celek může existovat samostatně, nezávisle na části. Modeluje se kosočtverec na straně instance, který zastupuje celek. Na obrázku 9 je uveden příklad agregace. Společnost má několik zaměstnanců a pokud podá jeden ze zaměstnanců výpověď, společnost stále existuje. (Rydval, 2018)

Obrázek 9 - Příklad agragace (Rydval, 2018)

Kompozice je vyjádření silnější vazby agregace. Vyznačuje se plným kosočtvercem na straně celku a zobrazuje závislost mezi rodičem a potomkem. Pokud bude jedna část odstraněna, druhá část postrádá význam.

Obrázek 10 - Příklad kompozice (Rydval, 2018)

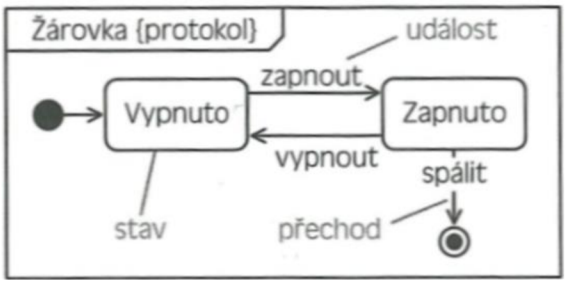
3.3.1.3 State Machine diagram

Při vývoji produktů pomáhají diagramy stavových strojů (State Machine diagram), znázornit životní cyklus jednotlivých objektů vizuálním způsobem. Stavový diagram modeluje chování objektů a stavy do kterých se během života dostávají. Třemi základními prvky jsou stavy, události a přechody.

- Stav představuje situaci během života objektu. Popisuje výkon aktivity nebo čeká na událost.
- Událost popisuje výskyt v čase a prostoru

- Přechod spojuje stavy. Po přepnutí z jednoho stavu do druhého musí být spuštěna akce, která znázorňuje přechod.

Pro znázornění diagramu stavových automatů je na obrázku 11 použitý jednoduchý příklad životního cyklu žárovky.

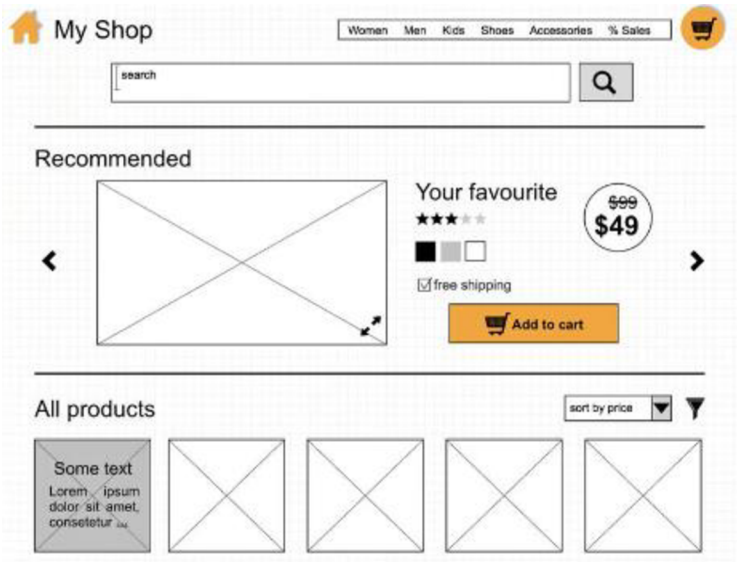


Obrázek 11 - Diagram stavových automatů (Arlow, 2011)

3.4 Wireframe

Samotný *wireframe* (drátěný model) popisuje (Ketterman, 2023) jako základní konstrukční prvk v procesu vývoje uživatelského rozhraní aplikace nebo webové stránky. Jedná se o vizualizaci, která odhaluje dispozici, strukturu a interakce prvků bez využití grafického obsahu nebo estetických prvků. Cílem *wireframu* je jednoznačně ukázat, jaké prvky jsou umístěny a jak spolu interagují na dané stránce nebo v aplikaci.

Během tvorby *wireframů*, je převážně zaměřeno na základní prvky (navigační prvky, tlačítka, textová pole a obrázky) a jejich strukturu včetně umístění na stránce. Jejich hlavní funkcí je reprezentovat funkčnost a hierarchii bez designových detailů. Mají zásadní význam v procesu vývoje aplikací, protože poskytují jasnou vizualizaci struktury a funkcí. Tímto způsobem umožňují lepší komunikaci mezi designéry, vývojáři a zákazníkem. Umožňují identifikovat a řešit problémy již v rané fázi vývoje.



Obrázek 12 – Příklad wireframe (Ketterman, 2023)

3.5 Delphi

Delphi je vývojový nástroj postavený na objektově orientované architektuře, který umožňuje navrhovat a efektivně vytvářet aplikace. Programátor má díky mnoha vizuálním nástrojům a integrovanému prostředí zjednodušenou návrhovou práci aplikace. Prostředí Delphi vychází z jazyka *Object Pascal*, což je objektová nadstavba programovacího jazyka *Turbo Pascal*.
Delphi bylo poprvé vydáno v roce 1995 společností *Borland Software Corporation* jako nástroj pro vytváření aplikací pro *Windows*. Program byl vyvinut z *Turbo Pascalu*, vydané v listopadu 1983 stejnou společností Borland. Hlavní architekt Turbo Pascalu a následně Delphi byl Anders Hejlsberg. (Embarcadero.com, 2020)

Původ názvu „Delphi“ navrhl v polovině roku 1993 hlavní vývojář v *Borland Software Technologies* Danny Thrope. Původně se jednalo pouze o kódové označení pro beta verzi. Vývoj produktu probíhal pod přísně střeženým dohledem a často se název měnil jako prevence před případným odposlechem konkurence. Název nesměl mít žádnou spojitost s produktem, aby nebylo příliš zřejmé, o jakém produktu se diskutuje. Před samotným vznikem názvu „Delphi“, po dlouhodobém vývoji napojení na relační databáze Oracle, bylo vývojového a výzkumného manažera Garyho Whizina přání, aby v názvu bylo ukotveno slovo Oracle. V překladu slova Oracle znamená věštec a Danny Thrope vymyslel slogan „*If you want to talk to [the] Oracle, go to Delphi*“ (Pokud chcete mluvit s věštcem, jděte do Delphi) odkud následně vzešel jednoslovný název pro vývojový nástroj. Delphi pochází z řecké mytologie, jednalo se o staré místo, chrám v troskách mrtvé civilizace, kde věštci (Oráclové) za úplatu předpovídali budoucnost.

Finální produkt měl být uveden pod názvem *AppBuilder*, ale několik měsíců před vydáním softwarová firma Novell vydala produkt s názvem *Visual AppBuilder* a tak se zůstalo u kódového názvu „Delphi“.(Thorpe, 2020) Od roku 2008 je Delphi vlastněno a dále vyvíjeno firmou *Embarcadero Technologies*. Díky své platformě FireMonkey je možno aplikace vyvíjet nejen pro Windows, ale i macOS, Android, iOS a Linux. V době psaní této diplomové práce je nejaktuálnější verze Delphi 11.2 vydána v září 2022.

3.5.1 Popularita Delphi

V současnosti sice Delphi nevládne celosvětovému žebříčku popularity, ale stále se jedná, o podporovaný nástroj, se kterým lze vyvíjet všechny druhy aplikací, jako jsou například účetní software, antivirové nástroje, skripty, aplikace strojového učení, technologie virtuálního studia a další.

Mezi nejoblíbenější indikátory oblíbenosti programovacích jazyků, které se provádí pomocí tzv. měření popularity, patří dva nejzákladnější TIOBE a PYPL.

Index popularity TIOBE

Výpočet indexu TIOBE spočívá v počítání zadaných dotazů o konkrétním programovacím jazyky ve vyhledávačích. K výpočtu je použito 25 nejpopulárnějších vyhledávačů (Google, Bing, Amazon, YouTube, atd). Na obrázku 13 jsou uvedeny výsledky hodnocení z listopadu 2022 a Delphi se podle měření Tiobe nachází na 12 místě. (Tiobe.com, 2022)

Nov 2022	Nov 2021	Change	Programming Language	Ratings	Change
1	1		Python	17.18%	+5.41%
2	2		C	15.08%	+4.35%
3	3		Java	11.98%	+1.26%
4	4		C++	10.75%	+2.46%
5	5		C#	4.25%	-1.81%
6	6		Visual Basic	4.11%	-1.61%
7	7		JavaScript	2.74%	+0.08%
8	8		Assembly language	2.18%	-0.34%
9	9		SQL	1.82%	-0.30%
10	10		PHP	1.69%	-0.12%
11	18	⇈	Go	1.15%	-0.06%
12	15	↑	R	1.14%	-0.14%
13	11	↓	Classic Visual Basic	1.10%	-0.46%
14	17	↑	Delphi/Object Pascal	1.08%	-0.14%

Obrázek 13 - Indikátor popularity TIOBE (Tiobe.com, 2022)

Index popularity PYPL

Index PYPL (*PopularitY of Programming Language Index*) je vytvořen analýzou toho, jak často jsou dané jazyky vyhledávány na Googlu. Čím více je vyhledáván daný programovací jazyk, tím se považuje za populárnější. Nezměněná data pocházejí ze služby Google Trends. Výsledek měření podle indexu PYPL je zobrazen na obrázku 14 z listopadu 2022. (Pypl.github.io, 2022)

Rank	Change	Language	Share	Trend
1		Python	28.44 %	-1.2 %
2		Java	17.03 %	-0.3 %
3		JavaScript	9.51 %	+0.3 %
4		C#	7.08 %	-0.1 %
5		C/C++	6.51 %	-0.4 %
6		PHP	5.12 %	-1.0 %
7		R	4.12 %	+0.4 %
8	↑↑↑	TypeScript	2.89 %	+1.3 %
9	↓	Objective-C	2.17 %	+0.1 %
10		Swift	2.11 %	+0.4 %
11	↑↑	Go	2.07 %	+0.7 %
12	↓↓↓	Kotlin	1.81 %	+0.1 %
13	↑↑	Rust	1.7 %	+0.6 %
14	↓↓	Matlab	1.58 %	+0.2 %
15	↑	Ruby	1.07 %	+0.1 %
16	↓↓	VBA	0.97 %	-0.2 %
17		Ada	0.81 %	-0.0 %
18	↑↑↑	Dart	0.76 %	+0.3 %
19	↓	Scala	0.73 %	-0.0 %
20	↑↑	Lua	0.64 %	+0.2 %
21	↓↓	Visual Basic	0.62 %	-0.1 %
22	↓↓	Abap	0.48 %	-0.0 %
23		Perl	0.38 %	+0.0 %
24		Groovy	0.37 %	+0.0 %
25	↑	Julia	0.3 %	+0.0 %
26	↑	Haskell	0.29 %	+0.0 %
27	↓↓	Cobol	0.28 %	-0.1 %
28		Delphi	0.12 %	-0.1 %

Obrázek 14 - Indikátor popularity PYPL (Pypl.github.io, 2022)

3.6 Identifikace osob

Nejzákladnější funkcionalitou docházkových terminálů je označení příchodů a odchodů zaměstnanců pro evidenci jejich pracovní doby včetně nepřítomnosti.
K tomu je zapotřebí jednoznačné určení totožnosti zaměstnance. Identifikace osob může probíhat pomocí bezkontaktních RFID a NFC čipů, biometrie, magnetických karet či pomocí tištěných QR kódů nebo osobního hesla. V aplikaci této diplomové práce je využito RFID technologie pro identifikaci osob, která je následně popsána.

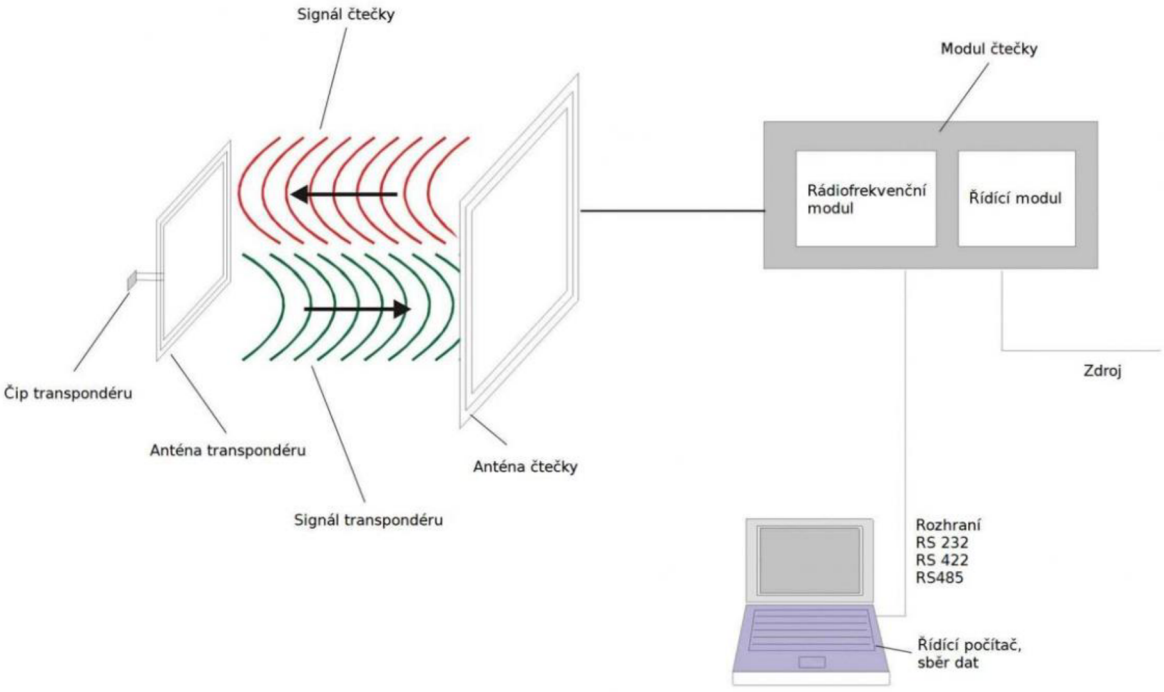
3.6.1 RFID

RFID technologie je běžnou součástí našeho života, kterou využíváme každý den a ani si to nemusíme uvědomovat. Technologie RFID (*Radio Frequency Identification*) je založena na bezkontaktní, rádiové komunikaci k identifikaci objektů na dálku. Nejedná se o novou technologii, prvopočátek lze vysledovat až do druhé světové války, kde pomocí rádiových signálů byly rozeznávány bojová letadla. Nynější využití RFID je značně rozšířené, například v logistice a zásobování slouží k označování palet a výrobků pro rychlejší identifikaci. V obchodě pro ochranu zboží před krádeží, ve zdravotních službách k identifikaci pacienta včetně zaznamenání léčby, identifikaci docházkových a přístupových údajů, identifikace cestujících ve veřejné dopravě, identifikační průkazy vydané státem atd. (Violino, 2005)

3.6.1.1 Princip fungování

Jak již bylo zmíněno výše, RFID technologie využívá rádiové vlny k bezkontaktní identifikaci na různou vzdálenost. Systém se skládá ze dvou komponent: čtečky (*RFID - reader*) a tagu, který je někdy označovaný jako štítek nebo transpondér. Čtečka je zařízení, které má jednu nebo více antén vysílající rádiové vlny a přijímají signály zpět z *RFID-tagu*. Štítky, které využívají rádiové vlny ke sdělování své identity a dalších informací blízkým čtečkám, mohou být pasivní nebo aktivní (Fialová, 2016). Rozdíl mezi aktivním a pasivním tagem je vysvětlen v kapitole 3.5.1.3 *RFID Tag*.

Na obrázku 15, je zobrazen princip komunikace pasivního RFID systému. Čtečka vysílá elektromagnetické pole a v dostatečné vzdálenosti aktivuje transpondér, který vyšle data zpět. Získaná data čtečkou jsou předána do počítače, se kterými pracuje patřičný software. (Cosmotron.cz, 2019)



Obrázek 15 - Princip činnosti pasivního RFID (Cosmotron, 2019)

3.6.1.2 Frekvenční dělení

Systém RFID jak je možné se dočíst v knize (Fialová, 2016) Bezkontaktní čipy a ochrana soukromí, funguje na různých vlnových délkách při použití libovolné frekvence od nízké po velmi vysokou. V současné době existují tři frekvenční oblasti. Pracují na nízké, vysoké nebo hyper vysoké frekvenci. Rádiové vlny se v těchto frekvencích liší a při použití těchto šířek pásma existují výhody i nevýhody.

Například nízkofrekvenční RFID systém má menší kapacitu přenosu dat, ale zvyšuje schopnost čtení v blízkosti kovů nebo kapalin. Pokud systém pracuje na vyšší frekvenci, obecně přenáší data rychleji a na delší detekční vzdálenost, ale rádiové vlny jsou citlivější na rušení způsobené kapalinami a kovy v prostředí.

Nejnovější technologické inovace posledních let však umožňují RFID UHF (ultra-vysokofrekvenční) systémy používat v prostředí s kapalinami a kovy.

1)Nízká frekvence (LF = 125kHz)

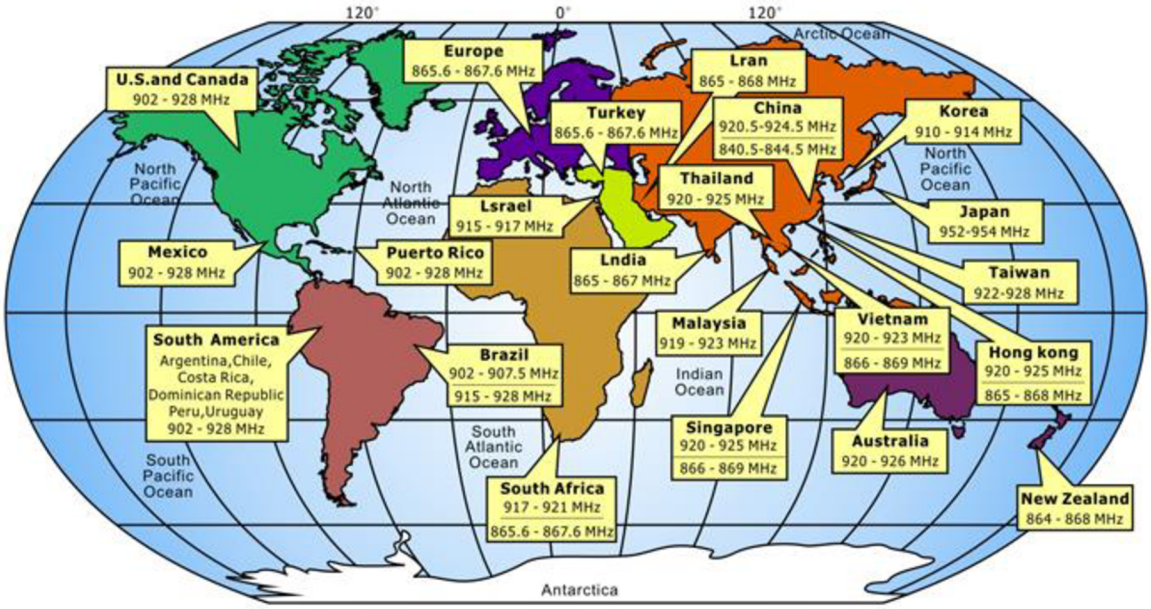
Pásmo LF pokrývá frekvence mezi 30 kHz až 300 khz. Typické LF RFID systémy pracují se 125 KHz nebo 134 Khz. Tato frekvence poskytuje krátký dosah čtení, asi 10 cm. Výhodou je, že transpondér RFID v tomto frekvenčním pásmu je relativně odolný, pokud jde o vliv kovů či kapalin. Tato technologie je převážně využívána pro přístupové systémy, elektrické zámky, pohony vrat, docházkové systémy, zařízení pro zadávání hesla, bezpečnostní systémy i registrace zvířat.

2) Vysoká frekvence (HF 13,56 MHz)

Rozsahy HF frekvencí jdou od 3 do 30 MHz. Většina RFID HF systémů pracuje s frekvencí 13,56 MHz, s dosahem čtení 10 cm. Často používána pro bezhotovostní platební systémy, např. v dopravě (autobusové jízdenky, jízdenky na vlak, elektronická peněženka atd.) Protože tato technologie používá sofistikované bezpečnostní algoritmy a čipy mají integrovanou paměť, jsou velmi užitečné pro bezhotovostní platební aplikace, jako jsou debetní nebo kreditní karty.

3) Ultra vysokofrekvenční (UHF)

(868MHz pro Evropu nebo 915MHz pro Severní Ameriku) je nejmladší v oblasti RFID technologií a perspektivní v průmyslových aplikacích, jako je řízení skladových zásob či sledování zásilek. Vzhledem k tomu, že čtecí vzdálenost může dosáhnout až několika metrů (4 - 5), umístění tagu je mnohem flexibilnější v porovnání s LF nebo HF technologií.



Obrázek 16 - Kmitočtová pásma pro UHF RFID systémy ve světě (Lightpioneer.com, 2017)

3.6.1.3 RFID Tag

RFID-tag je nosič dat, který uchovává řadu informací, od jednoho sériového čísla až po několik stránek. Je složen z čipu, antény a obalu. Právě na čipu jsou data fyzicky uložena a dělí se do tří kategorií podle využití:

a) **Read-Only (ROM)** - data zapsaná výrobcem. Zpravidla se jedná pouze o sériové číslo a čip slouží pouze ke čtení.
b) **Write-Once(WORM)** - umožněn prvotní zápis, následně slouží pouze pro čtení.
c) **Read/Write (RAM)** - mnohonásobné přepisování dat

Anténa je největší částí tagu. Přijímá a odesílá rádiové signály. RFID-tagy neboli transpondéry, se dělí na dva základní typy – aktivní a pasivní. Aktivní transpondéry disponují vlastním systémem pro zásobování energie, mají např. integrovanou baterii a mohou přenášet data na větší vzdálenost (až 100 m).

Pasivní transpondéry RFID získávají energii z vysílací čtečky, která vytváří prostřednictvím anténové cívky elektromagnetické pole, které je zachyceno transpondérem. Pomocí elektromagnetického pole je posléze do transpondérové antény indukován proud, který dodává energii, pomocí níž je transpondér poháněn. Čtecí vzdálenost nepřesahuje pět metrů.

Obal tagu slouží k ochraně čipu a antény a je uzpůsoben příslušnému použití. Nejjednodušší formu obalu představuje štítek RFID. Tato metoda je pro mnoho aplikací dostatečná. Jedná se však o formu obalu, kdy je čip chráněn pouze jednoduchou fólií, nebo vrstvou papíru. Pakliže je zapotřebí stabilnější a odolnější řešení, je čip a anténa zatavena do laminované karty. Pokud ani to nestačí, využívá se plastový kryt, který představuje nejodolnější, nejtrvanlivější a nejméně náchylný typ obalu. (Fialová, 2016)

3.7 Databáze

Databáze je sofistikovaný softwarový systém, určený k ukládání dat a jejich následnému zpracování. Jedná se o digitální úložiště, které umožňuje data ukládat, spravovat a následně vyhledávat informace. Databáze je uložena jako soubor nebo sada souborů.
Informace v těchto souborech může být rozdělena na záznamy, z nichž každý se skládá z jednoho nebo více polí.

Počátkem 60. let 20. století lze považovat za milník ve světě databází. Byly vyvinuty dva hlavní datové modely - síťový model vedený Charlesem Bachmanem známá jako IDS (*Integrated Data Store*) a hierarchický model IMS (*Information Management System*) vytvořený firmou IBM. Aplikace obvykle přistupovaly k datům sledováním ukazovatelů z jednoho záznamu na druhý.

V červnu roku 1970 Edgar F. Codd, který se v organizaci IBM, primárně zabýval vývojem pevných disků, publikoval dokument s názvem "Relační model dat pro velké sdílené banky". Tento dokument představoval nový způsob modelování dat. Coddovým nápadem bylo uspořádat data do tabulek, přičemž každá tabulka se používá pro jiný typ entity. Každá tabulka obsahuje pevný počet sloupců obsahující atributy a entity. Jeden nebo více sloupců každé tabulky je určeno jako primární klíč, pomocí kterého lze jednoznačně identifikovat řádky tabulky. V 80. a 90. letech se relační databáze staly velice populární a tento stav stále setrvává. Pro práci s daty u relačních databází se stal strukturovaný dotazovací jazyk SQL.

V 90. letech spolu s narůstajícím objektově orientovaného programování vznikají první databáze založené na principech objektově orientovaného programování. Data jsou

uspořádána do objektů, přičemž každý má své vlastní atributy a metody. Tento přístup k řízení databází poskytuje přirozenější a intuitivnější způsob reprezentace dat, protože se podobá skutečným objektům.

V posledních letech se stále více rozšiřují NoSQL databáze (nejen strukturovaný dotazovací jazyk). Odkazuje na databáze, které k ukládání a načítání dat používají jiný dotazovací jazyk než SQL. Tyto databáze se liší od tradičních relačních databází (RDB) tím, že nejsou omezeny pouze na relační model dat a umožňují pracovat s různými typy datových struktur, jako jsou například dokumenty, grafy.
NoSQL databáze umožňují ukládání a manipulaci s nestrukturovanými a polostrukturovanými daty, často vyznačují vyšší škálovatelností, vysokou dostupností a rychlým časem reakce, což je důležité pro aplikace, které musí zpracovat velké množství dat v reálném čase. (Kronenke a Auer, 2015)

3.7.1 Konceptuální model

Konceptuální datový model je uspořádaný pohled na koncepty a vztahy v databázi. Cílem vytvoření koncepčního datového modelu je vytvořit logický a snadno pochopitelný popis dat, který bude sloužit jako základ pro navrhnutí a implementaci skutečné databáze. Konceptuální model neposkytuje téměř žádné detaily o skutečné struktuře databáze, ale poskytuje přehledný a jasný pohled na to, co data v databázi obsahují a jakými vztahy mezi sebou jsou propojena.

Konceptuální model se definuje jako abstraktní model, který uspořádává popis dat, sémantiku dat a omezení konzistence dat. Tyto modely se zaměřují na to, jaká data jsou potřebná a jak by měla být uspořádána. Méně se hledí na to, jaké operace budou s daty prováděny. (Bruckner, 2012)

Tři základní principy konceptuálního datového modelu jsou:

- Entita: cokoliv z reálného světa
- Atribut: vlastnosti entity
- Vztah: závislost mezi entity.

Mezi nejpoužívanější technikami pro konceptuální modelování patří:

- Model vztahu entit (ER)
- UML (Unified Modeling Language)

4 Praktická část práce

V rámci vlastní práce se budu věnovat analýze požadavků, návrhu a implementaci aplikace pro emulaci docházkového terminálu. Cílem této aplikace bude kompletní nahrazení stávajícího docházkového terminálu rozšířené o možnost evidence pracovní doby na zakázce. Pro tento účel bude vytvořena desktopová aplikace, která bude spolupracovat s dodaným hardwarem, aby bylo možné přesně zaznamenávat docházku zaměstnanců a odpracované hodiny na zakázce. Tyto záznamy budou následně předávány aplikaci Cevis, která zajišťuje správu docházkových dat ve firmě Ivar.

4.1 Analýza požadavků

Pro účely vývoje aplikace je nezbytná důkladná analýza požadavků ve spolupráci se zákazníkem, tj. společností, která má aplikaci využívat. K získání kompletního seznamu požadavků byly uskutečněny tři osobní návštěvy, během nichž se zástupci firmy postupně zabývali a upřesňovali své požadavky. Tyto požadavky byly pečlivě shromážděny a jsou zahrnuty v tomto dokumentu, který slouží jako základní zdroj informací pro další kroky vývoje aplikace.

4.1.1 Popis klienta

Soukromá společnost založena v roce 1991 působící po celé České republice se specializuje na realizaci telefonních sítí, datových a optických tras, bezdrátových rozhlasů, výstavbu trafostanic a podobně. Firma zaměstnává okolo 70 zaměstnanců, které lze rozdělit do dvou skupin. Na zaměstnance, kteří vykonávají práci v areálu firmy (administrativa, projekce, sklad, příprava) a na zaměstnance terénní, kteří po většinu dní v měsíci vykonávají práci mimo areál firmy.

Zaměstnanci jsou povinni zaznamenávat na docházkovém terminálu začátek a konec pracovní doby a na papírový formulář uvést, na jaké zakázce pracovali daný den. U zaměstnanců, kteří pracují v terénu, se vyplnění formuláře provádí vždy ráno, při specifikaci práce nadřízeným. Zaměstnanci pracující v prostorách firmy, vyplňují formulář v průběhu dne, protože může nastat situace, že v daný den pracují na více zakázkách. Následně 1x týdně tyto formuláře přepisuje asistentka do výkazu práce v docházkovém

systému Cevis. Tento proces vkládání dat do systému se však ukázal nepraktický a velice znesnadňuje práci managementu při sledování již odpracovaných hodin u jednotlivých zakázek.

Jedná se o dlouholetého klienta, u kterého provozujeme docházkový systém včetně sledování práce na zakázkách. Toto je však možné pouze u těch zakázek, které jsou ručně do systému vloženy. Byli jsme poptáni o nový docházkový terminál, který by umožňoval evidenci zakázek a nahradil papírový formulář včetně zápisu dat do docházkového systému. Protože na trhu nebyl dostupný žádný docházkový terminál s touto funkcí, rozhodli jsme se vyvinout aplikaci, která na dotykovém zařízení bude emulovat docházkový terminál.

4.1.2 Funkční požadavky

Funkční požadavky, jak je již zmíněno v kapitole 3.2.1 na straně 21 popisují, co systém musí dělat, aby splnil svůj účel. Jedná se o požadavky ohledně funkčnosti systému, kde jsou popsány požadované vlastnosti. V následující tabulce, je výčet požadavků a níže jsou popsány jejich specifikace.

Označení	Popis
FP 1	Autentizace zaměstnance na základě RFID čipu
FP 2	Administrátorský přístup do správy terminálu pomocí hesla
FP 3	Možnost zvolit definované typy nepřítomnosti
FP 4	Po načtení IDM, systém informuje o platnosti záznamu
FP 5	Možnost umožnit zaměstnancům zobrazit své docházkové záznamy
FP 6	Před autentizací možnost volby, že se bude jednat pouze o práci na zakázce
FP 7	Po úspěšné autentizaci docházkového záznamu umožnit uživateli vybrat příslušnou zakázku
FP 8	Systém bude nabízet pouze platné zakázky
FP 9	Při volbě zakázky bude přednabídnuto prvních pět nejpoužívanějších
FP 10	Automatické zrušení předchozí volby
FP 11	Systém umožní nastavení intervalů pro automatické doplňování dat
FP 12	Systém se přepne do offline režimu v případě ztráty konektivity se serverem
FP 13	Systém bude umožňovat administrátorovi měnit heslo
FP 14	Systém umožní nastavení konverze kódu
FP 15	Systém umožní změnu směru čteček
FP 16	Systém umožní administrátorovi definování nepřítomností

Tabulka 1: Funkční požadavky (zdroj: vlastní)

FP 1 – Autentizace zaměstnance

Systém bude umožňovat zaměstnancům autentizovat se pomocí RFID čipu. Každý zaměstnanec bude mít přidělen jedinečný RFID čip, který bude obsahovat unikátní identifikátor. Zaměstnanec bude moci autentizovat svou identitu pouhým přiložením svého RFID čipu k čtečce.

FP 2 – Administrátorský přístup

Systém bude umožňovat administrátorovi přístup do správy terminálů pomocí hesla. V každém terminál bude přednastaveno defaultní heslo pro správce, který bude mít přístup k jeho konfiguraci a nastavení. Pokud se jedná o první přihlášení, bude administrátor vyzván k vytvoření nového hesla.

FP 3 – Volby nepřítomností

Systém bude umožňovat zaměstnancům volbu nepřítomnosti, které budou předem definovány. Každý zaměstnanec bude moci zvolit z předem navržených typů nepřítomností (nemoc, dovolená, pracovní cesta, apod). Zaměstnanec si vybere relevantní typ nepřítomnosti a následně se načte na RFID čtečce. Definice nepřítomností viz FP 16.

FP4 – Vyhodnocení načteného IDM

Po přiložení IDM (identifikační médium) na čtečku, systém zkontroluje platnost. V případě kladného vyhodnocení se na displeji zobrazí jméno vlastníka IDM a směr průchodu. V opačném případě bude zobrazena hláška „Neplatná karta“ a její vnitřní kód.

Aby bylo IDM kladně vyhodnoceno musí splnit tři podmínky:

1) Přiřazeného vlastníka
2) Nezablokovaný
3) Platné období

FP 5 – Náhled na docházkové záznamy

Systém bude umožňovat zaměstnancům přístup k jejich docházkovým záznamům. Po platné autentizaci pomocí IDM, bude zaměstnanci umožněn náhled na své záznamy o docházce. Tyto data budou přebírány z docházkového systému Cevis a bude zobrazena celá měsíční agenda (měsíční plán, záznamy, kumulace).

Tato funkcionalita bude pro zaměstnance, kteří chtějí mít přehled o své docházce, zkontrolovat své záznamy a případně upozornit na chyby nebo nesrovnalosti v docházkových záznamech.

FP 6 – Pořízení záznamu pouze na vybrané zakázky

V případě, že zaměstnanec pracuje v jednom dni na více zakázkách, není třeba pořizovat nový docházkový záznam. Stačí pouze informovat systém o tom, na které zakázce zaměstnanec začíná pracovat. Před autentizací zaměstnance bude systém nabízet možnost pořízení záznamu pouze pro vybranou zakázku.

FP 7 – Automatická nabídka vyplnění práce na zakázce

Po úspěšné autentizaci zaměstnance v docházkovém systému bude automaticky nabídnuta možnost výběru zakázky, na které zaměstnanec začíná pracovat. Uživatel si bude moci z nabízeného seznamu vybrat příslušnou zakázku, na které chce pracovat. Tato informace bude následně uložena jako součást docházkového záznamu pro daný pracovní den.

FP 8 – Platné zakázky

Systém bude kontrolovat platnost zakázek a nabízet pouze ty, které jsou aktuální a platné. Pokud je nějaká zakázka zrušena nebo již není platná, nebude nabízena jako možnost pro zaměstnance při výběru zakázky v terminálu.

FP 9 – Nabídka nejčetnějších zakázek

Systém bude analyzovat historické záznamy docházkového systému a na základě četnosti použití jednotlivých zakázek bude vybráno prvních pět nejčastěji používaných. Tyto zakázky budou přednabídnuty uživateli při výběru zakázek v systému. Pokud uživatel nevybere žádnou z těchto nabídnutých zakázek, systém umožní vybrat jakoukoliv jinou platnou. Přednabídnuté zakázky se budou aktualizovat na základě aktuálních dat uložené v databázi docházkového systému Cevis. Tento způsob výběru zakázek umožní zaměstnancům rychleji a snadněji najít a vybrat zakázku, na které pracují nejčastěji.

FP 10 – Automatické zrušení předchozí volby

Pokud zaměstnanec na terminálu stiskne nepožadovanou nepřítomnost, může tuto volbu změnit stisknutím jiné nepřítomnosti. Systém automaticky volbu zruší po určitém časovém intervalu nečinnosti (defaultně nastaveno 8 sekund) a navrátí se do úvodní nabídky.
Časová prodleva nečinnosti je nastavitelná administrátorem.

FP 11 – Intervaly pro načítání a stahování dat

Terminál automaticky vyčítá data o zaměstnancích, platných kartách, seznamu zakázek a nepřítomnostech. Tyto informace jsou ukládány na terminálu pro rychlejší přístup a zajištění funkčnosti i při výpadku konektivity s databázovým serverem. Kromě toho, terminál odesílá data pořízených záznamů do centrální databáze, aby tyto informace byly dostupné pro další zpracování a analýzy.

FP 12 – Přepnutí do offline režimu

V případě, že dojde k výpadku spojení terminálu s databázovým serverem, systém se automaticky přepne do *offline* režimu. Tento režim umožní zaměstnancům i nadále využívat terminál pomocí RFID čipu, ale pořízené záznamy se nebudou okamžitě ukládat do centrální databáze. Namísto toho se budou tato data dočasně ukládat na samotném terminálu.

Po obnovení spojení s databázovým serverem se data uložená v *offline* režimu automaticky synchronizují s centrální databází.

Přepnutí do *offline* režimu a synchronizace dat by měly probíhat automaticky bez nutnosti zásahu uživatele, aby byl minimalizován čas, kdy jsou data nedostupná.

FP 13 – Změna administrátorského hesla

Systém umožní administrátorovi změnit své heslo pro přístup do správy terminálu. Administrátor bude mít možnost zadat své staré heslo a následně vytvořit nové heslo. Nové heslo bude muset splňovat určitá bezpečnostní kritéria, jako například délka hesla, použití kombinace číslic, písmen a speciálních znaků. Po zadání nového hesla bude systém ověřovat, zda nové heslo splňuje všechna bezpečnostní kritéria a poté uloží nové heslo do databáze.

FP 14 – Nastavení formátu kódu.

Z důvodu různorodosti RFID technologie systém bude umožňovat nastavení konverze kódu pro případ, že bude potřeba převést kód z jednoho formátu do druhého. Administrátor bude mít možnost vybrat z předdefinovaných konverzních vzorců.

FP 15 – Změna směru čteček

Systém umožní administrátorovi změnit směr čteček na terminálu. Bude možné nastavit směr pozice čtení příchodu a odchodu. Tato volba umožní použití terminálu v různých konfiguracích, v závislosti na umístění terminálu a preferencích uživatele.

FP 16 – Definice nepřítomností

Systém umožní administrátorovi definovat seznam nepřítomností, které se budou zobrazovat na displeji terminálu. Administrátor bude mít možnost přidat, odebrat nebo upravit pořadí nepřítomností v seznamu. Pokud zaměstnanec zvolí určitou nepřítomnost na terminálu, bude mu automaticky přiřazena a uložena v systému. Seznam nepřítomností bude pravidelně aktualizován v nastavených intervalech viz. FP – 11.

4.1.3 Nefunkční požadavky

Nefunkční požadavky se zaměřují na vlastnosti systému, které souvisejí s jeho výkonem, spolehlivostí, bezpečností, udržovatelností a dalšími charakteristikami, které jsou důležité pro správné fungování systému, ale nejsou přímo spojeny s jeho funkcemi. Tyto požadavky jsou obvykle definovány s ohledem na požadavky zákazníka na výsledný produkt a na jeho využívání v daném prostředí.

Označení	Popis
NP 1	Systém bude dostupný v režimu 7/24
NP 2	Systém bude logovat události a aktivity
NP 3	Autentizace pouze pomocí RFID čipu
NP 4	Systém bude schopen pracovat v režimu s dvěma čtečkami
NP 5	Spustitelné pod systémem Windows
NP 6	Aplikace by měla splňovat snadno použitelnou instalaci a konfiguraci
NP 7	Aplikace by měla být uživatelsky přívětivá a intuitivní
NP 8	Aplikace musí umožňovat ovládání pomocí dotykové obrazovky

Tabulka 2: Nefunkční požadavky (zdroj: vlastní)

NP 1 – Dostupný v režimu 7/24

Systém musí být dostupný v režimu 7/24. To znamená, že systém musí být k dispozici 24 hodin denně, 7 dní v týdnu bez výpadků nebo omezení, aby uživatelé mohli používat systém kdykoliv, kdy je zapotřebí.

NP 2 – Logování

Systém bude automaticky zaznamenávat všechny relevantní události a aktivity, jako jsou přihlášení uživatelů, změny nastavení, požadavky na server, přístupové pokusy a další události. Logování bude probíhat v reálném čase a bude umožňovat pozdější analýzu. Data v logovacích souborech budou chráněna a zabezpečena proti neoprávněnému přístupu.

NP 3 – Autentizace RFID čipem

Systém umožní pouze autentizaci uživatele pomocí RFID čipu a neumožní jiné formy autentizace, jako například heslo či biometrické údaje.

NP 4 – Režim Dual reader

Systém bude umožňovat použití dvou čteček pro autentizaci zaměstnanců. Tento režim umožní rychlejší a efektivnější přístup zaměstnanců k terminálům, zvláště v případě vysokého počtu zaměstnanců nebo v oblastech s vysokou návštěvností. Jednoznačně bude určeno, která čtečka je určena pro směr příchod a odchod.

NP 5 – Běh aplikace pod systémem Windows

Aplikace nebude pouze součástí terminálu, ale bude také nainstalována na osobních počítačích. Jedná se o malé detašované pracoviště o nízkém počtu zaměstnanců. Systém bude spustitelný na operačním systému Windows a bude kompatibilní s nejnovějšími verzemi. Budou dodávány také nezbytné ovladače a součásti pro plné fungování systému.

NP 6 – Snadná instalace

Aplikace by měla umožnit snadno použitelnou instalaci a intuitivní konfigurační proces, který by měl minimalizovat potřebu manuálního nastavení s cílem snížit riziko chyb a zjednodušit nasazení systému pro nové uživatele.

NP 7 – Uživatelský přívětivé

Uživatelské rozhraní by mělo být snadno ovladatelné a uživatelsky přívětivé, aby se minimalizovala možnost chyb a zvýšila efektivita práce s aplikací. Dále by měla být aplikace navržena s ohledem na různé typy uživatelů a jejich potřeby.

NP 8 – Ovládání pomocí dotykové obrazovky

Systém musí být navržen tak, aby umožňoval uživatelům ovládat terminál pomocí dotykové obrazovky. Uživatelské rozhraní bude snadno ovladatelné a bude mít dostatečně velká tlačítka, aby bylo možné je ovládat pomocí prstu nebo *stylusu*.

4.2 Návrh aplikace v UML

V této kapitole se zabývám návrhem systému pomocí modelovacího jazyka UML, který umožňuje vizuální reprezentaci a popis systému pomocí různých diagramů. Tento návrh usnadňuje porozumění jeho funkcionalit a komunikaci se zákazníkem a členy týmu, kteří se na jeho vývoji podílejí.

Budou vytvořeny tři základní typy diagramů - diagram případů užití, diagram tříd a stavový diagram. Tyto diagramy slouží k popisu funkcionalit systému, jeho datových struktur a stavů, ve kterých se může nacházet.

Diagram případů užití bude sloužit k popisu funkcionalit systému z pohledu uživatelů. Diagram tříd popisuje strukturu tříd a vztahy mezi nimi, tedy jak jsou třídy navzájem propojeny a jaké atributy a metody obsahují. Stavový diagram bude sloužit k popisu stavů, ve kterých se může systém nacházet a jak se mezi nimi přechází.

4.2.1 Diagram případu užití

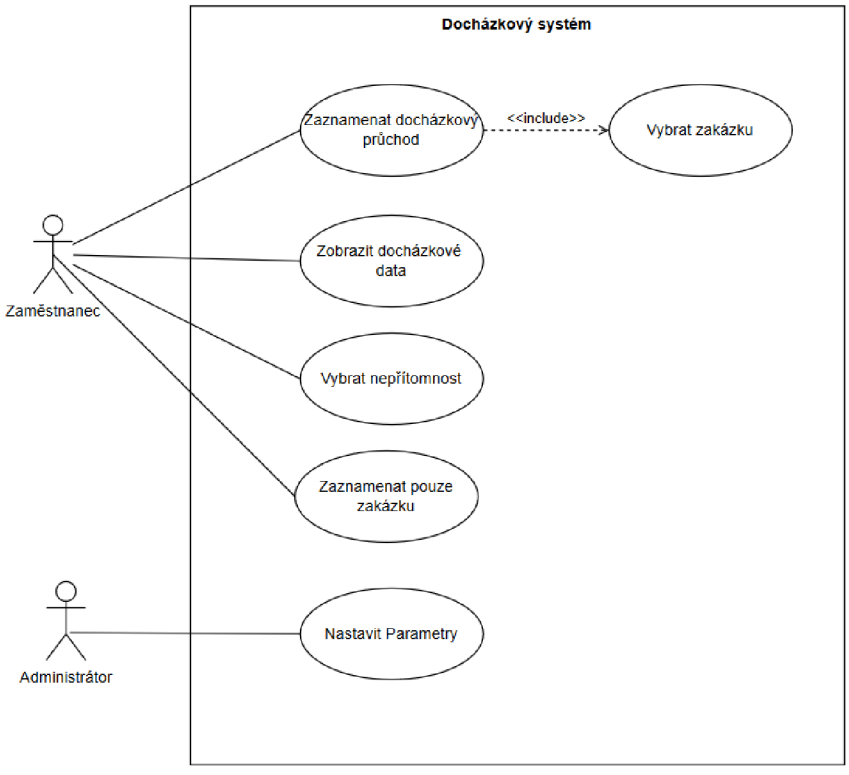
Diagram případu užití (Use Case diagram) je jedním z nejdůležitějších diagramů v UML, protože poskytuje celkový pohled na to, jakými způsoby mohou uživatelé využívat navrhovaný systémem. Diagram použití popisuje různé případy použití nebo scénáře, které výsledný systém může obsahovat.

Hlavním cílem Use Case diagramu je definovat chování systému z hlediska uživatelských požadavků. Tyto požadavky mohou být shromážděny z různých zdrojů, jako jsou požadavky zákazníků, obchodní analýza. Tyto požadavky jsou poté transformovány na funkční požadavky, které jsou základem pro tvorbu Use Case diagramu, uvedené v kapitole 4.1.2 Funkční požadavky.

V diagramu jsou definováni aktéři, kteří používají systém a mohou ovlivnit jeho chování. Aktéři jsou reprezentováni jako osoby, role nebo jiné systémy, které interagují s naším systémem. Každý aktér může mít vazbu na jeden nebo více případů užití, které se ho týkají.

V následujícím diagramu jsou uvedeni aktéři: *Zaměstnanec, Administrátor*. Aktér *Zaměstnanec* zastupuje všechny zaměstnance, kteří budou na terminálu pořizovat docházkové záznamy.

Aktér *Administrátor* má přístup k systému pouze z hlediska nastavení, což znamená, že jeho funkce spočívají v konfiguraci a správě systému, nikoli v provádění běžných operací v aplikaci.

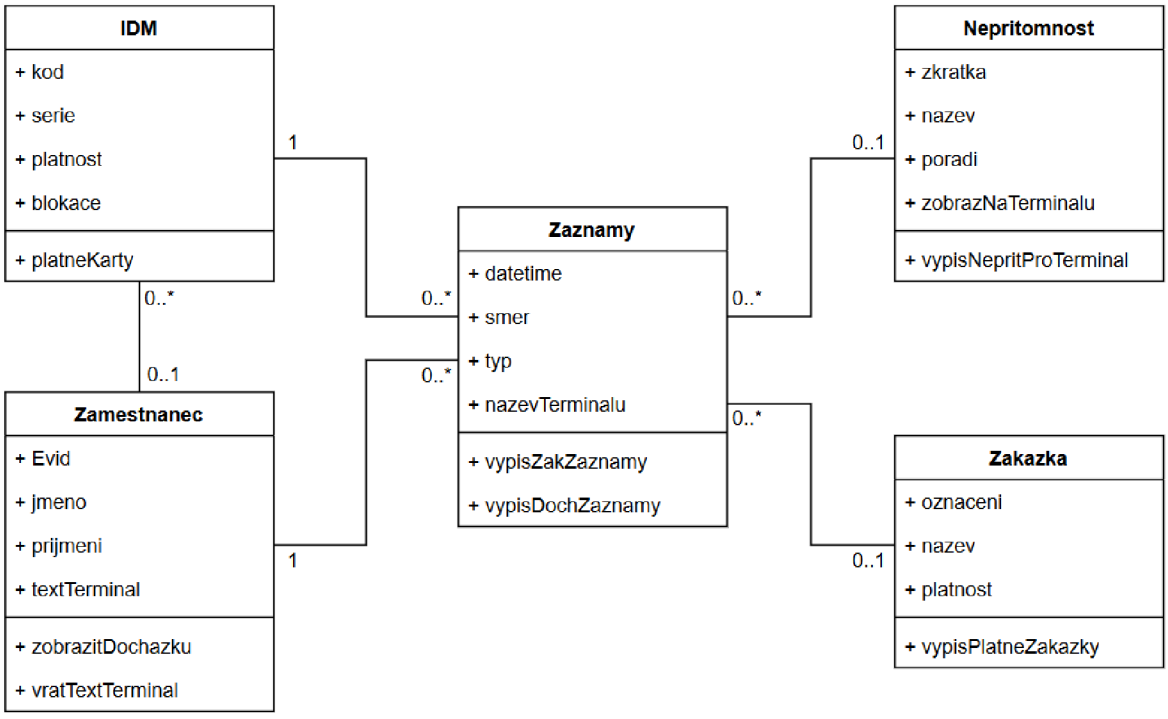


Obrázek 17 – Diagram případů užití (zdroj: vlastní)

4.2.2 Diagram tříd

Diagram tříd (*Class diagram*) je klíčovým nástrojem pro přehledné a efektivní modelování softwarového systému a jeho návrh. Slouží k popisu statické struktury systému a jasně vymezuje třídy objektů a jejich vzájemné vazby. Pomocí grafické vizualizace umožňuje snadno pochopit a reprezentovat složitost daného systému a komunikaci mezi jednotlivými třídami.

Diagram tříd je nezbytným prvkem procesu modelování, jelikož umožňuje definovat klíčové třídy objektů, které jsou nezbytné pro návrh budoucího systému. Grafická vizualizace umožňuje lepší pochopení složitosti systému a interakcí mezi jednotlivými třídami.



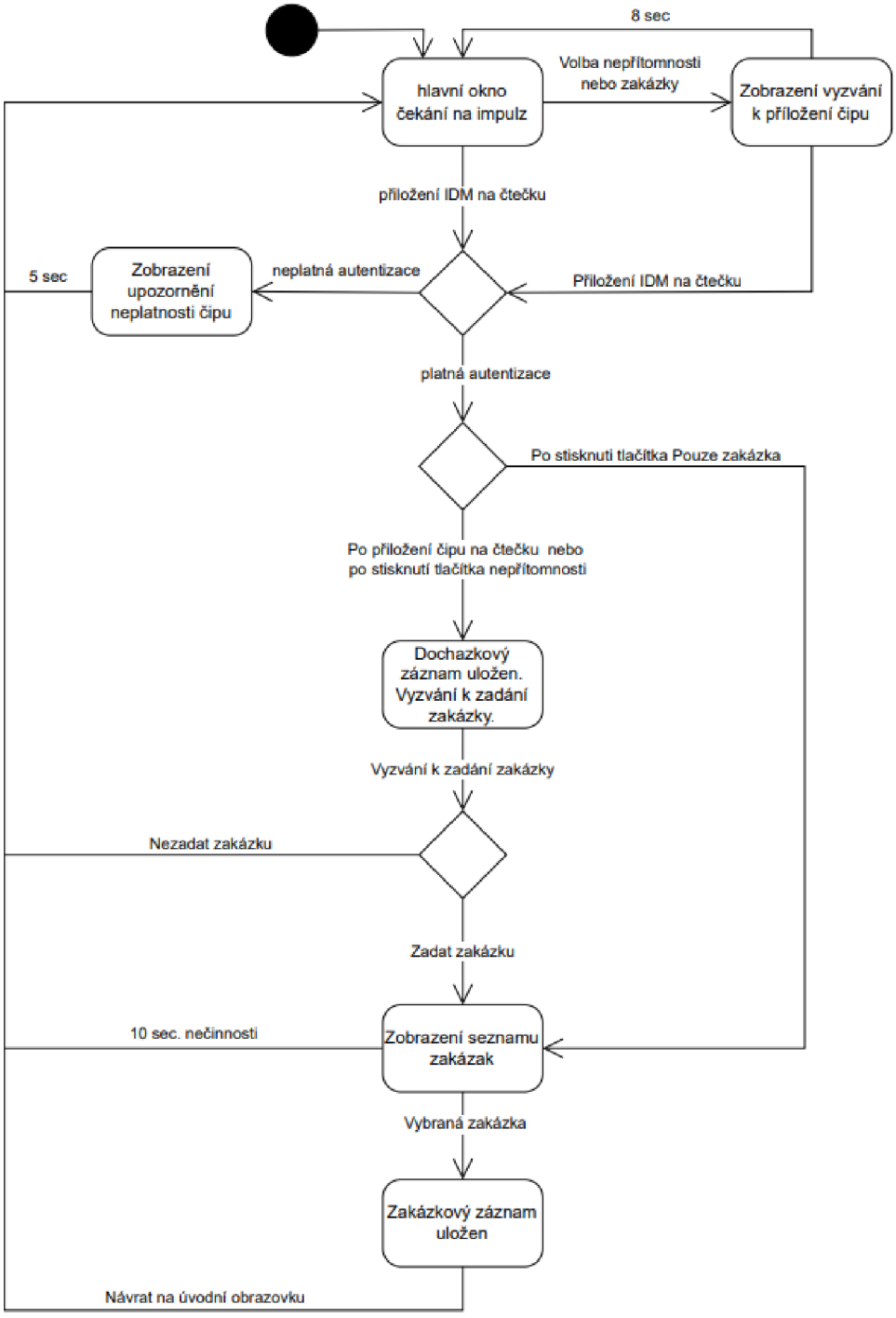
Obrázek 18 - Diagram tříd (zdoj: vlastní)

Byl vytvořen diagram tříd obsahující pět tříd. Třída *Zaměstnanec* obsahuje informace o zaměstnancích, kde evidenční číslo, jméno a příjmení jsou charakteristikami osoby. V atributu textTerminal je uložena informace, která se po identifikaci zobrazí na displeji terminálu. Tento atribut může být využit například pro zobrazení vypočítaného salda

zaměstnance z docházkového systému Cevis. Třída *IDM* reprezentuje identifikační média, v tomto případě RFID čipy, u kterých je nutné mít vyplněný kód karty, platnost a blokaci. Pokud nejsou splněny podmínky platnosti nebo blokace, čip není přijat. Třída *Záznamy* obsahuje informace o jednotlivých pořízených záznamech zaměstnanců, která jsou společná pro docházková i zakázková data. Třída *Nepřítomnost* definuje různé typy nepřítomností zaměstnanců, včetně informace, zda je daná nepřítomnost viditelná pro aplikaci (atribut *zobrazNaTerminalu*). Určení pořadí určuje řazení zobrazených nepřítomností. Třída *Zakázka* obsahuje seznam zakázek a jejich platnosti.

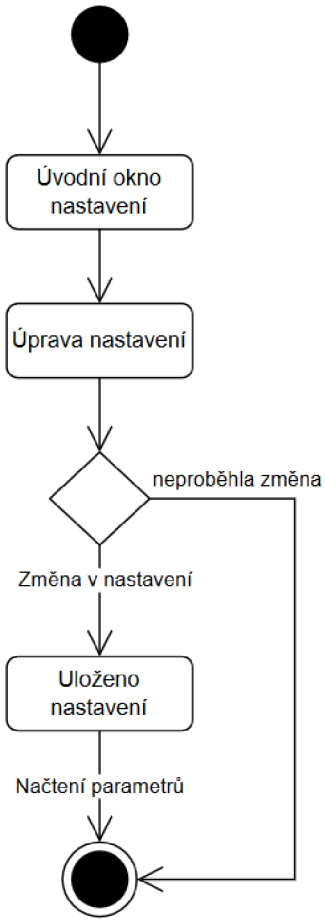
4.2.3 Stavový diagram

Stavové diagramy umožňují graficky vyjádřit chování systému nebo objektu. Tyto diagramy se používají k popisu toho, jak se systém mění v reakci na různé vstupy a jaké stavy může systém nabývat. Na obrázku 19 je vymodelován diagram stavů hlavního okna, který popisuje stavy při pořízení docházkového nebo zakázkového záznamu.



Obrázek 19 - Diagram stavů hlavního okna (zdroj: vlastní)

Na Obrázku 20 je zobrazen druhý stavový diagram, který popisuje okno nastavení. Hlavními stavy v tomto diagramu je zobrazení nastavení, jeho úprava a následné aplikování.



Obrázek 20 - Stavový diagram pro Nastavení (zdroj: vlastní)

4.3 Návrh uživatelského prostředí

Wireframe (drátěný model, popsán v kapitole 3.4 na straně 30).

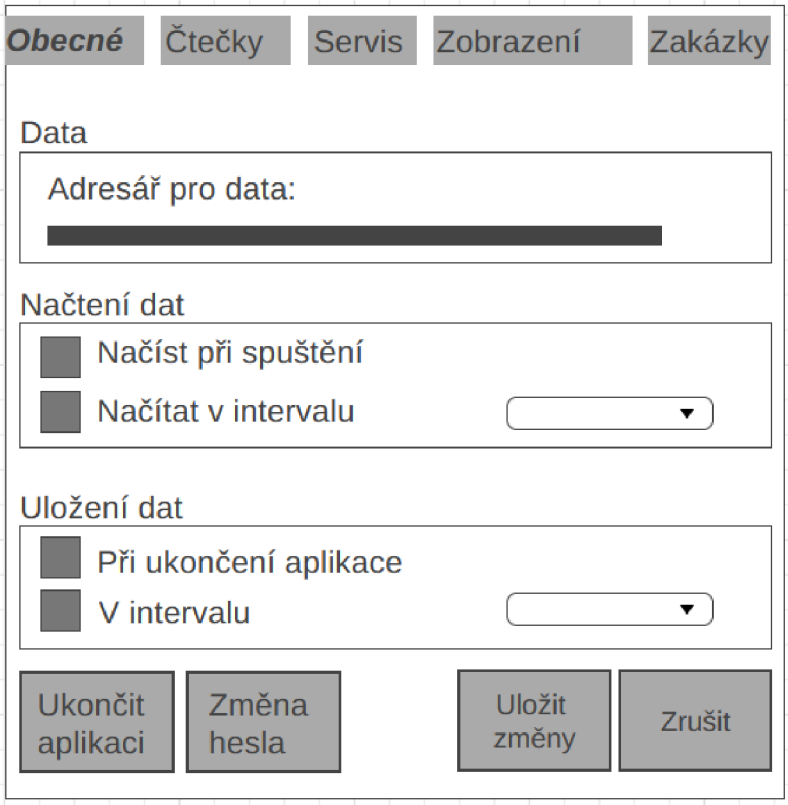
Na obrázku 21 je zobrazeno hlavní okno, které je klíčovým prvkem aplikace. Skládá se ze dvou částí: sdělovací a uživatelské. V horní části okna se nachází informační panel, který v klidovém režimu zobrazuje aktuální datum a čas. Po přiložení identifikačního média (IDM) na čtečku je datum a čas změněn na výsledek vyhodnocení autentizace. Pokud autentizace proběhla úspěšně, na obrazovce se zobrazí jméno zaměstnance, vnitřní kód IDM, text zaměstnance a popis „Platný záznam“. Pokud autentizace selže, zobrazí se pouze vnitřní kód IDM a popis „Neplatný záznam“. Po stisku některého z tlačítek

v uživatelské části se na displeji zobrazí informace o typu tlačítka a vyžádá o přiložení IDM.

V uživatelské části je zobrazen seznam tlačítek. Tyto tlačítka reprezentují nepřítomnosti, které jsou povolené zaměstnancům na terminálu. Na obrazovce je zobrazen seznam až dvanácti přerušení, která jsou zaměstnancům umožněna k výběru. Definice těchto přerušení se provádí v rámci systému Cevis. Pokud je tento počet nedostačující, lze ve spodní části stisknout tlačítko „Ostatní nepřítomnost“, následně se zobrazí rolovací nabídka zbylých přerušení. Poslední tlačítko pod označením "Pouze zakázky" umožňuje zaměstnanci zadat práci na zakázce bez záznamu o docházce.

Obrázek 21 - Návhr uživatelského rozhraní hlavního okna (zdroj: vlastní)

Druhý model zahrnuje okno nastavení, které je přístupné pouze administrátorovi. V horní liště se nachází menu pro přepínání mezi jednotlivými stránkami, které slouží k nastavení potřeb klienta. Mezi základní nastavení patří možnost zadat cestu pro ukládání systémových souborů, nastavení komunikačních intervalů s databází docházky, specifikace čtečky a nastavení formátu kódu využívaných IDM.



Obrázek 22 - Návrh uživatelského rozhraní pro okno Nastavení (zdroj: vlastní)

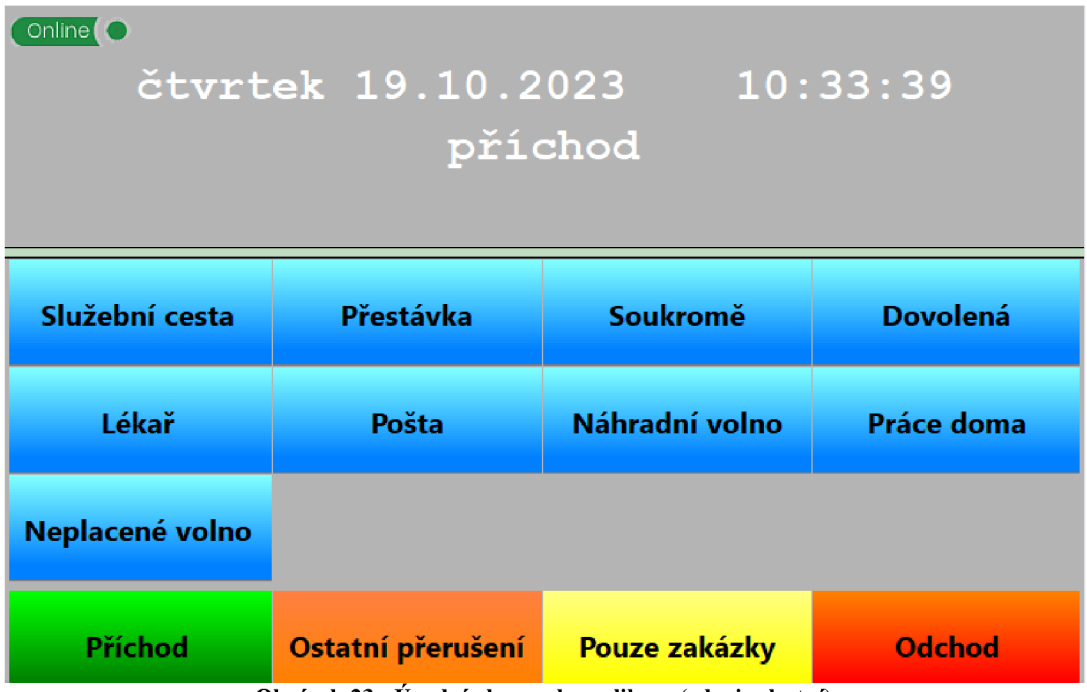
4.4 Implementace aplikace

Aplikace pro emulaci docházkového terminálu, byla vytvořena v prostředí Embarcadero Delphi 10.4 Sydney. Toto programovací prostředí se vyznačuje vhodností pro vývoj nativních aplikací pro operační systémy Windows, Android, iOS a Mac. Na základě předchozích návrhů, byla aplikace strukturována do tří jednotek (*units*). V kontextu jazyka Delphi je unita definována jako samostatná část programu, typicky představující jedno okno nebo jiný funkcionalitou zaměřený segment aplikace. Unita může být i čistě datovou entitou. V případě této aplikace jsou využity dvě unity pro reprezentaci grafického prostředí a jedna pro datovou strukturu. Klíčovou unitou je úvodní okno, uloženo pod názvem *uMain.pas*, kde je zahrnuta nejpodstatnější část řízení celé aplikace. Druhou unitou je Nastavení a v třetí je uložena datová struktura pro práci v *offline* režimu.

4.4.1 Unita uMain

Jak bylo dříve zmíněno entita *uMain* reflektuje úvodní obrazovku, sloužící k zaznamenávání docházky pro uživatele. Tato entita byla vytvořena s ohledem na grafický návrh a zahrnuje celkem 23 procedur. Následující část se věnuje důležitým komponentám použitým pro grafické zobrazení a popisu důležitých procedur.

Úvodní obrazovka programu je strukturovaná komponentou *TPanel*, která ji rozděluje na dvě části. V horní části jsou za použití komponent *TLabel* zobrazeny aktuální datum a čas. Po načtení identifikačního média (IDM), se v labelu zobrazí jméno majitele a vnitřní kód IDM. Druhá část, nacházející se ve spodní části obrazovky, využívá komponentu *TJvNavPanelButton* pro zobrazení jednotlivých možností nepřítomností, které jsou nadefinované v docházkové aplikaci Cevis. Uživatel má možnost označit pouze zakázku nebo vybrat směr (Příchod/odchod), za předpokladu, že je aplikace nastavena pro jednočtečkový provoz. Obrázek 23 ilustruje úvodní okno aplikace s popsanými grafickými komponenty.



Obrázek 23 - Úvodní obrazovka aplikace (zdroj: vlastní)

Mezi prvními procedurami, které jsou spouštěny při náběhu aplikace, se nachází procedura s názvem *SetCauseMenu*. Tato procedura má za úkol načíst data z datového zdroje *cdsExitFkey*, což představuje seznam definovaných nepřítomností v rámci docházkového

programu Cevis, a následně je zpracovat. S ohledem na předem stanovené pořadí jsou tlačítka nazývána konkrétními názvy nepřítomností a jsou zviditelněna v uživatelském rozhraní. Vlastnost „*Tag*“ u těchto tlačítek slouží k uložení identifikátoru příslušné nepřítomnosti, což umožňuje aplikaci správně rozpoznat a pracovat s jednotlivými typy nepřítomností.

```
procedure TfrmMain.SetCauseMenu;
var
  Button : TJvNavPanelButton;
  Item: TMenuItem;
  i: Integer;
begin
  ppmCause.Items.Clear;
  with dmMain.cdsExitfkey do begin
    IndexName := 'ByKey';
    First;
    i := 0;
    while not Eof do begin
      if dmMain.cdsExitfkeyKey.Value = '' then Button := nil
      else begin
        case i of
          0: begin Button := btnCause1; Button.Visible := True; end;
          1: begin Button := btnCause2; Button.Visible := True; end;
          2: begin Button := btnCause3; Button.Visible := True; end;
          3: begin Button := btnCause4; Button.Visible := True; end;
          4: begin Button := btnCause5; Button.Visible := True; end;
          5: begin Button := btnCause6; Button.Visible := True; end;
          6: begin Button := btnCause7; Button.Visible := True; end;
          7: begin Button := btnCause8; Button.Visible := True; end;
          8: begin Button := btnCause9; Button.Visible := True; end;
          9: begin Button := btnCause10; Button.Visible := True; end;
          10:begin Button := btnCause11; Button.Visible := True; end;
          11:begin Button := btnCause12; Button.Visible := True; end;
        else Button := nil;
        end;
        inc(i);
      end;

      if Assigned(Button) then begin
        Button.Caption := dmMain.cdsExitfkeyText.Value;
        Button.Tag := dmMain.cdsExitfkeyCause.Value;
      end
      else begin
        Item := NewItem(dmMain.cdsExitfkeyText.Value, 0, False, True,
        mniOnActionClick, 0, '');
        Item.Tag := dmMain.cdsExitfkeyCause.Value;
        ppmCause.Items.Add(Item);
      end;
      Next;
    end;
  end;
end;
```

Obrázek 24 - procedura SetCauseMenu (zdroj: vlastní)

Pro správu možností načítání identifikačních médií v prostředí této aplikace byla implementována komponenta *TAfComPort*. Tato komponenta je zodpovědná za komunikaci se čtečkou a umožňuje načítání dat z identifikačních médií. Samotný kód procedury, *PortDataReceived*, definuje chování, které je spouštěno v okamžiku, kdy jsou obdržena data od čtečky. Procedura obsahuje následující klíčové části:

- Procedura nejprve kontroluje, zda má aplikace více čteček (GDveCtecky = True). V případě, že má, je prováděna identifikace čtečky, která právě zaslala data.

- Poté probíhá načítání dat ze zvolené čtečky (*Port* nebo *PortOdchod*).

- Zpracování dat: Příchozí data jsou následně zpracovávána, kde se napřed určuje délka přijatých dat a kontroluje se počet výskytů znaku #13 (*carriage return*) pro zamezení potenciálních duplicitních záznamů.

- Pokud je identifikován znak #13 v přijatých datech, provádí se konverze těchto dat na vhodný formát a vytváří se zpráva, která obsahuje tato data. Tato zpráva je následně odeslána do aplikace prostřednictvím zprávy systému (*PostMessage*).

```
1510 procedure TfrmMain.PortDataRecived(Sender: TObject; Count: Integer);
     var
       PS: PString;
       delka, i: Integer;
     begin
       if GDveCtecky = True then
       begin
         if (Sender as TAfComPort) = Port then
         begin
           if tmrPort.Enabled = true then
1520       begin //opakovane cteni

           end
           else
           begin
             tmrPort.Enabled := True;
             Direction := 1;
             ReceiveData := ReceiveData + Port.ReadString;

             delka:= 0;
1530         for i := 0 to Length(ReceiveData) do
             begin
               if ReceiveData[i] = #13 then
                 Inc(delka);
             end;
1535         // Pokud zaznam prisel vicekrat (je smazan)
             if delka > 1 then ReceiveData := '';

             if Pos(#13, ReceiveData) > 0 then begin
               New(PS);
1540           PS^ := ConvertHexCodeByParam(Trim(ReceiveData), GConvertPromag,
               GReverseByte, GHexToDec, GHexToDecFull);
               PostMessage(Handle, WM_CARDISREAD, Integer(PS), 0);
               ReceiveData := '';
             end;
           end;
         end;

         if (Sender as TAfComPort) = PortOdchod then
         begin
1550       if tmrPortOdchod.Enabled = true then
           begin //opakovane cteni

           end
           else
           begin
             tmrPortOdchod.Enabled := true;
             Direction := 0;
             ReceiveData := ReceiveData + PortOdchod.ReadString;
```

Obrázek 25 - procedura PortDataRecived (zdroj: vlastní)

Procesní funkce *WritePreYMMDDByCard* představuje klíčový prvek docházkového systému, jehož cílem je zaznamenávat události příchodů a odchodů zaměstnanců. Významným aspektem této procedury je strategie ukládání těchto informací pomocí komponenty *TClientDataSet*, která ukládá data ve formátu XML do lokální paměti terminálu. Takový postup se osvědčil zejména v situacích, kdy dojde k nedostupnosti

centrální databáze. V těchto případech jsou data stále přístupná, což minimalizuje riziko nedostupnosti a zajišťuje nepřetržitou evidenci docházky.

Periodické odesílání dat z komponenty *TClientDataSet* do centrální databáze je prováděno v pravidelných časových intervalech. Tato strategie synchronizace umožňuje udržovat aktuálnost informací a minimalizovat ztrátu dat v případě přerušení komunikace. Celý tento proces hraje klíčovou roli v efektivní správě docházky zaměstnanců, přičemž umožňuje podrobný monitoring jejich pracovní aktivity.

```
function TdmMain.WritePreYMMDDByCard(CardCode: string; Cause: Integer): Boolean;
var
  FindEmploee: Boolean;
  Evid: Integer;
  N: TDateTime;
  function VyplnPopisZaznamu: string; ...
begin
  try
    frmMain.writeEntry := True;
    if frmMain.Direction = 1 then begin
1310  if GZakazky = True then
      begin
        if (mmdPreymmdd.RecordCount > 0) then
        begin
          WriteCdsPreymmdd;
        end;
      end;

      N := Now;
      Evid := GetCardOwner(CardCode, N);
1320  if Evid <= 0 then begin // toto by nemelo nastat
        frmMain.ShowLine12(CardCode, 'neplatná karta');
        WriteToLog('Nenalezen Majitel karty');
        Exit;
      end;

      FindEmploee := cdsEmploee.Locate('ID', Evid, []);
      if not Offline then
        SynchronizeTime;
      if FindEmploee then begin
1330    if Trim(cdsEmploeeWMSGLine2.Value) = '' then
          frmMain.ShowLine12(cdsEmploeeWMSGLine1.Value, 'Záznam: ' +
1332      VyplnPopisZaznamu)
        else
        frmMain.ShowLine12(cdsEmploeeWMSGLine1.Value,cdsEmploeeWMSGLine2.Value);

        if GZakazky = false then
        begin
          cdsPreYMMDD.Open;
          cdsPreYMMDD.Append;
1340      cdsPreYMMDDDatTim.Value := N;
          cdsPreYMMDDDir.Value := IntToStr(frmMain.Direction);
          cdsPreYMMDDCause.Value := Cause;
          cdsPreYMMDDCode.Value := CardCode;
          cdsPreYMMDDTerminal.Value := GetLocalComputerName;
          cdsPreYMMDDEvid.Value := Evid;
          cdsPreYMMDDTyp.Value := 0;
          cdsPreYMMDDExported.Value := False;
          cdsPreYMMDD.Post;
          cdsPreYMMDD.Close;
```

Obrázek 26 - funkce WritePreYMMDDByCard (zdroj: vlastní)

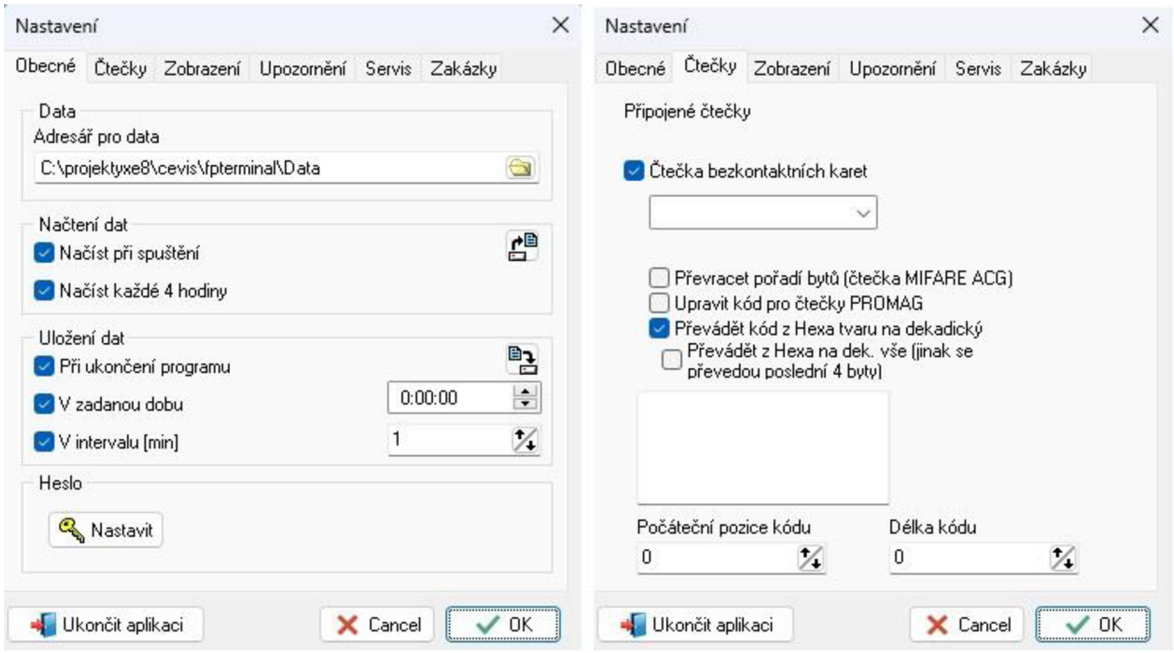
4.4.2 Unita uSettings

Za účelem konfigurace všech klíčových parametrů pro optimální provoz aplikace byla implementována unita *uSettings* (Nastavení). Pro lepší organizaci a přehlednost je tato jednotka rozdělena do pěti sekcí, což je reflektováno v rámci komponenty *TPageControl*. Každá z těchto sekcí se zaměřuje na konkrétní aspekty nastavení a zajišťuje správné fungování aplikace.

První ze sekcí, označená jako „Obecné“, poskytuje uživatelům možnost konfigurace důležitých parametrů, jako je například definice cesty pro ukládání dat či intervaly pro operace stahování a nahrávání záznamů do centrální databáze. Tímto způsobem je uživatel schopen přizpůsobit aplikaci svým konkrétním potřebám a požadavkům.

Další karta „Čtečky“ je zaměřena na nastavení komunikace s čtečkami. Uživatel zde může definovat, na jakém komunikačním portu jsou čtečky připojeny a provést základní volby pro převod načtených identifikačních médií. Tato funkce je nezbytná pro přizpůsobení aplikace konkrétním potřebám klienta, zejména pokud existují v organizaci identifikační média (IDM) využívaná pro jiné účely, než je aktuální aplikace.

Na záložce „Zobrazení“ lze konfigurovat časový interval (v sekundách), po němž bude volba aktivní na úvodní obrazovce po stisku tlačítka nepřítomnosti. Toto nastavení umožňuje přesně definovat, jak dlouho má být daná volba viditelná na obrazovce po uživatelově interakci.



Obrázek 27 - Obrazovka okna Nastavení (zdroj: vlastní)

4.4.2.1 Přepočet ACG

Firma Ivar standardně implementuje čtečky od firmy *Promag*, což je důležité uvést v souvislosti s procesem přepočtu kódu, který je poskytován firmou ACG (Advanced Crypto Graphics). Společnost Ivar tímto způsobem zajišťuje, že čtečky od *Promagu* jsou kompatibilní s technologiemi a bezpečnostními řešeními, které nabízí firma ACG, umožňující efektivní a spolehlivou identifikaci s využitím RFID karet. Tento proces přepočtu kódu je nezbytný k dosažení plné kompatibility a sjednocení formátu mezi čtečkami od *Promagu* a technologiemi ACG.

Funkce *ConvertToACG* má za úkol provést přepočet vstupního kódu *ACode* v podobě řetězce. Prvně kontroluje délku vstupního kódu a případně přidá nulu na začátek, pokud je jeho délka lichého počtu. Následně provádí převod kódu tak, že řetězec je rozdělen na dvoupísmenné dvojice, a tyto dvojice jsou následně postupně obráceny a spojeny do výsledného řetězce *Result*.

```
function ConvertToACG(ACode: string): string;
begin
  Result := '';
  if Length(ACode) mod 2 > 0 then ACode := '0' + ACode;
  while ACode > '' do begin
    Result := Result + Copy(ACode, Length(ACode) - 1, 2);
    ACode := Copy(ACode, 1, Length(ACode) - 2);
  end;
end;
```

Obrázek 28 - funkce ConvertToACG (zdroj: vlastní)

4.4.2.2 Přepočet Promag

Funkce *ConvertToPromag* je přizpůsobena specifické technologii čtení identifikačního kódu z RFID karet, a to zejména u karet vyráběných firmou *Promag*. Proximitní karty této společnosti v minulosti používaly záznamy o kódech, které se lišily od standardního formátu tím, že místo pěti bajtů měly pouze čtyři bajty a zároveň měly obrácený pořadí bitů.

Postup pro převod identifikačního kódu z proximitní karty je následující:

1. Čtečka získá desetimístný hexadecimální kód, například: 010300C822.
2. První bajt (v tomto případě '03') je oddělen a zbytek kódu je upraven jako '00C822'.
3. Pro každý hexadecimální znak je provedena úprava pořadí bitů, například: '1' na '8' (0001 na 1000), '7' na 'E' (0111 na 1110), '5' na 'A' (0101 na 1010), '6' na '6' (0110 na 0110), a tak dále.
4. Výsledný hexadecimální kód ('0C003144') je následně převeden na dekadický formát ('201339204').

Tento proces konverze umožňuje pracovat s identifikačními kódy na proximitních kartách od společnosti *Promag*, jak bylo vyžadováno v aplikaci. Alternativní metoda konverze, která je využita v aplikaci zobrazena na obrázku 29, kde dochází k záměně znaků.

```
function ConvertToPromag(ACode: string): string;
var
  i: Integer;
begin
  Result := '';
  for i := 1 to Length(ACode) do
    case ACode[i] of
      '0': Result := Result + '0';
      '1': Result := Result + '8';
      '2': Result := Result + '4';
      '3': Result := Result + 'C';
      '4': Result := Result + '2';
      '5': Result := Result + 'A';
      '6': Result := Result + '6';
      '7': Result := Result + 'E';
      '8': Result := Result + '1';
      '9': Result := Result + '9';
      'a': Result := Result + '5';
      'A': Result := Result + '5';
      'b': Result := Result + 'd';
      'B': Result := Result + 'D';
      'c': Result := Result + '3';
      'C': Result := Result + '3';
      'd': Result := Result + 'b';
      'D': Result := Result + 'B';
      'e': Result := Result + '7';
      'E': Result := Result + '7';
      'f': Result := Result + 'f';
      'F': Result := Result + 'F';
    else Result := Result + ACode[i];
    end;
end;
```

Obrázek 29 - funkce ConvertToPromag (zdroj: vlastní)

4.4.3 Unita udmMain

V datové unitě *udmMain* jsou obsaženy instance komponenty *TClientDataSet*, což jsou jednotky reprezentující tabulky v databázi. Tyto komponenty jsou odpovědné za uchovávání a manipulaci s daty v rámci aplikace. Dále v této unitě nalezneme procedury, které slouží k efektivnímu zpracování těchto dat. Mezi tyto procedury patří operace spojené s *TClientDataSet*, jako je načítání, úprava a ukládání dat.

Unita *udmMain* také obsahuje procedury pro komunikaci s centrální databází. To zahrnuje operace, jako je stahování informací o zaměstnancích, identifikačních médiích, načítání dat

o nepřítomnostech, načítání informací o evidovaných zakázkách a obdobně. Důležitou částí je také odesílání záznamů o jednotlivých průchodech a evidenci práce na zakázkách zpět do centrální databáze. Tyto operace zajišťují aktuálnost a korektnost dat v aplikaci a umožňují efektivní správu pracovních procesů.

4.5 Použité komponenty

Pro strategické nasazení naší docházkové aplikace byl vybrán průmyslový počítač HIGOLE F11 od renomované společnosti HigolePC. Tato volba přináší několik výhod a významných technických parametrů, které jsou zásadní pro náš projekt, kde spolehlivost a nepřetržitý provoz hrají klíčovou roli. Průmyslový počítač HIGOLE F11 byl zkonstruován tak, aby plně vyhovoval nárokům nepřetržitého provozu 24 hodin denně, 7 dní v týdnu. To jej činí ideální volbou pro náročné podmínky a zajišťuje spolehlivý výkon. Hlavní technické parametry tohoto počítače zahrnují:

- Procesor N3450, který nabízí čtyři jádra a zároveň se vyznačuje vysokou energetickou efektivitou. Jeho běžný takt činí 1,1 GHz, s možností dynamického zvýšení na 2,2 GHz. Tento procesor je vyroben na 14nm výrobním procesu a má nízkou tepelnou zátěž (TDP) pouze 6 W, což přispívá k energetické úspornosti.

- Operační systém Windows 10 Pro zajišťuje bezproblémovou kompatibilitu s naším softwarem a poskytuje nástroje pro jednoduchou správu a údržbu systému.

- 11,6 palcový dotykový HD displej zvyšuje kvalitu uživatelské interakce a usnadňuje pohodlné používání naší aplikace.

- 4 GB operační paměti a 64 GB eMMC paměti zajistí dostatečný výkon pro plynulý běh aplikace a rychlý přístup k datům.

- Rozlišení obrazovky 1200x1080 s technologií IPS a světelností 250ANSI zaručuje kvalitní zobrazení a optimální čitelnost naší aplikace.

- Počítač disponuje dostatečným množstvím rozhraní, včetně tří portů RS232 a jednoho portu RS485, což umožňuje jednoduché propojení s dalšími zařízeními.

- K dispozici jsou také porty USB 3.0 a USB 2.0 pro připojení dalších periferií a zařízení.

Obrázek 30 - HiGole F11 (zdroj: vlastní)

Pro dokončení našeho docházkového terminálu jsme zahrnuli dvě RFID čtečky Promag UR220U. Tyto čtečky se vyznačují excelentním výkonem a jsou vhodné pro náročné provozní podmínky.

Klíčové charakteristiky těchto čteček:

- Odolnost vůči povětrnostním vlivům: Díky jejich odolné konstrukci jsou čtečky Promag UR220U schopny pracovat v nepříznivém počasí, což je ideální pro různé klimatické podmínky.

- Indikace LED: Tyto čtečky jsou vybaveny šesti volitelnými LED barvami pro vizuální signalizaci režimu čtení a pohotovostního režimu, což zvyšuje uživatelskou interakci a usnadňuje použití.

Technické specifikace čteček zahrnují:

- Frekvence: 13,56 MHz
- Rozsah čtení: 2-5 cm, zajišťuje spolehlivé a rychlé načítání identifikačních kódů.
- Rozhraní: Podporují USB Virtual COM PORT.
- Rozměry: UR220 má rozměry 100 x 32 x 16,5 mm, zatímco UR225 má rozměry 87 x 32 x 16,5 mm.
- Teplotní rozsah: Čtečky pracují v teplotním rozmezí od 0 °C do 55 °C.
- Vlhkostní tolerance: Od 10 % do 90 %.

Obrázek 31 - čtečka Promag UR220U (zdroj: vlastní)

5 Výsledky a diskuze

5.1 Zhodnocení a výsledky

V rámci této diplomové práce bylo úspěšně vyvinuto emulační řešení pro docházkový terminál, které splňuje stanovené cíle a požadavky definované na začátku projektu. Vývoj této aplikace přináší několik důležitých poznatků a výsledků, které mohou být užitečné pro další vývoj a využití v praxi.

Aplikace byla navržena tak, aby napodobovala chování fyzického docházkového terminálu a umožňovala efektivní evidenci práce zaměstnanců na jednotlivých zakázkách. Součástí diplomové práce je zdrojový kód o 4528 řádcích a tří souborů. Samostatný vývoj aplikace přinesl následující klíčové výsledky:

Funkčnost a kompatibilita: Výsledná aplikace byla úspěšně otestována a potvrdila svou funkčnost. Byla navržena tak, aby byla kompatibilní s existujícím hardwarem a softwarovými systémy, což umožňuje snadnou integraci do stávajícího pracovního prostředí.

Stabilita a spolehlivost: Během vývoje byl kladen důraz na stabilitu a spolehlivost aplikace. To zahrnovalo odolnost proti chybám a zajištění nepřetržitého provozu, což je důležité pro docházkový systém fungující 24/7.

Flexibilita a konfigurovatelnost: Aplikace byla navržena s ohledem na flexibilitu a konfigurovatelnost. Umožňuje nastavit různé parametry, intervaly pro stahování a ukládání dat do centrální databáze.

Integrace s RFID čtečkami: Aplikace byla úspěšně integrována s RFID čtečkami Promag UR220U. To umožňuje rychlé a spolehlivé načítání identifikačních kódů zaměstnanců a zaznamenání jejich přítomnosti.

Evidence práce na zakázkách: Aplikace neomezuje své využití pouze na docházkový systém, ale umožňuje také evidovat práci na zakázkách. Tato flexibilita zvyšuje její hodnotu a využití v různých provozních prostředích.

5.2 Testování aplikace

Aplikace byla testována během ostrého provozu v reálném podnikovém prostředí na naší firmě, kde jedno z oddělení kompletně zaznamenávalo docházku a evidovalo práci na zakázkách. Během tohoto měsíčního testování byly sledovány následující klíčové aspekty, které poskytly cenné zkušenosti a zpětnou vazbu:

Použitelnost a uživatelská přívětivost: Uživatelé byli pozorováni při používání aplikace, a to včetně navigace v ní. Zjištění zahrnovala to, jak snadno se uživatelé naučili aplikaci používat a jak rychle se stali schopni se v ní orientovat.

Stabilita a spolehlivost: Stabilita aplikace byla sledována během testování, a to včetně odhalení chyb, pádů nebo nežádoucího chování aplikace. Všechny zjištěné problémy byly dokumentovány a postupně odstraňovány.

Výkon: Zatížení hardwaru a výkon aplikace byly sledovány během různých úkonů a operací v aplikaci. Cílem bylo zajistit, že aplikace efektivně využívá zdroje a má akceptovatelnou odezvu.

Přenosy dat do databáze: Funkčnost přenosů dat do centrální databáze byla důkladně sledována. Zajištění, že data byla spolehlivě a bez ztrát přenášena, bylo jedním z klíčových cílů.

5.3 Diskuze - návrhy na vylepšení

Vzhledem k pozitivním výsledkům testování aplikace pro evidenci docházky a práce na zakázkách na firmě Ivar jsme identifikovali několik možných způsobů, jak tuto aplikaci dále vylepšit. Tyto návrhy na vylepšení by mohly poskytnout značný přínos pro efektivitu a pohodlí uživatelů, a tím i pro celkový provoz.

Využití čtečky čárových kódů pro snadné vyhledávání zakázek:

Jeden z klíčových návrhů na vylepšení zahrnuje implementaci čtečky čárových kódů do aplikace. Tím by zaměstnanci mohli snadno vyhledat konkrétní zakázky prostřednictvím načtení čárového kódu. To by výrazně zrychlilo proces vyhledávání a eliminaci manuálního zadávání zakázkových čísel, čímž by se minimalizovala rizika spojená s chybami.

Rozšíření podpory hardware:

Rozšíření kompatibility aplikace s dalším hardwarem, jako jsou různé typy čteček a biometrická zařízení, by umožnilo aplikaci být ještě flexibilnější a lépe přizpůsobný potřebám uživatelů.

Rozšíření dostupnosti mobilní aplikace:

Vytvoření mobilní verze aplikace by umožnilo zaměstnancům evidence docházky a práce na zakázkách přímo v terénu. Tato rozšíření by byla zejména pro mobilní pracovníky mimořádně výhodná, zvyšující jejich efektivitu a pohodlí.

6 Závěr

Docházkové systémy představují nepostradatelný prvek moderního podnikání, a to v mnoha různých odvětvích. Umožňují podnikům efektivně sledovat docházku zaměstnanců, které v konečném důsledku ovlivňuje výkonnost, produktivitu a náklady na provoz. Záznam docházky a evidování pracovních výkonů na konkrétních zakázkách jsou klíčovými faktory pro plánování a optimalizaci provozu firem. V dnešní době, kdy podniky čelí neustálým změnám a rostoucí konkurenci, hraje spolehlivý docházkový systém zásadní roli při udržení kontroly nad pracovním časem zaměstnanců a zvyšování efektivity provozu. Tato diplomová práce se zaměřila na vývoj inovativního řešení, které má potenciál zlepšit správu docházky a sledování pracovních výkonů, a tím usnadnit každodenní úkony.

Cílem této diplomové práce bylo navrhnout a implementovat aplikaci pro emulaci docházkového terminálu s rozšířenými funkcionalitami. Tato rozšíření zahrnují evidenci pracovních výkonů na konkrétních zakázkách. Aplikace musí splňovat potřeby zákazníka a pořízené záznamy jsou předávány docházkovému systému Cevis od firmy Ivar a.s.

Teoretická část práce byla věnována problematice vývoje softwaru a zahrnovala podrobné popsání metod softwarového inženýrství, požadavků na vývoj, využití modelovacího jazyka UML, programování v jazyce Delphi a implementaci identifikace prostřednictvím RFID technologie.

Praktická část práce započala sběrem a analýzou požadavků kladených na navrhovanou aplikaci. Získané požadavky byly následně použity k modelování UML diagramů: případů užití, diagramu tříd a stavového diagramu. Pro tvorbu těchto modelů byl využit software Draw.IO. Před samotným vývojem aplikace byly také vytvořeny *wireframy*, které byly založeny na definovaných požadavcích a UML modelech. Po dokončení návrhu aplikace v souladu s definovanými požadavky a modely, následovala fáze samotné implementace. Tato etapa vývoje probíhala v prostředí Delphi 10.4 Sydney, což byla zvolená platforma pro vývoj aplikace. Implementace byla provedena s důrazem na detaily a funkčnost, aby splňovala očekávání a potřeby uživatelů. Konečným krokem bylo provedeno testování.

Postup testování a doporučení pro budoucí vylepšení aplikace jsou popsány v kapitole výsledky a diskuze.

V souladu s předem stanoveným cílem, jak bylo rozvedeno v kapitole „Výsledky a diskuze“, bylo dosaženo očekávaných výsledků a cíl práce byl naplněn.

7 Seznam použitých zdrojů

1. SOMMERVILLE, Ian (2013) *Softwarové inženýrství*. Brno: Computer Press. ISBN 978-80-251-3826-7.
2. SUCHANOVÁ, Tereza (2021) *Vývoj software: Vzdělávání* [online]. [cit. 2022-10-18]. Dostupné z: https://www.suchanova.cz/vyvoj-software/
3. WIEGERS, Karl Eugene (2008) *Požadavky na software*. Brno: Computer Press. ISBN 978-80-251-1877-1.
4. STANDISHGROUP.COM (2020) *The Standish Group* [online]. [cit. 2022-10-20]. Dostupné z: https://www.standishgroup.com/
5. OLIVEIRA, Roberta (2022) *What You Need to Know to be Successful with Your Technology Project Implementation* [online]. [cit. 2022-10-20]. Dostupné z: https://www.nucli.io/blog/what-you-need-to-know-to-be-successful-with-your-technology-project-implementation
6. SZTURCOVÁ, Daniela (2013) *Požadavky na systém: Objektově orientované technologie, prezentace* [online]. [cit. 2022-10-25]. Dostupné z: http://gisak.vsb.cz/wikivyuka/images/9/9c/OotxPozadavky.pdf
7. ARLOW, Jim a NEUSTADT, Ilja (2011) *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press. ISBN 978-80-251-1503-9.
8. OMG.ORG (2017) ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5.1. In: *OMG Standarts Development Organization* [online].[cit.2022-10-28].Dostupné z: https://www.omg.org/spec/UML/2.5.1/About-UML
9. LYNCH, Allison (2022) What is UML | Unified Modeling Language. In: *EDRAW* [online]. [cit. 2022-11-02]. Dostupné z: https://www.edrawsoft.com/what-is-uml-diagram.html
10. KRAVAL, Ilja (2016) Kdy použít v Use Case Diagramu vztah Extend a kdy Include (část 1). In: *Server objektových technologií* [online]. [cit. 2022-11-02]. Dostupné z: https://www.objects.cz/wp/is_uml/kdy-pouzit-v-use-case-diagramu-vztah-extend-a-kdy-include/
11. RYDVAL, Slávek (2018) Kompozice a agregace. In: *Můj život s UML* [online]. [cit. 2022-11-05]. Dostupné z: http://blok.kurzy-uml.cz/kompozice-a-agregace

12. Wayback Machine (2020) In: *Delphi* [online] [cit. 2022-11-06]. Dostupné z: https://delphi.embarcadero.com/
13. THORPE, Danny (2020) WHY IT'S CALLED DELPHI. In: *Delphi* [online]. [cit. 2022-11-06]. Dostupné z: https://delphi.embarcadero.com/why-its-called-delphi/
14. TIOBE.COM (2022) TIOBE Index for November 2022: November Headline: Rust keeps its top 20 position. In: *TIOBE* [online]. [cit. 2022-11-10]. Dostupné z: https://www.tiobe.com/tiobe-index/
15. PYPL.GITHUB.IO (2022) PYPL PopularitY of Programming Language. In: *PYPL Index* [online]. [cit. 2022-11-24]. Dostupné z: https://pypl.github.io/PYPL.html
16. VIOLINO, BOB (2005) The History of RFID Technology. In: *RFID JOURNAL* [online]. [cit. 2022-11-12]. Dostupné z: https://www.rfidjournal.com/the-history-of-rfid-technology
17. FIALOVÁ, Eva (2016) *Bezkontaktní čipy a ochrana soukromí*. Praha: Leges. Praktik (Leges). ISBN 978-80-7502-150-2.
18. COSMOTRON.COM (2019) Technologie: Standardy RFID. In: *Cosmotron* [online]. [cit.2022-11-15]. Dostupné z: https://www.cosmotron.cz/produkty/rfid/technologie/
19. LIGHTPIONEER.COM (2017) Worldwide UHF Frequency Allocation Map by Country. In: *LightPioneer* [online]. [cit. 2022-11-15]. Dostupné z: http://www.lightpioneer.com/worldwide-uhf-frequency-allocation-map-by-country/
20. KROENKE, David a AUER, David (2015) J. *Databáze*. Brno: Computer Press. ISBN 9788025143520.
21. BRUCKNER, Tomáš (2012) *Tvorba informačních systémů: principy, metodiky, architektury*. Management v informační společnosti. Praha: Grada. ISBN 978-80-247-4153-6.
22. KETTERMAN, Shane (2023) Form and Function: A Guide to the Top Wireframe Tools. In: *UX Design* [online]. [cit. 2023-11-26]. Dostupné z: https://www.toptal.com/designers/ui/top-wireframe-tools-guide

8 Seznam obrázků a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

UML – Unified Modeling Languege
XML – Extensible Markup Language
RFID – Radio-Frequency Identification
Wireframe – Drátěný model
IEEE – Institute of Electrical and Electronics Engineers
SQL – Structured Query Language
UHF – Ultra Hight Frequency
LF – Low Frequency
HF – Hight Frequency
IDS – Integrated Data Store
IMS – Information management Systém
eMMC - Embedded MultiMediaCard
SDLC - Synchronous Data Link Control