FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Informační systém pro podporu provozu jídelny
Student:	Bc. Tomáš Kello
Vedoucí:	Ing. Peter Macejko
Studijní program:	Informatika
Studijní obor:	Návrh a programování vestavných systémů
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Vytvořte informační systém pro podporu výdeje a objednávání jídel ve stravovacím zařízení. Systém by měl podporovat zejména následující operace:
- identifikaci strávníka pomocí čipové karty,
- rezervace jídla strávníkem,
- vyzvednutí jídla strávníkem,
- informování výdejny o vydávaných jídlech,
- administrační rozhraní pro správu klientů, objednávek a jídelníčků.

Součástí práce je nejen návrh a implementace SW částí systému, ale i HW částí (čtečka karet, zobrazovací monitor, …). Zvažte využití běžně dostupných jednočipových počítačů (Raspberry Pi, …).

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 28. února 2018

Poďakovanie

Veľmi rád by som sa poďakoval vedúcemu práce Ing. Petrovi Macejkovi za jeho odborné vedenie, pomoc a cenné rady, ktoré mi poskytol pri vypracovaní magisterskej práce

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 2. mája 2018 .....

0.0.1 Odkaz na túto prácu

Kello, Tomáš. *Informační systém pro podporu provozu jídelny.* Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práca rieši informačný systém vo veľkokapacitných jedálňach. Obsahujúci databázový, aplikačný server, aplikáciu na správu systému a tri druhy terminálov: informačný, objednávací a výdajný. Na termináli si klient môže zobraziť svoje objednávky, vytvoriť nové alebo prevziať si jedlo na základe jeho funkcie. Databáza obsahuje zoznam užívateľov, ich objednávky, bezkontaktné karty a jedálny lístok na každý deň. Terminál označuje autonómne zariadenie vybavené dotykovým displejom a čítačkou kariet. Je vytvorený univerzálne a môže ponúknuť ktorúkoľvek dostupnú funkciu. Tá je pre každý terminál nastavená v databáze. Prínosom tejto práce je zameraný na zníženie nákladov a zjednodušenie výdaju jedál.

Klíčová slova jedáleň, výdaj jedál, objednávanie, MySQL, Raspberry Pi, bezkontaktná karta

Abstract

Program implementing information system for school or company canteen. Include database, application server, application for system management and three types of terminals: information, ordering and serving. With this system user can show his orders, create new or get his orders. Database contain list of users with their orders, cards, menu for every day and meals, which are possible to cook in canteen. Terminal means autonomous device with touch display and card reader. It can act as any device depending on configuration in database. System is focused to be used in high school and companies with own canteen for employers to reduce cost and simplify meal serving.

Keywords canteen, meal serving, ordering, MySQL, Raspberry Pi, proxy card

Obsah

Zoznam obrázkov

Úvod

Každý zamestnávateľ je povinný prispievať na stravu zamestnancom podľa §152 Zákonníka práce [1]. Preto v menších firmách dostávajú zamestnanci k výplate stravovacie lístky. Pre väčšie firmy je výhodnejšie zabezpečiť vlastné stravovacie zariadenia. Rovnako tak školy aj nemocnice používajú vlastné stravovanie. Spôsob objednávania a výdaju sa v prevádzkach líšia. Najjednoduchšie, kde jedáleň neponúka možnosť výberu jedla môže použiť kupóny s dátumom na výdaj jedla. Ďalšou možnosťou je platba za jedlo na mieste. V takom prípade sa dá využiť pokladňa s okamžitou platbou za jedlo. Má to nevýhodu predpovedania, koľko porcii je potrebné navariť a ktoré jedlo je preferované stravníkmi viacej. Následkom toho stravník, ktorý príde neskôr nedostane svoje obľúbené jedlo, ale musí si vybrať z toho čo ostalo. Rovnako tak jedáleň nevie odhadnúť presné množstvo porcii, ktoré treba navariť a vzniká tak plytvanie a vyhadzovanie potravín. Treťou možnosťou je použitie informačného systému na objednávanie a identifikáciu stravníkov pri výdaji. Ponúka to väčšie možnosti výberu jedál, možnosť pre jedáleň prispôsobiť varenie podľa počtu objednávok a tiež bezkontaktné platby. To všetko sa tiež zahŕňa do služieb a benefitov, ktoré daná spoločnosť môže ponúkať študentom, zamestnancom, užívateľom či stravníkom.

Naskytla sa mi možnosť vytvoriť informačný systém pre stravovacie zariadenia s tým, že by obsahoval univerzálny zobrazovací terminál. Ten by bol vyvíjaný v spolupráci s Brnenskou spoločnosťou, ktorá na trhu v stravovaní má viacročné skúsenosti a používa zastaralý terminál, ktorý by chceli nahradiť novým. Táto možnosť, že moje zariadenie nájde uplatnenie v praxi ma veľmi zaujalo. Keďže sa jedná u úplne nový terminál, nie je závislý na spätnej kompatibilite s firmou.

Pri vývoji nového systému a terminálu je veľa možností realizácie a práca bude náročná na analýzu. Budú opísané viaceré možnosti ako systém a terminál vytvoriť. Vo väčšine prípadov som informácie čerpal z vlastných skúseností, školy a rozširoval som ich čítaním návodov k zariadeniam, ktoré

budú spomenuté v práci.

Výsledkom je vytvoriť informačný systém pre veľkokapacitné jedálne, kde obsluha má možnosť nastaviť jedálny lístok pre jednotlivé dni v týždni a s identifikáciou výdajného miesta, kde sa jedlo varí v prípade, že ich prevádzka obsahuje viacej. Následne po vytvorení jedálnička si zákazníci môžu dané jedlá objednať, prezerať už existujúce objednávky a prevziať si objednávku v priestore výdaju jedál. Počas práce so systémom je klient identifikovaný pomocou bezkontaktných kariet. Pre vedúcu jedálne je potrebné vytvoriť aplikáciu na správu systému, varených jedál a objednávok.

Kapitola 1

Podobné systémy

Jedálne základných a stredných škôl zvyčajne neponúkajú výber jedál a väčšinou fungujú na papierových lístkoch, na ktoré sa im vydáva strava. Avšak postupne sa tieto papierové riešenia začínajú nahradzovať. V prípade väčších firiem môžu mať obednajšiu prestávku, kde v jeden čas vydajú jedlo pre všetkých. V tomto prípade by použitie informačného systému ponúklo možnosť si jedlo vybrať. A podľa zamestnancov navariť správne množstvo z jednotlivých alternatív.

WEGA LH Slovenská firma zabezpečujúca komplexné informačné systémy firiem. Obsahuje dochádzkové a stravovacie systémy s podporou biometrického rozpoznávania identity alebo s využitím bezkontaktných kariet. Podporujú niekoľko typov základných ale aj bezpečnostných šifrovaných kariet Mi-Fare. Firma vznikla v roku 1999, čo znamená že majú za sebou niekoľko rokov skúseností. Medzi ich klientov patria stredné a základné školy, ktoré využívajú len stravovaciu časť systému a z funkčného hľadiska ich riešenie spĺňa všetky potrebné náležitosti.

Jediný nedostatok je koncové zariadenie, zobrazené na obrázku 1.1. Dost-

Obr. 1.1: Bezkontaktný terminal firmy WEGA LH

áva sa do styku so stravníkmi, takže by malo vyzerať reprezentatívne a ponúkať interakciu. Obsahuje síce displej, ktorý dokáže zobrazovať text, ale neponúka interakciu. Tá sa v aktuálnej dobe očakáva spolu s väčším farebným a dotykovým displejom.

1.1 Terminál

Hlavnou funkciou zariadenia je zobrazovanie informácii s možnosťou interaktívnosti. Hlavne v podobe čítania bezkontaktných kariet. Ak mám nájsť podobné zariadenie, musí obsahovať obrazovku, v ideálnom prípade dotykovú a komunikačné periférie ako napríklad USB, I2C, UART. Ktoré sa dajú využiť na pripojenie čítacej hlavy proxy kariet a internetového pripojenia na komunikáciu so serverom. Výhodou by boli ďalšie možnosti ako zvuková signalizácia, svetelný výstup, napríklad pripojenie LED diódy alebo pripojenie klávesnice či iné hardvérové tlačidlá.

1.1.1 Ki-Wi

Na stránke www.ki-wi.cz ponúkajú informačné kiosky. Ich riešenie obsahuje webový server a následne sa jednotlivé zariadenia pripájajú a zobrazujú webovú stránku, ktorá obsahuje interaktívny obsah. Na týchto zariadeniach beží operačný systém Windows, ktorý je upravený aby nebolo možné používať vstavané funkcie systému. Po zapnutí zariadenia sa automaticky spustí webový prehliadač so stránkou webového servera v móde na celú obrazovku. Takže nezobrazuje systémové lišty ani adresu stránky.

V ich ponuke je niekoľko druhou informačných panelov. Ich funkčnosť je všade rovnaká a rozdiel je vo veľkosti displeja a uchytenia v prevádzke. Najčastejšie sa využíva zariadenie s vlastným stojanom alebo s montážou na stenu, ktorá by bola výhodnejšia v prípade použitia v stravovacích priestorov. Príklad takého zariadenia je na obrázku 1.2.

Veľkou výhodou riešenia je jednoduchosť implementácie funkčnosti do kioskov. Keďže využili štandardný spôsob komunikácie pomocou internetu a zobrazujú HTML obsah. Ponúka to vytvorenie niekoľkých obrazoviek s prechodom medzi nimi, možnosť reagovať na stlačenie tlačidlá, vytvorenie štruktúry stránky s následnou úpravou obsahu podľa prihláseného užívateľa a ďalšie funkcie spojené s možnosťou integrovať JavaScript, PHP či CSS. Dajú sa použiť na vytvorenie dizajnu s animáciami a interaktívnym obsahom bez nutnosti komunikácie so serverom. Nevýhodou je možnosť pripojenia čítačky kariet. Počítače štandardne neobsahujú komunikačné periférie ako I2C alebo UART, preto by som preferoval USB. Kde po pripojení USB čítačky karty by bolo potrebné implementovať na zariadení odchyt informácii a následné posielanie čisla priloženej karty na server.

Obr. 1.2: ponuka od firmy Ki-Wi

Obr. 1.3: ponuka od firmy AV factory

1.1.2 AV factory

Ďalšie podobné riešenie na stránke eshop.avfactory.cz obsahuje podobný informačný stojan ako z firmy Ki-wi. Avšak toto riešenie neobsahuje centralizovaný server. Firma ponúka len samotné informačné zariadenie, ktoré obsahuje počítač s operačným systémom od firmy Microsoft a možnosťou pripojenia na internet. Toto riešenie je len čiastočné a bolo by treba dorobiť systém komunikácie, server a tiež aplikáciu pre daný kiosk. Takže uvádzaná cena 89 298,00 Kč je len za samotný kiosk bez ďalších softvérových riešení. V prípade použitia by bol výhodou pekný dizajn, ktorý je zobrazený na obrázku 1.3 spolu so stojanom. Ten umožňuje umiestnenie kiosku na podlahu vo vnútorných priestoroch.

1.2 Zhrnutie

U všetkých prezentovaných riešeniach som našiel nesplnené požiadavky, ktoré by som chcel mojím systém implementovať. A navyše by som systém chcel rozšíriť o ďalšie, ktoré získam na základe spätnej väzby od zákazníkov alebo sa objavia počas vývoja.

Kapitola 2

Analýza

Pri vývoji nového zariadenia sa stretávam nie len s programovaním a vytvorením softvérovej časti. Ale je potrebné nájsť aj hardvér, ktorý budem používať a poskytne mi možnosti na splnenie požiadaviek. V analýze sa pokúsim porovnať a zdôvodniť, prečo som sa rozhodol pre jednotlivé komponenty a riešenia.

Analýza je rozdelená na úvod, kde sú opísané požiadavky a očakávania od troch časti systému a následne ich detailný rozbor. Prvá sa bude venovať výberu serverov, obsahujúca databázový a aplikačný server. Druhá časť bude obsiahlejšia a venovaná zobrazovaciemu zariadeniu. To spája výber hardvérových komponent a softvérových riešení ako je výber frameworku, programovacieho jazyka a operačného systému. Na záver ešte pár slov k obsluhe systému a správe aplikácii, užívateľov a ďalších dát, ktoré sa v aplikácii používajú.

2.1 Požiadavky a očakávania

Komplexný systém rozdelím do menších častí:

- Databázový server – uschováva všetky nastavenia, zoznam jedál a klientov
- Správa systému – určená pre vedúcich jedálne. Umožňuje správu užívateľov, bezkontaktných kariet, jedál, jedálničkov a objednávok
- Terminál – autonómne zariadenie schopné interakcie so zákazníkom, zvukovú signalizáciu, čítanie kariet, zobrazovanie informácii a komunikáciu s aplikačným serverom
- Aplikačný server – obsahuje obslužné podprogramy, ktoré sa starajú o beh terminálu a komunikáciu s databázou

  - Objednávacie zariadenie - v prípade že klient má dostatok financií, umožnuje mu objednať jedlo z ponuky jedálne
  - Informačné zariadenie - klientovi zobrazí všetky jeho objednávky, ktoré vytvoril a neboli zatiaľ vydané
  - Výdajné zariadenie - zobrazenie objednaného jedla pre obsluhu v kuchyni a následné označenie jedla za vydané v informačnom systéme. Podľa informácie na displeji obsluha zákazníkovi vydá jeho objednávku

Nedostatky aktuálneho terminálu opísaného v kapitole 2.2, ktoré sa pokúsim zlepšiť:

- Obmedzenosť zobrazovacej plochy - Hlavný displej dokáže zobraziť len 4 znaky, reprezentujúce druhy jedál, ktoré má klient objednané
- Ovládanie zariadenia - funkcie zahájiť a ukončiť výdaj sa vykonáva priložením systémovej karty s danou funkciou. Čo vyžaduje obsluhu aby disponovala niekoľkými kartami a prikladala ich zo strany klienta
- Dizajn zariadenia je zastaralý
- Rozšíriť portfólio funkcii pre obsluhu
  - zobraziť vydané jedlá
  - vrátiť už vydané jedlo a nastaviť ho v systéme za nevydané
  - zahájiť výdaj len určitých jedál, ktoré navarili v danej jedálni
  - ohlásiť vzdialenú pomoc priamo zo zariadenia, bez potreby volať servis
- Zabezpečiť jednoduchú konfiguráciu a servis
- Znížiť náklady na výrobu jedného terminálu

2.2 Aktuálne riešenie používaného terminálu

Aktuálne sa používa 7 segmentový obojstranný displej rozšírený o čítačku kariet s pripojením do lokálnej siete cez internetový protokol TCP/IP.Terminál je zobrazený na obrázku 2.1. S využitím protokolu posiela zariadenie serveru číslo priloženej karty a ten odpovie informáciou, ktorá sa má na displeji zobraziť.

Obr. 2.1: Výdajné miesto v prevádzke

2.3 Server

Táto časť bude pozostávať z databázového a aplikačného servera. V prípade databázového servera môžem použiť riešenie od firmy Microsoft (MS SQL) alebo Oracle (Oracle SQL). Oba sú komplexné a ponúkajú veľké množstvo funkcii spojené s ukladaním dát, ich analýzou, sledovanie záťaže a rýchlosti spracovania požiadaviek. Ich nevýhodou je cena licencie. Z ponuky voľne šíriteľných riešení je MySql databáza. Síce neponúka niektoré pokročilé funkcie, ale v prípade našej aplikácie by bola postačujúca, keďže systém bude používať len základné funkcie ako je: výber, vloženie a úprava dát.

Požiadavky na aplikačný server sú získanie dát z databázy a následná komunikácia s koncovými terminálmi. Na komunikáciu využijem sieťový protokol TCP/IP. Dôležité pri implementácii komunikácie je asynchrónnosť. Je vylúčené aby server čakal na odpoveď od zariadenia po odoslaní správy. V prevádzke môžu byť aj desiatky terminálov pripojených na jeden server a preto je potrebné aby aplikácia bola viacvláknová. Na základe požiadavok a vzhľadom na použitie by som odporúčal použitie programovací jazyk C++, C# alebo Javu. Výsledný program, respektíve systémovú službu bude potrebné nastaviť, aby sa spúšťala spoločne s operačným systémom a v prípade pádu bola automaticky reštartovaná. Tým zabezpečila nepretržitú funkčnosť.

Databázový aj aplikačný server je možné umiestniť do jedného fyzického zariadenia. V takom prípade potrebujem o niečo výkonnejšie zariadenie ale ušetrím na počte serverov. Zároveň s tým spojenou správou systému, ktorá sa zníži na jedno serverové zariadenie. Ohľadom výberu operačného systému je možné použiť Windows Server, ten podporuje všetky tri druhy databáz. Ďalšou možnosťou je vybrať si niektorý z Linuxových systémov. Tam je ponuka širšia a záleží na využití a osobných preferenciách. Ponúkajú možnosť podpory pre MySQL a Oracle SQL.

2.4 Zobrazovacie zariadenie

Výsledkom zobrazovacieho zariadenia bude vytvorenie terminála, ktorý bude obsahovať krabičku s displej na jednej alebo oboch stranách. Obojstranný terminál má uplatnenie v prípade výdaja jedál, kde jeden monitor je používaný obsluhou s dodatočnými informáciami o jedle a možnosťou ovládať výdaj. A druhá strana je určená pre klienta spolu s čítačkou kariet. Jednostranný displej má uplatnenie pri objednávaní a zobrazení informácii o objednaných jedlách užívateľa.

2.4.1 Hardvér

Pri výbere hardvérových komponent sa vždy budem pozerať na niekoľko dôležitých aspektov, ktoré budem zohľadňovať.

- Pripojenie, softvérová podpora a dostupnosť dokumentácie
- Dizajn
- Kvalita prevedenia a odolnosť voči teplu a pare
- Cena
- Možnosti uchytenia v krabičke
- Dostupnosť v budúcnosti

Jednou z požiadaviek je odolnosť voči teplu a pare. Očakáva sa použitie v jedálňach, kde zvyčajne jeho umiestnenie bude v blízkosti teplých jedál. Z toho vyplýva, že zariadenie by malo byť čo najviac odolné voči teplu a pare vychádzajúcej z hrncov a navareného jedla.

2.4.2 Výpočtová jednotka

Ako bolo spomenuté terminál musí umožňovať komunikáciu so serverom cez TCP/IP. V tomto prípade je najlepšie použiť výpočtovú jednotku so vstavaným sieťovým konektorom. Jednou z možností je kancelársky počítač s operačným systémom. Ak by som použil operačný systém Windows. Je potrebné platiť licenciu na každom počítači v termináli a v systéme sa počíta s niekoľkými terminálmi a tým náklady na prevádzku by sa násobili ich počtom. Existuje alternatíva v podobe voľne dostupného operačného systém Linux. Taktiež okrem ceny je dôležitý aj rozmer. V prípade použitia kancelárskeho počítača by jeho umiestnenie muselo byť vzdialenejšie od displeja, keďže jeho rozmery sú príliš veľké aby bol umiestnený vo vzduchu pri displeji. Umiestniť sa môže na zem a následne mať displej pripojený cez HDMI, VGA alebo podobný obrazový kábel. Okrem veľkých rozmerov stolového počítača je zvyčajne aj jeho cena vyššia.

Obr. 2.2: Najmenšia verzia Raspberry Pi Zero

Mikroprocesor môže obsahovať sieťový konektor, ktorý záleží na danom výrobku. Napríklad Arduino nemá natívnu podporu sieťového pripojenia. Ale napríklad firma mBed ponúka mikroprocesor s doskou, ktorý už sieťovým pripojením disponuje. S ním som sa dostal do kontaktu počas výmenného pobytu v Belgicku cez program Erasmus. Používa architektúru ARM [2] a Real-Time operačný systém Arm Mbed OS v aktuálnej verzii v5.8 [3]. Taktiež podporuje pripojenie displejov, ale len základných riadkových segmentových čo v našom prípade nie je dostačujúce. Jeho použitie je skôr smerované na jednoduchšie systémy a preto budem potrebovať počítač, na ktorom bude bežať operačný systém na všeobecné použitie (non-real time operating system). Ten má dlhšiu odozvu na spracovanie prerušenia ale ponúka širšiu podporu a tým mi zjednoduší vývoj pri pripojení dotykovej obrazovky alebo klávesnice.

Mikropočítač Používa operačný systém ale jeho výkon a rozmery sú výrazne nižšie od stolových počítačov. Jedna zo známych firiem na trhu s mikropočítačmi, ktorých výkon nepresahuje mobilný telefón ale zato disponuje s veľkým množstvom periférii je Raspberry Pi. Pôvodne smerovaný na vzdelávacie účely. Avšak vďaka dobrej cene, výkonu, spoľahlivosti a veľkosti sa aktuálne používa vo viacerých nepretržitých prevádzkach 24 hodín denne 7 dní v týždni aj napriek tomu, že na to nebol smerovaný. Firma ponúka niekoľko verzii produktu. Základná verzia v približnej veľkosti kreditnej karty a menšiu alternatívu s modelovým názvom ZERO zobrazené na obrázku 2.2. Avšak toto zariadenie nemá DSI výstup obrazu a vstavané GPIO konektory. Preto sa sústredím na základnú verziu zobrazenú na obrázku 2.3.

Jej procesor je postavený na ARM architektúre, preto sa na zariadení nedá spustiť operačný systém pre architektúry x86 alebo x64. Oficiálny operačný systém pre Raspberry Pi je Raspbian, ktorý je distribúciou Debianu modifikovanou práve pre daný mikropočítač. Ďalšie možnosti ako Windows IoT, Ubuntu Mate, Android, RISC OS, nie sú priamo vyvíjané pre tento mikropočítač a úpravy nie sú vždy stabilné. Ja som skúšal Windows IoT. Tento

Obr. 2.3: Raspberry Pi verzie 2

operačný systém počas testovania nebol dostatočne stabilný na použitie. Po prerušení napájania malo Raspberry Pi problémy pri znovu spustení. Taktiež s dokumentáciou a online podporou to zatiaľ nie je dostatočné. Ako ďalší systém som vyskúšal Raspbian. Ma veľmi dobrú užívateľskú podporu s veľa ukážkovými programami. Je priamo podporovaný autorskou firmou a všetky novinky a ukážkové programy z ich stránky sú spúšťané na tomto operačnom systéme.

Pri výbere verzie je viac možností. Aktuálne najnovšia verzia nesie označenie 3. Od jeho predchodcu verzie 2 je rozdiel hlavne vo výkone. Zvyšné periférie ako USB, LAN, pamäťová karta, zvukové a obrazové konektory sú rovnaké, dokonca aj rozmiestnenie na doske je identické. Preto ďalej budem používať len označenie Raspberry Pi, čím označím oba modely 2 aj 3. Od verzie 1, ktorá bola zverejnená v roku 2011, bola doska redizajnovaná a rozšírená o ďalšie periférie.

Napriek tomu, že Raspberry Pi má dva obrazové výstupy, jeden digitálny HDMI a druhý DSI nedokáže používať oba súčasne. Preto pre obojstranný displej by sa museli použiť dva mikropočítače, pre každý displej vlastný.

Ďalšie známe zariadenia sú väčšinou len alternatívy. Väčšinou sú kombináciou viacerých periférii, väčšieho výkonu alebo nižšej ceny. Medzi konkurenciu schopné patria napríklad: Orange Pi, Banan Pi, ASUS Tinker Board, Beagleboard a ďalšie.

2.4.3 Softvér

2.4.3.1 Web/Natívna aplikácia/Vzdialená plocha

Pri použití webového servera a internetového prehliadača na strane zariadenia, ako to používa firma Ki-Wi, vzniká problém s inicializačným nastavením a vzdialenej správe. Web aplikácia by musela byť ako klient a pripájať sa na server, kde je potrebné definovať jeho adresu na každom zariadení osobitne. Ďalšie problémy, ktoré očakávam pri vývoji je s implementáciou čítačky kariet, zvukovej signalizácie a rýchlosti odozvy na zmenu stavu v databáze či propagáciu stavu z jedného terminála na druhý. Na internete som našiel projekty,

kde tieto problémy riešia. Ale nie je ich veľa a sú závislé na verzii webového prehliadača a operačného systému. Veľkú výhodu však má v prípade novej verzie. Kde stačí aktualizovať webovú stránku na serveri a nie je potrebný zásah do terminálov.

Ďalšou možnosťou je spustenie virtuálnej plochy na serveri. To znamená že by celý systém fungoval len na serveri, tým by sa odľahčila výpočtová záťaž terminála, ktorý by sa podľa nastavených parametrov pripojil na server cez vzdialenú plochu a zobrazil, čo server zrovna ponúka. Nevýhodou je implementácia serverovej časti na oznamovanie, čo sa má zobraziť a druhá implementácia programu, ktorý by bežal na virtuálnej ploche. Výhodou by bola centralizovanosť. Kde všetko sa odohráva na strane servera a tým jednoduchá pomoc v prípade poruchy. Rovnako tak aktualizácia softvéru by bola centralizovaná. Problémy v tomto riešení môžu vzniknúť na strane klienta. V prípade webového rozhrania by sa strata spojenia so serverom vedela rozumne zobraziť. Ale v prípade vzdialenej plochy by to mohol byť problém, keďže programy na pripojenie sa na vzdialenú plochu majú správanie v prípade straty spojenia naprogramované a priebeh znovu-pripájania by sa na termináli nedal modifikovať. Jeho odstránenie, prípadne vytvorenie vlastnej aplikácie na pripojenie sa na vzdialenú plochu by znamenalo vytvorenie ďalšieho programu. Tým by sa celkové riešenie komplikovalo a vznikali by problémy pri hľadaní chýb. Okrem toho je tu rovnaký problém s nastavením, kde každé zariadenie vyžaduje vlastnú konfiguráciu.

Pri natívnej aplikácii mám najväčšie možnosti pri implementácii. Viem vyriešiť problém so stratou spojenia, čítačku kariet, zvukovú signalizáciu prípadne ďalšie funkcie, ktoré by som chcel pridať neskôr. Taktiež viem implementovať systémové správy, ktoré by sa zobrazili na základe druhu poruchy. Tým by užívatelia systému vedeli poruchu opraviť alebo aspoň kontaktovať správcu s presnou príčinou chyby. Avšak každá výhoda niečo stojí. Toto riešenie je náročné na čas a je potrebné naprogramovať aplikáciu so všetkou funkcionalitou. Pri použití natívnej aplikácie uloženej v pamäti zariadenia je tiež problém s jeho aktualizáciami. Pre každý terminál je potrebné osobitne nahrať novú verziu. V prípade použitia linuxového operačného systému je možnosť pripojiť sa na zariadenie vzdialene a nahradiť starú aplikáciu novou pomocou aplikačného servera.

2.4.3.2 Programovací jazyk

Ponuka programovacích jazykov je veľmi široká. Výber mi môže trochu ovplyvniť operačný systém. Avšak jedno z možností, ktorý je tiež prenositeľný medzi všetkými operačnými systémami sú C++ a skriptovací jazyk Python. Keďže ja so skriptovacím jazykom nemám veľké skúsenosti, uprednostnil by som C++. Ten v kombinácii s QT vývojovým prostredím, ktoré rozširuje samotný jazyk o vlastné funkcie a ponúka grafický návrh aplikácie vyzerá dostačujúco. U grafického návrhu využíva X Window so systémovou knižnicou XLib. Vďaka

Obr. 2.4: Raspberry Pi zadný pohľad

nej vykresľuje na obrazovke jednotlivé objekty aplikácie ako tlačidla, texty, tabuľky, obrázky a rôzne geometrické tvary.

2.4.4 Displej a dotyková obrazovka

Aktuálne rozmery zobrazovacieho displeja zodpovedajú približne displeju s uhlopriečkou 7 palcov. Dostupné je množstvo displejov na notebooky, ktoré väčšinou nie sú dotykové. Ďalej displeje na tablety, ktorých použite by bolo výhodnejšie, keďže väčšinou obsahujú aj dotykový panel. Tieto displeje sú rozmerovo tenké s dobrým dizajnom a kvalitou, avšak ich pripojenie je zvyčajne cez 30 alebo 40 pinový flexi konektor. K tomu je potrebný adaptér na HDMI alebo VGA. V prípade, že by displej obsahoval aj dotykovú plochu, potom aj daný adaptér by musel dotyky podporovať. Prípadne by sa mohol použiť bezdotykový displej s adaptérom a následne naň použila dotyková fólia s pripojením do USB. Avšak tu vznikajú problémy s kalibráciou, správnym nalepením a ďalšie neočakávané problémy pri snahe spojiť displej a dotykovú plochu od rôznych výrobcov.

Z toho sa javí lepšie použiť priamo vstavaný dotykový displej. V prípade, že by som použil zariadenie Raspberry Pi, je možné použiť 7 palcový dotykový displej, ktorý sa dá vidieť na obrázku 2.4. Tento displej navrhla a predáva rovnaká spoločnosť, preto aj jeho kompatibilita je zaručená. Ďalšou výhodou je uchytenie. Samotný displej obsahuje závity po kraji na uchytenie displeja ale aj ďalšie závity na uchytenie minipočítača Raspberry Pi na jeho zadnej strane v presných rozmeroch spolu so skrutkami v balení. Na záver by som ešte dodal, že cena tohto displeja sa pohybuje okolo 80 eur (2000 českých korún), čo je výrazne lacnejšie ako displeje z notebookov a tabletov.

V kombinácii mikropočítača Raspberry Pi, voľne dostupného operačného systému Raspbian a 7 palcového dotykového displeja, vyrobeného priamo na podporu tohto zariadenia vzniká silné spojenie, ktorému ťažko hľadať konkurenciu. U mikropočítača nájdem niekoľko ďalších alternatív. Avšak žiaden z nich mi neponúka pripojenie k tomuto displeju. A keďže cena Raspberry Pi nie je až tak vysoká a z funkčného hľadiska ponúka všetko čo potrebujem, je možné práve tieto dva zariadenia spojiť a použiť v budúcom termináli.

Obr. 2.5: Pripojenie zvukovej signalizácie

2.4.5 Zvuková signalizácia

Ďalšou požiadavkou pre výsledné zariadenie, je zvuková signalizácia. Väčšina mikropočítačov obsahuje zvukový výstup, ktorý by sa dal použiť. Avšak jeho hlasitosť je dosť obmedzená a reproduktory sú náročné na priestor. Preto je výhodnejšie zvoliť digitálne výstupy, ktoré po pripojení a aktivovaní výstupného pin-u na bzučiak generujú dostatočne hlasitý zvuk s malými rozmermi.

Ako je vidieť na obrázku 2.5 k pripojeniu bzučiaka stačia dva káble (v mojom prípade žltý a modrý), čo tiež pomáha k jednoduchej inštalácii zariadenia. A vďaka jednoduchosti zhotovenia sa dá očakávať bezchybný chod s nízkou frekvenciou porúch.

2.4.6 Čítačka kariet

Poslednou dôležitou požiadavkou je pripojenie čítačky kariet. Ak použijeme operačný systém, ktorý ponúka podporu USB zariadení, bude najjednoduchším riešením pripojenie USB čítacej hlavy, ktorá sa po pripojení tvári ako klávesnica. Po priložení karty simuluje stlačenia klávesov a do počítača sa odošle štartovací znak, číslo karty a špeciálny koncový znak.

Väčšina mikropočítačov okrem USB, ponúkajú viaceré možnosti komunikácie ako je UART, SPI, I2C. V prípade kancelárskeho počítača je možné pripojiť prevodník z USB do potrebnej zbernice. Napríklad vďaka UART môžem pripojiť lacnejšiu čítaciu hlavu, ktorá podporuje RFID karty frekvencie 125 KHz zobrazenej na obrázku 2.6. Rovnako ako USB-čková čítacia hlava, aj táto v prenose zaobalí čísla karty štartovacím a koncovým špeciálnym znakom. K funkčnosti zariadenia čítacej karty potrebuje napájanie a dva jednosmerné komunikačné spojenia so zariadením na prenos informácii oboma smermi podľa štandardu UART [4]. Ale keďže počítač nepotrebuje čítacej hlave nič posielať. Stačí zabezpečiť len jedno spojenie v smere z čítacej hlavy do mikropočítača. Následne po spustení aplikácie zariadenie čaká na priloženie karty a zahájenie prenosu zo strany čítacej hlavy. Po prijatí spracuje správu a zistí číslo karty, ktoré obratom preposiela do servera. Až ten sa rozhoduje, čo sa stane po tom ako systém zistil priloženie karty.

Obr. 2.6: Čítacia hlava s pripojením UART

2.4.7 Napájanie

V prípade použitia ktoréhokoľvek počítača budem potrebovať pripojenie k serveru pomocou TCP/IP. Z toho vyplýva minimálne jeden internetový kábel a napájanie. Ohľadom napájania budem potrebovať zásobovať elektrickou energiou displej a počítač. U stolových počítačov by to bolo 230 voltov striedavých, u mikropočítačov väčšinou USB nabíjačka. Tá v sebe nesie napätie 5V. V prípade monitora je to 230V a u displejov záleží na výrobcovi. Väčšinou 5 alebo 12V jednosmerného napätia.

V prípade obojstranného displeja budem používať dva identické zariadenia. Ku každému displeju jedno. To vyžaduje dva napájania a dva internetové spojenia. Pre zjednodušenie inštalácie môžem spojiť dva internetové káble do jedného a následne v krabičke ich rozdeliť pomocou internetového rozdeľovača podobného tomu na obrázku 2.7. Rozdeľovače sa vždy používajú dva. Kde po výstupe zo smerovača sa dva internetové káble najskôr spoja do jedného a následne v krabičke sa spätne rozdelia do dvoch. Tento internetový kábel obsahuje 8 žíl. Kde v prípade prenosovej rýchlosti 100Mbit v skutočnosti využíva len polovicu z nich. Vďaka tomu dokáže rozdeľovač spojiť dva prenosy do jedného a tým využívať všetkých osem žíl kábla.

U napájania je situácia jednoduchšia. Pri použití stolového počítača na 230V môžem použiť elektrickú rozdvojku a napájať jedným káblom počítač aj displej. V prípade mikropočítača je potrebné použiť dostatočne výkonný transformátor na 5V jednosmerných. Následne zabezpečiť dostatočne hrubý kábel v priereze medzi transformátorom a mikropočítačom, aby neprehorel v prevádzke z dôvodu vysokého prúdu. A následne použiť rozdeľovač micro-USB kábla, ktorý na jednej strane má jeden konektor typu „samica“ a na druhej strane sa nachádzajú dva konektory typu „samec“ podobný tomu na obrázku

Obr. 2.7: Rozdeľovač sieťového pripojenia

Obr. 2.8: Rozdeľovač micro USB

2.8. To mi umožní pomocou jedného napájacieho zdroja dodávať energiu do dvoch a viac zariadení pri použití viacerých rozdeľovačov. To využijem ak budem potrebovať dva mikropočítače s dvomi displejmi, čo vyžaduje až 4 napájania.

2.4.8 PoE

Pri použití mikropočítača, ktorý funguje na jednosmernom napätí s nízkym odberom je možnosť použitia PoE. To znamená prenos elektrického prúdu a sieťového prenosu v jednom 8 žilovom UTP alebo STP kábli. Opisuje to norma

IEEE802,3af/at, ktorá je dostupná online na [5].

K jeho fungovaniu neobmedzuje samotný prenos dát, ktorý využíva vysoké frekvencie a v prípade napájania je to jednosmerný prúd. Vďaka tomu sa navzájom nerušia a môžu sa využívať spoločne. Samozrejme pri prenose nastávajú poklesy napätia, preto podľa štandardu je prevádzkové napätie v rozmedzí 44 do 57 voltov s prúdom okolo 600mA a po konvertovaní na 5V je možné získať až 25 wattov.

2.4.9 Krabička

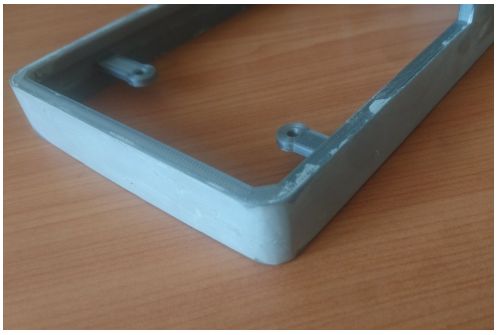
Zapúzdrenie zariadenia musí byť reprezentatívne. Je to časť s ktorou sa zákazník v bežnej prevádzke dostane do kontaktu. Rovnako tak je dôležité aby spĺňala požiadavky. Ochránila vnútro pred parou a ponúkala realizátorovi jednoduchú možnosť otvorenia a prístupu k použitému hardvéru v prípade poruchy.

Pri výbere materiálu môžem využiť kovy. Ich výhodou je pevnosť, odvádzanie tepla s čím je spojené chladenie a odolnosť voči vysokým teplotám bez deformácie. Nevýhodou je náročnosť pri spracovaní. Ďalšou možnosťou je plast. Stále populárnejšia 3D tlač z plastu je veľmi výhodná minimálne na vytvorenie formy a prototypov. Pri 3D tlači môžem použiť niekoľko druhov plastov a farebných variácii. Častejšie používaným je PLA. Hlavne z dôvodu jednoduchšieho tlačenia a lepšej pevnosti medzi vrstvami. To znamená že po vytlačení je výrobok pevnejší. Nevýhodou je nižšia odolnosť voči teplu. Tá sa udáva do 60-80°C. Druhou alternatívou je ABS, je opakom spomenutého PLA, pevnosť medzi vrstvami je výrazne nižšia a veľa pokusov o vytlačenie sa nepodarí. Avšak ponúka väčšiu odolnosti voči teplu. Tá sa udáva do 120°C.

Pri plastových výrobkoch sa dajú vonkajšie steny ešte povrchovo upraviť. Pomocou acetónu sa okraje plastu rozleptajú a po vyschnutí sa lepšie spoja. A pomocou sádry sa na povrch krabičky nanesie tenká vrstva, ktorá sa ešte povrchovo zahladí brúsnym papierom. Ukážka opracovania je na obrázku 2.9. Takto môžem získať hladší povrch a vyššiu odolnosť s možnosťou následného náteru vrchnou farbou.

2.4.10 Správa systému

V prípade programu na správu systému by mal výber programovacieho jazyka závisieť na serverovej časti. Obe tieto aplikácie budú využívať rovnakú databázu a dá sa očakávať, že niektoré databázové funkcie budú spoločné pre obe aplikácie. Preto sa odporúča zvoliť rovnaký programovací jazyk pre obe aplikácie a vytvoriť komunikačnú vrstvu, ktorá sa bude starať o získanie dát. Následne sa jednotlivé funkcie budú zdieľať, čím sa zníži možnosť výskytu chýb, skráti sa čas na vývoj a vyhnem sa redundantnému kódu.



Obr. 2.9: Opracovanie vytlačeného modelu

Kapitola 3

Návrh

V tejto kapitole je napísané, ako bude výsledná aplikácia vyzerať a aké prostriedky budem využívať. Môj výber je založený na analýze, kde boli spomenuté viaceré možnosti. Významom databázovej vrstvy, ktorá bude na databázovom serveri je uloženie dát a udržiavanie ich konzistencie. O ich úpravu sa bude starať aplikácia na správu systému. Aplikačný server bude obsahovať implementáciu chovania jednotlivých terminálov. Bude udržiavať ich stav, interné informácie a dáta k zobrazeniu. Tie v prípade zmeny sa odošlú na terminál. Jeho úlohou bude zobraziť prijaté informácii na displeji a sledovať interakciu od užívateľa. Každá interakcia s užívateľom je odoslaná terminálom a spracovaná na aplikačnom serveri. Po spracovaní sa zmeny odošlú späť na terminál a tým získavam nekonečný cyklus obsluhy informačného systému.

3.1 Server a jeho správa

Na základe analýzy som sa rozhodol pre využitie MySQL databázy. Z funkčného hľadiska mi ponúka všetky požiadavky potrebné na fungovanie systému 2.3. Dokonca v prípade MySQL je riešenie zadarmo bez potreby kupovať licenciu.

V prípade aplikačného servera a správe systému by som vzhľadom na moje skúsenosti preferoval programovací jazyk C#. Ponúka sieťovú komunikáciu a natívnu podporu pre MS SQL. Keďže som sa rozhodol pre použitie MySQL tak v prípade použitia voľne dostupnej knižnice MySQL.Data.dll sa dokáže aplikácia pripojiť aj na MySQL databázu. Z hľadiska syntaxe sa vývojári knižnice veľmi priblížili k natívnej knižnici pre MS SQL. V prípade programovacieho jazyka vývoj zastrešuje firma Microsoft a beží prioritne pod operačným systémom Windows. Síce existujú programy napísané v C# a spustené aj na iných platformách. Ale ich funkčná sila je značne obmedzená a programátor nemôže využiť všetky výhody programovacieho jazyka. Týmto smerom nepôjdem a ostanem pri štandardných aplikáciách C# s .NET 4.7.2[6]. Z toho vyplýva obmedzenie aplikačného servera na operačný systém Microsoft

Windows s čím sú spojené aj náklady na nákup licencie. Jedná sa však len o jedno zariadenie servera.

V prvej časti vývoja programu vytvorím komunikačnú vrstvu medzi programom a databázou. Tá bude zdieľaná a využívaná oboma aplikáciami. Následne aplikačný server implementujem ako systémovú službu a správu systému ako vizuálnu aplikáciu. V tomto prípade mi C# umožňuje vytvoriť grafické rozhranie pomocou WinForm, WPF alebo UWP aplikácie.

U WinForm aplikácie sa dizajn definuje programovo a grafické úpravy sú náročnejšie. Pri zvyšných dvoch je situácia jednoduchšia. Ich dizajn je definovaný pomocou značkovacieho jazyka XAML. Podporuje vytvorenie šablón a globálnych parametrov. Pre definovanie určitej vlastnosť na všetky texty v aplikácii, môžem definíciu zapísať do globálneho súbora, čím sa to automaticky aplikuje pri každom použití textu. Ohľadom UWP aplikácie, stále je to relatívne nový druh aplikácii a nestihol sa dostatočne spopulárniť. Stále sa objavujú nedostatky, ktoré je potrebné dodatočne opravovať. Keďže sú licencované firmou Microsoft, tak jednotlivé chyby je ťažké pretlačiť, aby boli opravené, keďže dennodenne dostávajú veľké množstvo návrhov na zmeny a opravy. Preto budem implementovať WPF aplikáciu. Jej logika bude implementovaná v C# a grafické uživateľské rozhranie bude zapísané v XAML súboroch.

3.2 Terminál

V prípade zariadenia v termináli som sa rozhodol pre mikropočítač Raspberry Pi spolu so 7 palcovým dotykovým displejom od rovnakej firmy. Ten ponúka skvelú podporu a spoluprácu s ním. Nevýhodou je, že momentálne firma neponúka alternatívu displeja s väčšou uhlopriečkou, prípadne s lepším rozlíšením, podsvietením či iné varianty. Keďže je to špecifický produkt od jednej firmy, môže nastať problém s dostupnosťou. Avšak u tejto firmy si nemyslím že by mal byť problém, keďže je úspešná a stále aktívne vytvára nové návody na používanie daného displeja. Ako som v analýze spomenul, mikropočítač disponuje veľkým množstvom periférii a vstupno-výstupnými portami. To mi ponúka možnosť využiť takmer akúkoľvek čítaciu hlavu a zvukovú signalizáciu dostupnú na trhu. Tým nechávam otvorený trh s konkurenciou a som schopný v prípade lepšej ponuky ju využiť a nahradiť pôvodný hardvér. To znamená, že nebudem viazaný na jedného výrobcu, ale dokážem sa prispôsobiť vývoji na trhu. Pri zvukovej signalizácii je výhodnejšie použitie bzučiaka miesto reproduktorov. Ten vyžaduje pripojenie na digitálne ovládaný výstupný port. Na trhu sa nachádza niekoľko výrobcov a variant bzučiakov. Dôležité pri výbere je výstupná frekvencia zvuku, vstupné napätie a hlasitosť. Keďže naše zariadenie Raspberry Pi pracuje s napätím 3,3V, tak to je aj jeho výstupné napätie. Preto pri výbere bzučiaka musí byť jeho vstupné napätie podporujúce túto hodnotu.

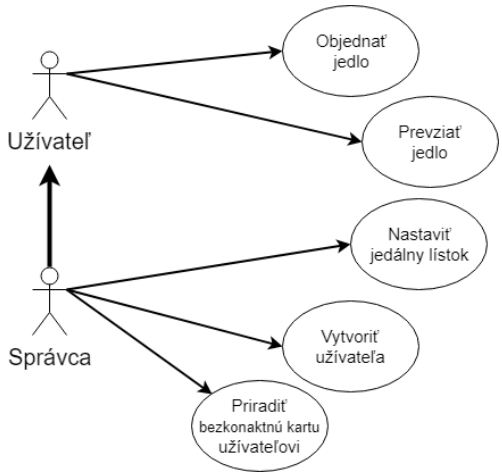
Na výrobu zapuzdrenia terminálu začnem s 3D tlačiarňou. Počas štúdia na vysokej škole som absolvoval predmet, zameraný na návrh a 3D tlač. Vďaka tomu budem môcť využiť znalosti nadobudnuté počas štúdia a vytvoriť dizajn a rozmiestnenie súčiastok vo vnútri. Keďže krabička bude vytlačená z plastu, je potrebné ju otestovať na teplo a pevnosť. V prípade, že nebude spĺňať požiadavky na uvedenie do prevádzky sa následne pokúsim nájsť firmu, ktorá by krabičku dokázala realizovať v rovnakom tvare ale z kovu, čím by zabezpečila aj lepšiu termoreguláciu a znížila pravdepodobnosť prehriatia terminálu.

V prípade obojstranného terminálu budú v krabičke dva zariadenia skladajúce sa z mikropočítača Raspberry Pi a displeja. Na beh mikropočítača je potrebný elektrický prúd o hodnote 2,5A podľa manuálu [7]. Napájanie displeja je zabezpečené z mikropočítača a je obsiahnuté v spomenutom prúde 2,5A. Čo v súčte pre dva zariadenia je 5A. V prepočte na výkon je to 25W, ktorých musí zdroj dokázať dodať. V prípade, že napájanie do terminálu pôjde jedným káblom, budem musieť použiť vhodný kábel a v krabičke zabezpečiť rozdeľovač. Pre displej mi problém s napájania vyrieši samotné Raspberry Pi, ktoré po vzájomnom prepojení zdieľa svoje napájanie. Preto postačí zabezpečiť napájanie dvoch mikropočítačov v krabičke. To zabezpečím použitím jedného rozdeľovača. U internetového pripojenia môžem využiť tiež rozdeľovač, opísaný v analýze a spojiť dva pripojenia do jedného. Tým ušetrím jeden internetový kábel pri inštalácii nového terminálu.

U jednostranného displeja je situácia jednoduchšia. Už niekoľko rokov sledujem použitie PoE u mikropočítačov, IP telefónov či webových kamerách. To zabezpečuje integráciu napájania do internetového pripojenia. Keďže Raspberry Pi nemá vstavanú podporu PoE, je potrebné použitie rozdeľovača, ktoré v krabičke rozdelí pôvodný prívod na internetové pripojenie a napájanie, ktoré následne stabilizuje na potrebných 5V.

3.3 Prípady použitia

Na obrázku 3.1 sú zobrazené prípady použitia. Kde užívateľ má len obmedzené možnosti využitia systému a v prípade správcu sú tieto funkcie rozšírené o ďalšie možnosti. Pre zachovanie prehľadnosti sú na obrázku zachytené len najčastejšie prípady.



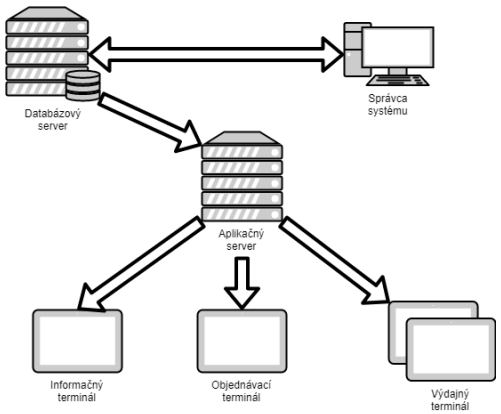
Obr. 3.1: Prípady použitia systému

Kapitola 4

Výsledné riešenie

Graficky zobrazené časti systému, medzi ktorými prebieha komunikácia sú zobrazené na obrázku 4.1. V strede obrázka je aplikačný server na ktorom beží systémová služba implementovaná v jazyku C#. Tá po spustení načíta konfiguráciu uloženú v databázovom serveri. Na jej základe spustí potrebné obslužné funkcie jednotlivých terminálov a nadviaže s nimi spojenie. Každý terminál je identifikovaný IP adresou a portom. Ten je potrebný v prípade, že na zariadení je spustených viacej inštancii aplikácie. Ďalšie parametre uložené v databázi sú funkcia terminálu a jeho meno pre jednoduchšiu identifikáciu v prípade použitia viacerých terminálov s rovnakou funkciou v jednej prevádzke.

Softvér bežiaci na termináli je implementovaný v jazyku C++ a pre zobrazenie grafickej časti je použité vývojové prostredie QT. To zabezpečuje komunikáciu s knižnicou xLib na vykresľovanie užívateľského rozhrania. Všetky termináli obsahujú rovnaký softvér a nie je potrebný zásah do terminálu pri zmene jeho funkcie. Pri takejto požiadavke je potrebné v databázi nastaviť novú funkciu terminálu a reštartovať aplikačný server pre načítanie novej kon-



Obr. 4.1: Schéma systému

figurácie.

Po spustení systému je možné využívať služby pripojených terminálov. Zákazníci si môžu po priložení identifikačnej karty zobraziť objednané jedlá alebo vytvoriť novú objednávku. V prípade, že klient má jedlo objednané, môže si ho ísť vyzdvihnúť u výdajného terminálu. Toto je opis každodenného štandardného používania, avšak systém ponúka aj ďalšie funkcie. Obsluha výdajného terminálu môže počas výdaja jedál prezrieť históriu vydaných, pozastaviť výdaj alebo prezerať jedálny lístok na aktuálny deň.

Ďalšia časť systému, ktorá sa stará o jeho správu umožňuje v prvom rade nastavenie jedál, ktoré sa v jedálni varia a jedálny lístok na jednotlivé dni. Ďalej ponúka možnosti nastavenia terminálov a ich parametrov, správu klientov, ich osobné údaje, identifikačné karty a ich objednávky. Užívatelia aplikácie majú prístup ku všetkým údajom, ktoré môžu pridávať, vymazávať alebo editovať. Taktiež je možné zobraziť stav všetkých objednávok zákazníka v prípade sťažnosti, že sa terminál správal neočakávane, keď s ním pracoval a skontrolovať, čo sa v skutočnosti v systéme stalo.

Kapitola 5

Implementácia

V prvej časti kapitole sa zameriam na prepojenie a vzájomnú komunikáciu medzi komponentami. Následne opíšem implementáciu jednotlivých komponent.

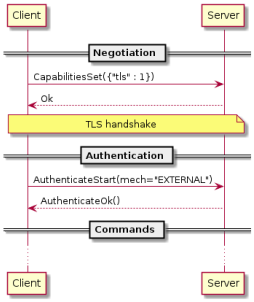
5.1 Komunikácia

Prebieha pomocou lokálnej siete a TCP/IP paketov. Medzi databázovým serverom a dátovou vrstvou aplikácii, je to pomocou knižnice MySQL.Data.dll, ktorá vyžaduje prihlasovacie údaje a adresu databázy. Druhý typ komunikácie je typu klient-server medzi aplikačným serverom a terminálmi.

5.1.1 Komunikácia s MySQL

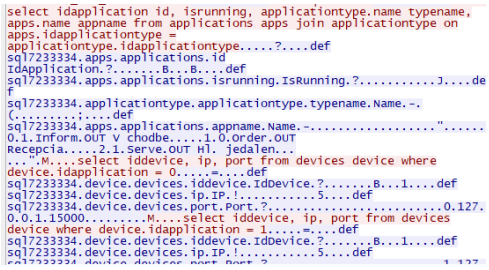
Vďaka aplikácii Wireshark sa dá sledovať komunikácia medzi aplikačným a databázovým serverom. V prípade, že sa vypne SSL šifrovaná komunikácia je možné sledovať obsah správ. Počas komunikácie využíva spomenutý TCP/IP protokol v transportnej a sieťovej vrstve a MySQL protokol v aplikačnej vrstve. Po nadviazaní spojenia je potrebná autentifikácia. Priebeh je zobrazený na obrázku 5.1. V prvej časti zistí klient podporu TLS. V prípade dostupnosti pokračuje v komunikácii pomocou TLS zabezpečenia a overí prihlasovacie údaje.

Po úspešnom prihlásení prebieha výmena dát v podobe vykonávania dotazov a volaním procedúr. Príklad takej komunikácie je zobrazený na obrázku 5.2. Červeným je zvýraznená správa v smere z aplikačného na databázový server a modrou jeho odpovede. Na prvom riadku vidieť textovú reprezentáciu príkazu *SELECT*. Odpoveď z databázového servera začína názvom stĺpca nasledovaná jeho dátami. Komunikácia ďalej pokračovala rovnakým spôsobom s ďalšími dotazmi.



Obr. 5.1: Autentifikácia pomocou MySQL

Zdroj: https://dev.mysql.com/doc/internals/en/images/plantuml.png



Obr. 5.2: Komunikácia protokolu MySQL

5.1.2 Komunikácia medzi serverom a terminálom

Používam architektúru klient-server, kde server po spustení začne očakávať nadviazanie spojenia na TCP/IP porte. U klienta je potrebné zadať IP adresu a port servera, na ktorý sa má pripojiť. V prípade servera je to len port. Preto z pohľadu komunikácie bude úlohu servera zastupovať terminál, kde jediným parametrom je port a ten bude pevne daný pre každý terminál. Týmto riešením som centralizoval všetky nastavenia na stranu aplikačného servera, ktorý bude obsahovať IP adresu a port každého terminálu.

5.1.2.1 Formát obsahu správ

Počas komunikácie je potrebné zabezpečiť čitateľnosť paketov, aby v prípade poruchy sa dala sledovať predchádzajúca komunikácia. Dobre čitateľný je formát XML ktorý je použitý v projekte. XML používa značky, niekedy pomenované ako tagy. Tie identifikujú názov dát medzi jeho otváracím a uzatváracím tagom. Počas komunikácie bude každá správa obalená tagom, ktorý definuje smer komunikácie. V smere od terminálu do aplikačného servera budem používať tag „RLO“, v prípade opačnom „RLI“. Následne telo správy obsahuje potrebné dáta na odoslanie. Príklad štruktúry správ v komunikácii je vidieť na obrázku 5.3.

```
<RPI>
  <Show id=”diMain” >
    <ModifyAtribute id=”Title” attribute=”Text” value=”
     Zadajte meno:”/>
  </Show>
</RPI>

```

Obr. 5.3: Obsah XML správy

Po nadviazaní TCP/IP spojenia prebieha kontrola kompatibility verzie a pre terminál sa nastavuje systémový čas, štartovací znak čítačky kariet a ukladajú sa štruktúry obrazoviek do terminálu. Všetky funkcie sú odosielané podľa opísaného formátu XML.

Aplikácie sú implementované tak, aby obe využívali asynchrónnu komunikáciu pomocou signálov. Tým predídem aktívnemu čakaniu na príjem správy a zabezpečím okamžité spracovanie prichádzajúcej správy. Z hľadiska implementácie je potrebné pri vytváraní inštancie komunikátoru definovať obslužnú funkciu, ktorá bude volaná po prijatí paketu.

Typy správ:

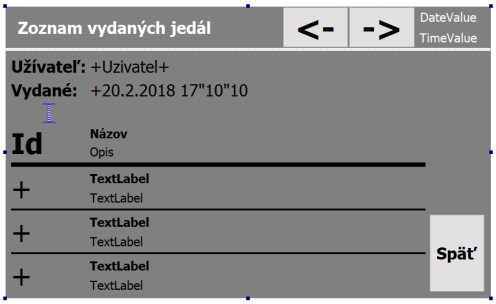
- Nastaviť systémový čas
- Nastaviť štartovací znak čítačky kariet
- Nastaviť štýl aplikácie
- Uložiť rozloženie obrazovky pod identifikátorom
- Zobraziť uložené rozloženie obrazovky definované identifikátorom
- Nastaviť text na obrazovke
- Spusti zvukovú signalizáciu
- Oznámenie o priloženej karte
- Oznámenie o stlačení tlačidla

5.2 Definícia užívateľskej obrazovky

QT aplikácia využíva na definovanie rozloženia okna XML. V ňom sú zapísané jednotlivé bloky. Blokom nazývam všetky objekty, ktoré na obrazovke môžem vykresliť. Zároveň každý blok je identifikovaný unikátnym menom. Medzi typy objektov patrí niekoľko druhov tlačidiel, textov, geometrických tvarov, obrázkov, tvarov a ďalšie. Vďaka XML môžem pridávať k jednotlivým tagom parametre ako napríklad rozmery, farby, ohraničenie, okraje, v prípade

```
<widget class="QLabel" name="TitleValue">
 <property name="sizePolicy">
  <sizepolicy hsizetype="Ignored" vsizetype="Ignored">
   <horstretch>0</horstretch>
   <verstretch>0</verstretch>
  </sizepolicy>
 </property>
 <property name="font">
  <font>
   <pointsize>20</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="styleSheet">
  <string notr="true">color:white</string>
 </property>
 <property name="text">
  <string>Zoznam vydaných jedál</string>
 </property>
 <property name="class" stdset="0">
  <string>clientTextTitleStyle</string>
 </property>
</widget>
```

Obr. 5.4: Textová podoba rozloženia obrazovky



Obr. 5.5: Grafická reprezentácia XML súboru

použitého textu jeho veľkosť, typ a do tela tagu vnárať ďalšie tagy a tým vytvárať stromovú štruktúru. Tá vo výsledku reprezentuje užívateľskú obrazovku. Na obrázku 5.4 sa dá vidieť príklad, kde je zobrazený popis okna v XML jazyku a na obrázku 5.5 jeho grafická reprezentácia.

```
CREATE TABLE ‘orders‘ (
  ‘IdOrder‘ int(11) NOT NULL,
  ‘ForDate‘ date NOT NULL,
  ‘IdMenu‘ int(11) NOT NULL,
  ‘IdClient‘ int(11) NOT NULL,
  ‘Vydane‘ datetime(6) DEFAULT NULL,
  PRIMARY KEY (‘IdOrder‘),
  KEY ‘FK_ORDERS_CLIENTS‘ (‘IdClient‘),
  KEY ‘FK_ORDERS_MENU‘ (‘ForDate‘,‘IdMenu‘),
  CONSTRAINT ‘FK_ORDERS_CLIENTS‘
    FOREIGN KEY (‘IdClient‘) REFERENCES ‘clients‘ (‘
      IdClient‘)
    ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT ‘FK_ORDERS_MENU‘
    FOREIGN KEY (‘ForDate‘, ‘IdMenu‘) REFERENCES ‘
      menu‘ (‘ForDate‘, ‘IdMenu‘)
    ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Obr. 5.6: Vytvorenie tabuľky v jazyku SQL

5.3 Databázový server

Obsahuje niekoľko relácii reprezentujúce stav systému a dáta k jeho fungovaniu. Keďže používam MySQL databázu, je potrebné pomocou DDL príkazov z programovacieho jazyka SQL definovať štruktúry a vzťahy medzi reláciami. Tým vznikne databázová schéma. Následne počas prevádzky sa budú používať DML príkazy SQL jazyka. Tie slúžia na manipuláciu s dátami v rámci vytvorenej schémy.

5.3.1 SQL programovací jazyk

Pomocou jazyka SQL, dokážem vytvoriť relácie dát, závislosti medzi nimi, procedúry, dočasné tabuľky. Tieto funkcie patria do DDL skupiny na vytvorenie štruktúry respektíve schémy databázy. Príklad príkazu *CREATE* na vytvorenie relácie objednávok je zobrazený na obrázku 5.6.

Ďalej SQL jazyk umožňuje prácu s dátami v reláciách. Táto skupina príkazov sa volá DML a patria sem operácie na vkladanie, vymazávanie, editáciu a výber záznamov z relácii. Každý príkaz môže byť rozšírený o podmienku. V aplikácii používam príkaz SELECT s podmienkou definujúcou jeho kartu na zistenie mena klienta. Tento dotaz je zapísaný na obrázku 5.7, kde je použité číslo karty 5.

Systém sa zaoberá výdajom jedál. K tomu je neoddeliteľnou súčasťou jedálny lístok. Ten sa skladá zo záznamov s relácie menu. Každý z nich obsahuje dátum, na ktorý je vytvorený a poradie v daný deň. V kombinácii tvoria primárny kľúč tabuľky. Záznam ďalej obsahuje prepojenia na polievku,

```
1 SELECT FirstName, LastName FROM clients WHERE
      CardNumber = 5;
```

Obr. 5.7: Príkaz SELECT v jazyku SQLMySQL

hlavné jedlo a dezert. V prípade, že jedáleň nechce ponúkať hotové kombinácie jedál, je možné nevyplniť niektorú z nich. Prípadne môžu pre každé jedlo vytvoriť vlastný záznam a predávať polievky, hlavné jedlá a dezerty osobitne. Posledným príznakom záznamu je identifikátor, kde si objednávku môžu prevziať. Používa sa v prípade, že systém používa viacej výdajných miest.

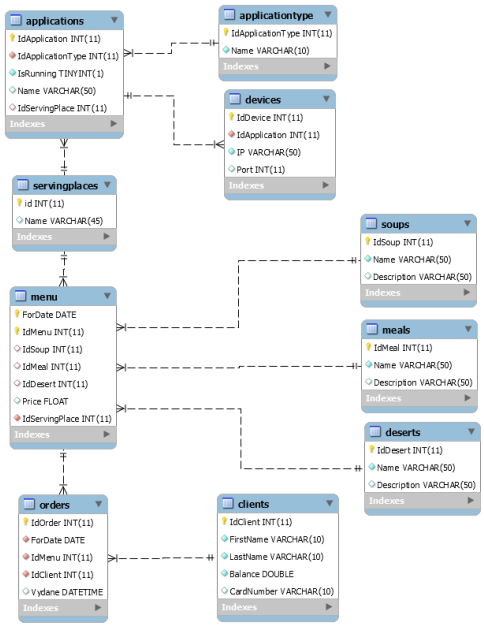
Keďže objednávka spája klienta a menu, je potrebné unikátne tieto dva záznamy identifikovať. V prípade klienta používam jeho identifikačné číslo. U menu je unikátnosť zabezpečená kombináciou dátumu ku ktorému dňu je menu určené a poradie jedla v daný deň. Okrem prepojenia je objednávka rozšírená o vlastné identifikačné číslo, ktoré je zároveň primárnym kľúčom tabuľky. Vďaka tomu je táto hodnota unikátna a dokážem každú objednávku jednoznačne identifikovať. Poslednou hodnotou je čas a dátum, keby bola objednávka vydaná. V prípade, že táto hodnota nie je zadaná, v databáze sa zobrazuje ako „null“ alebo prázdna hodnota. Znamená to, že klient si jedlo objednal, ale zatiaľ mu nebolo vydané. Následne po vydaní danej objednávky sa v databáze upraví jej záznam a nastaví sa čas výdaja.

5.3.2 Diagram databáze

ER diagram databáze je zobrazený na obrázku 5.8. V hornej časti zobrazuje systémovú časť. Tá obsahuje vytvorené typy aplikácii. Momentálne sú vytvorené tri typy ako bolo spomenuté v Analýze 2. V prípade rozšírenia aplikačného servera o nový typ terminálu, stačí meno nového terminálu pridať do databázy medzi typy aplikácii a v správe systému bude možné ho použiť. V ďalšej tabuľke je zoznam zariadení alebo inak nazvaných terminálov. Tie sú identifikované IP adresou a portom. Prepojenie týchto dvoch častí je pomocou aplikácie danej prevádzky.

V spodnej časti obrázka 5.8 je zobrazená časť databázy, ktorá sa v čase dynamicky vyvíja. Obsahuje zoznam klientov spolu s ich identifikačnými kartami, zoznamy polievok, jedál a dezertov, ktoré jedáleň dokáže uvariť a tie následne kombinuje a vytvára jedálničky na jednotlivé dni. Po vytvorení jedálnička je možné vytvoriť objednávku viazanú na klienta a jedlo.

Tieto dve časti sú spoločne prepojené cez zoznam výdajných miest. Toto prepojenie je potrebné v prípade použitia viacerých výdajných miest, kde každá objednávka musí mať nastavené pre ktoré výdajné miesto patrí. V systéme je možné pre rovnaké výdajné miesto priradiť viac výdajných terminálov. V takom prípade si klient môže prevziať svoju objednávku u ktoréhokoľvek z nich.



Obr. 5.8: ER diagram databázi

5.3.3 Procedúry

Práca s databázou je zaobalená do volania procedúr. Tie sú interne uložené v databázovom serveri a zastrešujú niekoľko SQL príkazov do jedného volania. Pri volaní procedúry je potrebné definovať všetky jej parametre, ktoré nemajú prednastavenú hodnotu. Následne odpoveď získavam v podobe tabuľky. Príklad použitia procedúry je pri výdaji jedla, ktorej implementácia je zobrazená na obrázku 5.9. V tomto prípade je vstupným parametrom identifikačné číslo klienta. Toto číslo je možné získať pomocou čísla bezkontaktnej karty, ktorú klient priložil k čítačke kariet cez tabuľku klientov. Avšak pri výdaji je už klient prihlásený a poznám jeho identifikačné údaje.

Funkčná časť procedúry začína za kľúčovým slovom BEGIN a končí slovom „END“. V tele procedúry môžem vidieť deklaráciu identifikačného čísla objednávky. To sa inicializuje na prvú objednávku klienta pre aktuálny deň, ktorá nebola vydaná. Toto obmedzenie je zapísané vo „where“ podmienke. V našom príklade sa skladá z troch menších výrazov. Prvá časť vyberie z objednávok len tie, ktoré patria prihlásenému klientovi. To je definované ako výraz „IdClient = ClientId“. V druhej podmienke vyberie filter len objednávky, ktoré neboli vydané a teda hodnota v stĺpci Vydane je prázdna. V jazyku SQL zapísané ako „Vydane is null“. Kde „null“ znamená prázdna hodnota. A poslednou tretou častou je výber objednávok na daný deň. Tu je použitá SQL funkcia CAST na získanie dátumu z globálne hodnoty „CURRENT_TIMESTAMP“ reprezentujúca aktuálny dátum a čas. Keďže objednávka je platná počas

```
1  CREATE DEFINER=‘root‘@‘localhost‘ PROCEDURE ‘
       ServeOrder‘(
2    IN ClientId int
3  )
4  BEGIN
5    DECLARE tmpOrderId int;
6    SET tmpOrderId =
7        (SELECT IdOrder FROM Orders
8          WHERE IdClient = ClientId
9        AND Vydane is null
10           AND ForDate = CAST(CURRENT_TIMESTAMP AS
         DATE)
11     );
12   UPDATE Orders SET Vydane = CURRENT_TIMESTAMP
       WHERE IdOrder = tmpOrderId;
13   SELECT Orders.IdMenu, Orders.ForDate, Soups.*, Meals.*,
       Deserts.*  FROM Orders
14     LEFT JOIN Menu ON Orders.IdMenu = Menu.IdMenu
15     LEFT JOIN Soups ON Menu.IdSoup = Soups.IdSoup
16     LEFT JOIN Meals ON Menu.IdMeal = Meals.IdMeal
17     LEFT JOIN Deserts ON Menu.IdDesert = Deserts.IdDesert
18     WHERE IdOrder = tmpOrderId;
19 END
```

Obr. 5.9: Vytvorenie procedúry v jazyku MySQL

celého dňa, tak čas v tomto prípade nie je potrebný. Tieto tri podmienky sú spojené kľúčovým slovom „AND“ vyjadrujúce binárnu operáciu. Tá skladá všetky podmienky do jedného s významom, že výsledné záznamy musia spĺňať všetky tri podmienky. V prípade že taká objednávka existuje, sa v ďalšom kroku jej hodnota vydané nastaví na aktuálny čas. V tomto prípade používam celý formát dátumu aj s časom. Aby bolo následne možné vyhľadať, kedy klient si vyzdvihol objednávku. Aplikácii, ktorá túto procedúru volala sa následne vráti popis jedla spolu s popisom polievky, hlavného jedla a dezertu, ktoré k objednávke patria. Vďaka tomu je výdajný terminál schopný zobraziť obsluhe pri výdajnom pulte, ktoré jedlo má klientovi vydať. Keďže procedúra vracia názov a presný popis jedál, je potrebné ho získať z tabuliek Soups, Meals a Deserts. Avšak procedúra pozná len identifikátor objednávky. Preto pomocou prepojenia sa dokážem dostať na menu, ktoré obsahuje ďalšie prepojenia do týchto tabuliek. Vďaka týmto spojeniam a použití funkcie JOIN na spojenie tabuliek je procedúra schopná vrátiť aplikácii všetky informácie v jednom volaní.

5.4 Terminál

Predstavuje mikropočítač Raspberry Pi s pripojeným dotykovým displejom a čítačkou kariet. Beží na operačnom systéme Raspbian, založenom na Linuxovom jadre. Aplikácia pre toto zariadenie je implementovaná na podporu zobrazenia rôzdnych grafických objektov a tým povoľuje využitie aj v iných oblastiach. Po jej zapnutí sa spustí TCP/IP server. Ten očakáva pripojenie klienta, ktorý ho bude ovládať. Ovládaním znamená odosielanie inštrukcii terminálu, čo má zobraziť na jeho dotykovom displeji. A v prípade interakcie užívateľov priložením karty alebo stlačením tlačidla na obrazovke vhodne reagovať. V mojom prípade bude klientom, ktorý sa na mikropočítač pripojí a bude ho ovládať aplikačný server.

Terminál povoľuje pripojenie len jedného klienta súčasne. V prípade, že terminál má aktuálne pripojený server, tak v prípade nového pripojenia sa ukončí spojenie so starým a pokračuje v práci s novým pripojením. Ten musí terminál inicializovať znovu, keďže všetky nastavenia zo starého spojenia sa vymazávajú.

Zariadenie využíva sieťový konektor 8p8c, známy aj ako RJ45, na pripojenie do lokálnej siete. V internetových sieťach, rovnako ako aj v lokálnej je každé zariadenie identifikované unikátnou IP adresou. Tá môže byť dynamicky získaná z DNS servera alebo staticky zapísaná v systémových súborov. V prípade dynamickej adresy sa IP adresa môže počas behu zmeniť. To by spôsobilo problém pre server, ktorý by sa snažil hľadať terminál na jeho pôvodnej adrese. Riešením by mohlo by použitie doménového mena. Avšak to nie je unikátny identifikátor, preto sa odporúča po uvedení terminálu do prevádzky mu prideliť statickú adresu IP a zapísať ju do systémových súborov. To zabezpečí jej automatické nastavenie pri ďalšom spustení.

5.4.1 Úpravy operačného systému

Pri vývoji pracujem s operačným systémom založeným na Debiane, to mi dovoľuje ho upraviť. Rozhodol som sa pre odľahčenú verziu Raspbianu určenú pre daný mikropočítač. Tá obsahuje iba základné funkcie potrebné na prácu s ním, dokonca neobsahuje ani pracovnú plochu a po zapnutí ostáva v stave s konzolou. Po prvom spustení systému som odinštaloval všetky nepotrebné programy a služby. Následne som nainštalovali programy, ktoré budem využívať alebo sú potrebné pre spustenie aplikácie. Medzi ne patria aplikácie xorg, xinit a x11, ktoré v kombinácii dokážu komunikovať pomocou xLib s aplikáciou a pomocou protokolu xWindow vykresľovať na obrazovku. Bez nich by nebolo možné spustiť grafickú aplikáciu. Je to tiež miniatúrna verzia programov pre podporu grafického rozhrania, ktorá v sebe neobsahuje správu okien aplikácii. To znamená, že dovoľuje spustenie len jednej aplikácie a v režime na celú obrazovku. V mojom prípade je toto chovanie výhodou, keďže predídem možnosti aplikáciu minimalizovať či otvoriť inú na pozadí.

Ďalšou výhodou použitia základných verzii programov je tiež ich veľkosť. Tá je zvyčajne nižšia, keďže neponúka celú funkcionalitu. Mne toto pomôže v šetrení pamäte, keďže Raspberry Pi používa microSD kartu ako jediné úložisko dát.

5.4.2 Zoznam príkazov

K ovládaniu terminálu sa používajú textové príkazy vo formáte XML. Tie sa následne dekódujú a vykonávajú chronologicky. Tento zoznam nie je konečný a môže sa v ďalších verziách softvéru rozšíriť na základe rozširovania funkcionality terminálu. Prvá časť príkazov je systémová:

- **Init** s parametrami StartKey a CurrentTime – inicializuje zariadenie a nastavuje štartovací znak čítačky kariet a systémový čas zariadenia
- **StyleSheet** – definuje globálny štýl aplikácie, ktorý je definovaný v tele tagu
- **StoreLayout** s parametrom Id – ukladá nové rozloženie obrazovky v jazyku XML do internej pamäti.

Druhá časť zoznamu obsahuje príkazy opisujúce aktuálny stav terminálu. Medzi ne patrí:

- **Show** s parametrom id – Zobrazí rozloženie obrazovky identifikované parametrom id, ktoré musí byť pred zobrazením uložené v internej pamäti pomocou príkazu **StoreLayout**
- **ModifyAttribute** s parametrami Id, Attribute a Value – využíva sa na modifikáciu objektov zobrazených na obrazovke. Id jednoznačne definuje názov objektu, Attribute definuje vlastnosť, ktorej aplikácia nastavý hodnotu z Value
- **Message** s parametrami Title a Message– zobrazí systémové okno s nadpisom Title a správou Message
- **Display** s parametrom isEnable – na základe parametru prepne displej do úsporného režimu alebo ho zobudí z neho
- **Beep** s parametrami Length, Delay a Count – spustí zvukovú signalizáciu o dĺžke Length s opakovaniami Count a medzi opakovaniami je pauza o dĺžke Delay
- **Modal** s parametrom text a zoznamom tlačidiel – zobrazí modálne okno s textom a tlačidlami
- **ModalClose** – ak je otvorené modálne okno, zavrie ho

- **ButtonClick** s parametrom Id – reprezentujúce kliknutie na tlačidlo, ktoré je identifikované pomocou Id

- **CardRead** s parametrom Number – oznamujúca priloženie karty, kde jej číslo je obsahom atribútu Number

- **Error** s parametrom Message – akákoľvek chyba závažnosti Warning alebo Critical, ktorá nastala na strane terminálu s popisom obsiahnutým v atribúte Message

V prípade že aplikačný server nepošle všetky parametre terminálu, alebo je v nich chyba. Zapíše podrobný problém do logu a odošle spätnú správu s chybovým textom vo formáte XML správy Error.

5.4.3 Zobrazenie rozloženia obrazovky z textovej podoby

Vytvorenie vlastných stavebných prvkov obrazovky je náročné a potrebné každý z nich definovať osobitne. Preto som sa rozhodol využiť už existujúce riešenie, ktoré ponúka QT. Ako v analýze bolo spomenuté mi ponúka definovať aj zložitejšie zobrazenia obsahujúce relatívne a absolútne rozmiestnenie objektov či automatické prispôsobenie obsahu.

Počas inicializácie obdrží terminál jednotlivé rozloženia identifikované pomocou textu. Tie si aplikácia interne uloží do zoznamu, kde kľúčom vo vyhľadávaní je jeho meno.

Následne počas behu aplikácie sa využívajú príkazy *Show* s identifikátorom uloženej obrazovky. Podľa neho aplikácia vyhľadá príslušné zobrazenie a vykreslí ho. Zobrazenie z textovej podoby XML zabezpečuje *QUiLoader* s funkciou *load*. Tá na vstup dostane XML text v podobe *QBuffer* s kódovaním UTF-8 aby som zabezpečili správne zobrazenie českých a slovenských písmen. Výstupom je *QWidget* reprezentujúci kontajner so všetkými prvkami k zobrazeniu. Avšak vykresliť do okna dokáže aplikácia len objekt, ktorý dedí od *QLayout*. Tým je napríklad *QVBoxLayout*. Preto si jeho inštanciu vytvorím a pridám mu už vytvorený *QWidget*. Celá imlementácia je na obrázku 5.10. Medzi prednastavené parametre objektu sú okraje o hodne 5 pixelov z každej strany. Keďže používam malý displej, potrebujem využiť čo najväčšiu časť z neho. Preto pôvodné hodnoty okrajov nastavím na nulu. Týmto som si vytvorili reprezentáciu nového okna a už je len potrebné ho nastaviť do aplikačného okna. V prvom rade vymažem všetky objekty na obrazovke a následne pridám požadované rozloženie s názvom „layout“.

5.4.4 Úprava objektov obrazovky

Aplikovaním rozloženia obrazovky podľa požiadaviek servera ešte nezobrazujem konečný stav okna. V prípade prihlásenia užívateľa je potrebné upraviť

```
1  void Layout::SetLayout(QString layoutName)
2  {
3    QBuffer buf;
4    buf.setData(Layouts[layoutName].toUtf8());
5    QWidget *myWidget = loader.load(&buf, this);
6
7    QVBoxLayout *layout = new QVBoxLayout;
8    layout->addWidget(myWidget);
9    layout->setContentsMargins(0,0,0,0);
10
11   qDeleteAll(ui->centralWidget->children());
12
13   ui->centralWidget->setLayout(layout);
14 }
```

Obr. 5.10: Zobrazenie XML reprezentáciu obrazovky

texty, ktoré sa na obrazovke zobrazujú. Prípadne môžem ovládať farbu obrazovky podľa stavu, či klient môže priložiť kartu k čítačke kariet reprezentujúce zelenou. Alebo má čakať reprezentujúce červenou. K tejto funkcii je definovaný typ správy s názvom *ModifyAttribute*. Po prijatí tohto príkazu sa spracujú jeho parametri a aplikácia sa pokúsi nájsť v jej okne objekt s názvom z parametru Id. Pri úspešnom nájdení môžem pomocou funkcie *setProperty* nad objektom upraviť požadovaná hodnotu. Táto funkcia je dostačujúca vo väčšinu prípadov. Avšak počas vývoju aplikačného servera som sa stretol so situáciou, kedy som sa snažil využiť funkcie, ktoré nepodporovala. Tie funkcie som definovali a doprogramovali ručne. Takou je nastavenie viditeľnosti objektu a dynamické rozloženie. To vyžadovalo naprogramovanie výnimiek, ktoré kontrolujú názov parametru. V prípade, že je to jeden z týchto, zavolala sa špecifická rutina namiesto použitia spomínanej *setProperty*.

5.4.5 Voliteľné argumenty

Pri spustení programu je možné definovať jeho argumenty. Niektoré z nich sú parametrizované a vyžadujú dodatočnú informáciu. Pre použitie parametrov je potrebné program spustiť z príkazovej riadky a za názvom programu sa píšu so symbolickou predponou pomlčky. S nimi je možné upraviť chovanie aplikácie. Medzi implementované argumenty patria:

- h – vypíše textovú nápovedu k programu
- v – vypíše verziu programu
- e – spustí program vo variante emulátora
- p – nastaví port, na ktorom bude aplikácia očakávať pripojenie aplikačného servera

- d – definuje id displeja, na ktorom sa aplikácia spustí v prípade pripojených viac displejov k zariadeniu
- l – nastaví počet ukladaných súborov s logmi. Kde jeden súbor predstavuje jeden deň

Výpis aplikácie je pomocou funkcie *print*. Tá výstup smeruje do konzoly, z ktorej je aplikácia spustená.

5.4.6 Automatické spustenie aplikácie

Operačný systém Linux obsahuje súbor s príkazmi, ktorý sa vykoná po načítaní operačného systému. Ten zabezpečí automatické spustenie našej aplikácie. Avšak aby bolo možné aplikáciu reštartovať v prípade vážnej chyby, tak som vytvorili skript, ktorý v nekonečnom cykle spustí aplikáciu, počká na jej zatvorenie, zapíše si návratovú hodnotu zatvorenia s aktuálnym časom a znovu ju spustí. Preto do pôvodného systémového súboru som nezapísal umiestnenie programu, ktorý sa má spustiť. Ale skriptu, ktorý sa o beh programu bude starať. Keďže terminál používa jeden displej, nie je potrebné definovať na ktorom sa má aplikácia spustiť ani port aplikácie na ktorom má očakávať pripojenie. Aktuálne sa aplikácia spúšťa bez dodatočných parametrov a využíva prednastavené chovanie. Ale napríklad ak by som rozšíril kapacitu internej pamäte, je možné bez zásahu do programu zmeniť počet uložených logovacích súborov, reprezentujúcich počet dní zálohy.

5.4.7 Pripojenie čítacej hlavy

Na trhu s čítacími hlavami existuje niekoľko výrobcov a každý ponúka viaceré riešenia. Tie sa dajú rozdeliť do dvoch hlavných skupín, kde jedna používa na pripojenie USB port. A druhá skupina využíva inú dostupnú komunikáciu. Najjednoduchším na programovanie je USB čítacia hlava, ktorá sa javí systému ako ďalšia pripojená klávesnica. V prípade programovania je potrebné zabezpečiť odchytávanie všetkých stlačených znakov klávesnice a spracovávať ich. V ďalších možností komunikácie sú napríklad:

- UART
- I2C
- SPI

Ich koncept je rovnaký s USB čítacou hlavou. Po priložení karty odošlú štartovací znak, číslo karty a ukončia prenos koncovým znakom. To znamená že k implementácii musím aktivovať danú komunikáciu na zariadení a zabezpečiť obsluhu prijatých dát. Tá zabezpečuje rozpoznanie štartovacieho, koncového znaku a uloženie čísla karty. Implementačne je to funkcia, ktorá je volaná s

```
1 void BeepSound::OneBeep(double mSec)
2 {
3     this->gpio->setval_gpio("1");
4     QThread::msleep(mSec);
5     this->gpio->setval_gpio("0");
6 }
```

Obr. 5.11: Implementácia zvukovej signalizácie

parametrom znaku, ktorý bol prijatý. V prípade prijatia štartovacieho znaku inicializuje premennú na uloženie čísla karty a nastaví príznak na ich ukladanie. Pri ďalšom volaní už funkcia vie, že prijatý znak je súčasťou čísla karty a uloží si ho. V prípade, že prijatým znakom je koncový znak, vyvolá signál s parametrom čísla karty. Obsluha signálu zabezpečuje okamžité odoslanie paketu so správou, ktorá obsahuje túto udalosť zapísanú podľa XML štruktúry.

5.4.8 Zvuková signalizácia

Zariadenie je vybavené GPIO portami, čo znamená General-Purpose Input/Output. V preklade vstupy a výstupy so všeobecným použitím. Aplikácia ich využíva na aktiváciu výstupného pinu, ktorým ovláda pripojený bzučiak. Aktivovaním sa rozumie nastavenie kladného napätia na daný pin. V neaktívnom stave by napätie na pine malo zodpovedať 0 voltom. U kladného napätia je to prevádzkové napätie mikropočítača, čo znamená 3,3 volta. To je dostatočné na nami vybraný bzučiak k spusteniu.

Implementácia zvukovej signalizácie je umiestnená v samostatnej triede s názvom *beepsound.cpp*. Pri inicializácii triedy sa nastaví požadovaný pin za výstupný, čím je pripravený k používaniu. Príklad jednorazového pípnutia s dĺžkou definovanou ako parameter je zobrazený na obrázku 5.11. V prvom riadku aktivuje zvukovú signalizáciu. Následne uspí dané vlákno na požadovaný čas a nakoniec zvuk vypne.

U zvukovej signalizácie sú dôležité parametre hlasitosť a frekvencia. Frekvencia je určená výrobcom, tú som si vybral pri kúpe zariadenia a nedá sa zmeniť. V prípade hlasitosti ju môžem prispôsobiť veľkosťou napätia. Zvyčajne výrobca udáva rozmedzie, v ktorom generuje zvuk. Avšak v mojom prípade nie som schopný zvýšiť alebo znížiť výstupné napätie programovo. Možnosťou je využitie hardvéru a pridanie zosilňovača. Avšak treba brať ohľad aj na maximálny výkon, ktorý ponúka daný výstupný port.

5.4.9 Logy

Reprezentujú textové súbory, kde jednotlivé riadky reprezentujú záznamy. Začínajú časovým údajom vytvorenia, jeho dôležitosťou, názvu zdroja, kto ho zapísal a ukončené sú samotným textom. Terminál používa dva druhy lo-

gov. Zdrojom jedných z nich je samotná aplikácia a druhé sú na sledovanie stavu zariadenia. Po spustení operačného systému sa okrem spustenia aplikácie spúšťajú aj dva ďalšie skripty na sledovanie teploty procesora, jeho vyťaženie a zaplnenie operačnej pamäte RAM. Tieto informácie môžu byť dôležité v prípade, že by sa zariadenie začalo chovať neočakávane. Všetky súbory s logmi sa ukladajú do užívateľského adresára s podpriečinkom "Logs".

5.4.9.1 Systémové logy

Zapisujú sa pomocou skriptov, ktoré v pravidelných intervaloch zapisujú sledované hodnoty do súborov. Každá zapísaná hodnota má v súbore vlastný riadok, ktorý začína časovou stopou vytvorenia, informáciou čo sleduje a jeho hodnotou.

5.4.9.2 Logy z aplikácie

Obsahujú štyri úrovne dôležitosti. Ich implementáciu zabezpečuje QT aplikácia automaticky. Avšak všetky z nich smeruje v textovej podobe do okna terminálu bez časovej stopy a ďalších metadát. To spôsobuje že po jeho zavretí alebo reštartovaní zariadenia sa stratia. Preto po spustení aplikácie je implementované nahradenie pôvodnej obsluhy. Funkcia novej obsluhy zabezpečuje zaobalenie správy podľa popisu aby začínala časovým údajom vytvorenia, nasledovali ďalšie metadáta a ukončené samotným textom správy. Následne po vytvorení textového záznamu, sa táto správa uloží do súboru s logmi. Pre každý deň je definovaný nový súbor s menom skladajúcim sa z troch častí. Prvý je text *AppLog* nasledovaný dátumom *yyyy-MM-dd* a ukončený číslom portu. Tieto tri časti sú navzájom oddelené pomlčkou a ukončené príponou *.log* definujúcou druh súboru. Úrovne logov v QT aplikácie, zoradené od najnižšej dôležitosti:

- Debug
- Warning
- Critical
- Fatal

V prvom prípade je obsah správy len informačný a oznamuje stav aplikácie. Ďalšie dva už naznačujú neočakávané správanie, ktoré môže viesť k pádu programu. Preto sa obsah týchto správ v kópii posiela aj aplikačnému serveru. Posledným a najzávažnejším je Fatal. V tomto prípade už aplikácia nedokáže ďalej pokračovať. Preto sa program ešte pokúsi o zapísanie chyby, ktorá to spôsobila a následne sa zavrie. Vďaka vytvorenému skriptu je zabezpečené znova spustenie s dúfaním, že sa do rovnakého stavu, ktorý spôsobil pád aplikácie nedostane.

5.4.10 Kopírovanie softvéru na nové zariadenie

Keďže operačný systém obsahuje mnohé úpravy, inštalácie a nastavenia. Je nepraktické tento postup opakovať pri každom novom terminály. Vďaka použitiu rovnakého mikropočítača vo všetkých termináloch, nie je potrebná žiadna zmena v ovládačoch. Ako operačnú pamäť používa microSD kartu, ktorú keď pripojím k inému mikropočítaču sa správa takmer rovnako. Rozdiel je len v adresách periférii. Ako napríklad MAC adresa internetového pripojenia. To ale nemá za následok žiadnu zmenu vo funkčnosti. Z toho vyplýva, že ak vytvorím presnú kópiu pamäte, môžem s rovnakými nastaveniami spustiť viac zariadení. Keďže karta obsahuje veľké množstvo systémových súborov, ktoré môžu byť aj skryté a samotná pamäť je rozdelená do niekoľkých partícii. Nestačí spraviť kópiu súborov, ktoré obsahuje ale vyžaduje sa jej binárna kópia. To znamená prejsť celú pamäť a preniesť všetky bity presne v rovnakom formáte bez zisťovania významu. Aby som pri vytváraní novej kópie nepotreboval opakovane čítať dáta zo starej karty. Čo by proces spomaľovalo a vyžadovalo disponovať s pôvodnou kartou a dvomi čítačkami kariet. Existujú programy, ktoré vytvorenie obrazu pamäti do *.iso* súboru zabezpečia. Následne sa tento súbor môže zapísať na nové zariadenie.

5.4.10.1 Problémy pri kópii

Pri kopírovaní obrazu pamäti na novú pamäťovú kartu je potrebné zabezpečiť, aby mala rovnaký alebo väčší počet bitov k zápisu. Keďže vytváram bitový obraz. Program nevie význam jednotlivých bitov či je to prázdna pamäť alebo reprezentujú dáky súbor. Preto ak vytvorím obraz 16Gb karty, na ktorej je jeden súbor o veľkosti 1Gb. Výsledný obraz bude obsahovať 16Gb, kde väčšina bude reprezentovať prázdne miesto. Očakával som, že každá karta s kapacitou 16Gb bude mať rovnakú veľkosť a spomenutý problém nebudem musieť brať do úvahy. Bohužiaľ použil som dve karty s rovnakou kapacitou ale od iného výrobcu a tento problém nastal, kde na novej karte chýbalo pár bitov aby sa tam obraz zmestil.

Rozhodol som sa pre používanie karty s veľkostou 16Gb, kde cenový rozdiel od 8Gb nie je veľký. Vytvorený program spolu s operačným systémom zaberá okolo 3Gb avšak 4Gb by mi nestačilo, keďže potrebujem aj priestor na zápis logov. Z toho vyplýva, že minimálna veľkosť karty, ktorú môžem použiť je spomenutých 8Gb. Aby som v budúcnosti nebol obmedzený pamäťou, rozhodol som sa pre spomenutých 16Gb a nákup kariet od jedného výrobcu.

Každé zariadenie musí obsahovať údaje, ktoré sú špecifické. Sú to väčšinou nastavenia spojené s pripojením do lokálnej siete, vďaka ktorým budem schopný rozlíšiť termináli medzi sebou. Takými sú IP adresa a doménové meno – hostname. V prípade, že používam DHCP automatické priradenie IP adresy, nie je potrebná žiadna ďalšia konfigurácia. Dokonca ani meno zariadenia neprekáža ak bude rovnaké s ostatnými. Jedine to zabraňuje jeho použitiu, keďže

nebude unikátne. Preto sa samozrejme odporúča nastaviť doménové meno. V prípade použitia statickej IP je potrebné ich nakonfigurovať manuálne podľa prevádzky. V tomto prípade nestačí len adresa IP, ale aj jej maska a východzia brána, voliteľne DNS adresa prípadne spomenuté doménové meno hostname.

5.4.11 Správa zariadenia

Operačné systémy založené na Linuxovom jadre sú založené na ovládaní pomocou príkazovej riadky. Ak je zariadenie pripojené do siete internetu, je možné sa naň pripojiť pomocou vzdialenej príkazovej riadky a využívať všetky funkcie, ktoré ponúka. Tieto systémy už v základe ponúkajú pripojenie cez protokoly *TELNET* a *SSH*. Jediným rozdielom v *SSH* je, že využíva šifrované spojenie, ktoré je nemožné dešifrovať v rozumnom čase ako je vysvetlené v [8]. V oboch prípadoch na pripojenie vyžadujú IP adresu zariadenia, aby bolo možné ho nájsť v sieti a následne prihlasovacie údaje obsahujúce meno a heslo užívateľa. Pri použití operačného systému Windows, môžem využiť napríklad aplikáciu *Putty*, ktorá podporuje oba protokoly. V prípade Linuxového operačného systému je podpora natívna z príkazovej riadky. Stačí zadať názov protokolu a IP adresu terminálu.

Na prenos súborov z alebo do terminálu sa využívajú protokoly *FTP* a *SFTP*. Rovnako ako pri pripojení príkazovej riadky, jediný rozdiel medzi nimi je, že *SFTP* ponúka prenos, ktorý je šifrovaný. Zvyšné nastavenia sú rovnaké a k pripojeniu potrebujem IP adresu terminálu a prihlasovacie údaje. Avšak v tomto prípade sa vyžaduje použitie programu na prenos v prípade oboch operačných systémov. Tými sú napríklad *FileZilla*, *WinSCP* alebo *TotalCommander*. Všetky spomenuté ponúkajú podporu oboch protokolov.

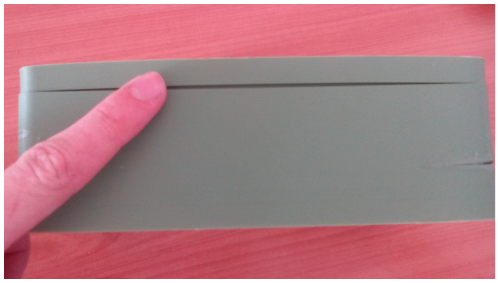
Pre zvýšenie zabezpečenia systému, je na termináloch zakázané pripojenie bez šifrovania a zmenené prednastavené prihlasovacie údaje. Preto k pripojeniu je potrebné využiť protokoly *SSH* prípadne *SFTP*.

5.4.12 Zapúzdrenie

Dizajn mikropočítača nie je určený na použitie v prevádzke. Disponuje veľkým množstvom konektorov a neobsahuje obal. Preto by sa zariadenie mohlo v prevádzke poškodiť. Ja som k nemu pripojili ešte displej a teda potrebujem vytvoriť krabičku, ktorá obe tieto komponenty zakryje. Požiadavkami ako už bolo spomenuté je dizajn, pevnosť a čo najväčšia odolnosť voči teplu a vode. Vďaka mojim skúsenostiam s 3D tlačou, ktorú som počas magisterského štúdia aj študoval sa pokúsim začať s touto alternatívou.

Displej ponúka uchytenie mikropočítača na jeho zadnú stranu a tiež disponuje 4 bodmi na uchytenie skrutkami. Tie som sa pokúsil využiť a vytvoriť obal podľa rámiku displeja.

Ako som spomínal na tlačenie sa používajú dva druhy plastov. Avšak rozmery displeja sú na tlač dosť veľké a ako som očakával, tlač z plastu ABS, ktorý



Obr. 5.12: Chyba tlače z ABS plastu

je odolnejší bol problém. Ako je zobrazené na obrázku 5.12, sa pre rozťažnosť ABS materiálu po vytlačení a počas chladenia deformoval. Následne jednotlivé časti do seba nezapadajú a vznikajú medzery. Taktiež je možné vidieť rozliepanie sa krajov krabičky v zahnutých častiach.

Týmto usudzujem, že použitie ABS plastu nebude vhodné v našom prípade a sústredím sa na menej odolnú ale lepšie spracovateľnú alternatívu.

5.4.12.1 Jednostranný displej

Krabičku som rozdelil na dve časti, kde jedna z nich sa spája s displejom a obsahuje ochranný rámik okolo neho. Následne druhá časť tvori zadný kryt, vďaka ktorému je umožnený prístup k mikropočítaču. Ich spojenie zabezpečujú 4 skrutky po krajoch. Vďaka mäkkosti plastu je možné do neho vyvŕtať závit. Tým vznikla použiteľná jednostranná krabička. Jej uchytenie v prevádzke je možné uchytením zadného krytu na stenu a následné spojenie prednej časti s displejom do zadnej.

5.4.12.2 Obojstranný displej

Výsledkom je krabička s displejom na oboch stranách, preto nie je možné použiť koncept z jednostrannej varianty, kde sa zadná časť použila ako otvor. Spoločným je upevnenie mikropočítača a displeja vďaka pripraveným úchytom. Preto krabičke stačí vymodelovať uchytenie a spojenie displejov. Podľa pôvodného zariadenia, ktoré firma používala, využívali uchytenie na tyč. Preto je krabička navrhnutá ako dva rámiky okolo displeja, kde jedného obvod je o čosi menší a vsúva sa do väčšieho. Na uchytenie má krabička na spodnej strane dieru, ktorá bude využitá na spojenie s držiakom a privedenie potrebných káblov do vnútra krabičky.

5.4.12.3 Test teploty

Problém plastového materiálu je teplota a následne jeho pevnosť v priestoroch so zvýšenou teplotou. Očakávaná teplota v prevádzke by nemala presiahnuť

Obr. 5.13: Testovanie v akváriu

40°C. Túto teplotu by s rezervou mal výrobok zvládnuť, avšak dá sa očakávať, že je to už zvýšená teplota a jeho vlastnosti sa môžu meniť.

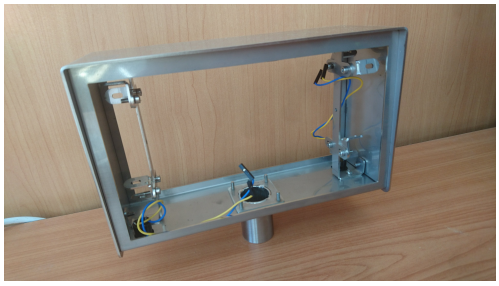
Na testovanie som si zabezpečil akvárium. Do jeho vnútra umiestnil ohrievač, misku s vodou na vytvorenie vlhkosti a terminál s pripojením do elektriky aj lokálnej siete pre komunikáciu s aplikačným serverom. Pre zvýšenie teploty som ešte upchal všetky diery, cez ktoré by mohla unikať para a teplo. V prvom teste som terminál nezaťažoval a posielal paket na obnovenie okna každých 30s. To by nemalo spôsobiť veľkú záťaž na procesor. Následne som zariadenie nechal v akváriu cez víkend. Teplota sa pohybovala okolo 60°C s nízkou vlhkosťou. Avšak stačilo to na deformáciu krabičky. Po trojdňovom teste procesor pracoval bez problémov, avšak krabička už nebola schopná zariadenie udržať. Fotografia z testovania je zobrazená na obrázku 5.13.

V čase fotografie sa tam súčasne testovali dva obojstranné zariadenia. Jedno bolo z PLA plastu, ktorý nezvládol teplotu a podľahol značnej deformácii. A druhé odolnejšie z ABS materiálu vydržalo bez deformácie avšak nezvládol proces tlačenia, ktorá nebola dostatočne dokonalá na to aby bola použitá v prevádzke.

Po neúspešnom teste ostali aspoň skúsenosti a akceptovateľný 3D návrh krabičky. Realizácia terminálu začala byť ohrozená. Po dlhom hľadaní sa zistilo, že smerom 3D tlače nemôžem pokračovať, keďže neponúkajú ďalšie možnosti. Jedna z firiem, ktorá mala záujem vyrobiť krabičku, vyrábala z plechu pomocou ohýbania. Nedokázali zvárať, preto všetky spojenia sú za pomoci skrutiek. Z toho vyplýva, že ani odolnosť voči vlhkosti tam nie je zabezpečená, keďže väčšina hrán je spravená dotýkaním sa plechu o plech. Dokonca nedokázali vytvoriť ani rovnaký priemer zahnutia ako má displej, preto výsledný dizajn nebol dostačujúci na použitie. Výsledok ich práce je na obrázku 5.14

Tretou firmou na ktorú som dostal kontakt sa zaoberá výrobou z nehrdzavejúcej oceli antikor. Po oslovení boli ochotný vytvoriť prototyp. Dodal som im súbor s 3D modelom, vytlačenú krabičku a potrebný hardvér. Oni sa následne postarali o návrh podľa ich dostupnej techniky a zručností. Výsledok

Obr. 5.14: Krabička od EMCO



Obr. 5.15: Oceľová obojstranná krabička

je zobrazený na obrázku 5.15. Vďaka pevnosti oceli sa nie je dôvod obávať o deformáciu. Súčasťou krabičky je aj držiak, ktorý sa následne nastokne na tyč v prevádzke. Vďaka jeho okrúhlemu tvaru, je displej možné otáčať okolo vlastnej osi. V porovnaní materiálu s plastom som si všimol aj veľký rozdiel v pasívnom chladení. Bez potreby inštalovať aktívne prvky sa teplota procesora mikropočítača za rovnakej prevádzky znížila v priemere o 9°C. To sa dalo očakávať, keďže plast je izolant a na opačnú stranu oceľ je dobrým vodičom tepla. Jedinú nevýhodu, ktorú v tomto materiáli vidím je jeho vodivosť. Je dobrým vodičom tepelnej ale aj elektrickej energie. To môže viesť k skratom v prípade, že sa niektoré z vodičov dotknú krabičky. Našťastie sa do krabičky privádza len jednosmerné napätie v bezpečnom rozsahu do 50V. A aj v prípade že by sa dostalo na povrch krabičky nehrozí žiadne zranenie v prípade dotknutia sa.

5.4.13 Simulátor

Vďaka vývoju v multiplatformovom vývojovom prostredí QT a využívaním taktiež multiplatformového programovacieho jazyka C++ je možné aplikáciu spustiť pod operačným systémom Linux aj Windows. Kde obe tieto aplikácie vyzerajú takmer rovnako. Avšak pôvodná aplikácia sa spúšta v režime celej obrazovky, aby zamedzila otváranie a používanie iných aplikácii na termináli. Toto je nežiadúce v prípade simulátora. Preto je implementovaný voliteľný

parameter „e“, ktorý spustí program v režime emulátora. Teda v okne o rozmeroch rozlíšenia 7 palcového displeja. Okrem toho sa u simulátoru využíva parameter, na zmenu portu. Vďaka ktorému dokážem súčasne spustiť viacej simulátorov súčasne na jednom zariadení na rozdielnych portoch a tým testovať aplikačný server bez potreby mať dostupný terminál.

5.5 Dátová vrstva pre C# aplikácie

Je rovnako implementovaná v programovacom jazyku C# ako zvyšné dve aplikácie, ktoré ju budú využívať. Tým zabezpečím najlepšiu kompatibilitu. Na spoluprácu obsahujú referenciu na tento projekt a tým im ponúka svoje funkcie. Obsahom dátovej vrstvy je implementácia jednej triedy. Tá v sebe skrýva funkcie na získanie dát z databázy. Návratová hodnota funkcii je zvyčajne objekt, reprezentujúci jeden záznam v tabuľke. V prípade viacerých objektov sú zaobalené v zozname. V databáze sa nachádzajú tri relácie s rovnakou štruktúrou. Jedná sa o polievky, hlavné jedlá a dezerty, ktoré obsahujú rovnaké parametre. Preto sú reprezentované spoločným rodičom s názvom *MenuItem*, ten obsahuje všetky tieto parametre. Tým je uľahčená možnosť rozšírenia menu o viac druhov jedál. Príklad implementácie je na obrázku 5.16. Okrem parametrov definuje rodičovská trieda aj funkciu *ToString*. Tá sa používa v prípade, že chcem objekt reprezentovať textovo napríklad u zápisu logov. Ako z implementácie je vidieť, po zavolaní tejto funkcie sa objekt zobrazí v tvare názvu jedla nasledovaný podrobným popisom.

Okrem dedenia využívam aj vnorovanie objektov. Kde objekt ako napríklad Menu v sebe obsahuje objekty polievka, hlavné jedlo a dezert. Tie sú definované ako pole tried *MenuItem* o dĺžke tri, zobrazené na obrázku 5.17.

Definíciu objektov som v štruktúre projektu vložil do priečinka *Data*, kde každý objekt je v samostatnom súbore a triedu zabezpečujúcu komunikáciu umiestnil do hlavného adresára pod menom *DatabaseLayer.cs*. Ako bolo spomenuté využívam databázu MySQL. K vytvoreniu dotazov na databázu potrebujem knižnicu MySQL.Data.dll. Na príklade 5.18 je príklad volania databázovej procedúry s parametrom „count“ typu celé číslo. V prvej časti sa vytvorí a nadviaže komunikácia so serverom, kde sa použije text z premennej *CONNECTION_STRING* reprezentujúci adresu servera a prihlasovacie údaje. Ďalej sa vytvorí MySQL dotaz s typom procedúry a textom názvu procedúry. Následne sa k dotazu pridajú všetky nevyhnutné parametre na spustenie dotazu. V našom prípade je to parameter *Count*, ktorého hodnotu nastavím z lokálnej premennej. Keďže databáza je strikne typová, je potrebné zadať aj typ premennej. Avšak pri použití funkcie *Parameters.AddWithValue* sa typ získa z lokálnej premennej, ktorá je parametrom.

Rovnakým konceptom sú vytvorené aj zvyšné funkcie na získavanie dát z databáze. Knižnica MySQL ponúka okrem volaní funkcii aj vykonanie dotazu priamo nad tabuľkami. V takom prípade sa typ dotazu nastaví na textový a

```
namespace DataLayer.Data
{
  public class MenuItem
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }

    public override string ToString()
    {
      return $"{Name}: {Description}";
    }
  }

  public class Soup : MenuItem
  {
  }

  public class Meal : MenuItem
  {
  }
  public class Desert : MenuItem
  {
  }
}
```

Obr. 5.16: Implementácia druhov jedál

```
using System;
namespace DataLayer.Data
{
  public class Menu
  {
    public int IdMenu { get; set; }
    public DateTime ForDate { get; set; }
    public MenuItem[] Items {get;set;} = new MenuItem[3];
  }
}
```

Obr. 5.17: Trieda reprezentujúca jedálniček

```
using MySql.Data.MySqlClient;
public IEnumerable<Order> GetServedOrders(int count)
{
  // pripojenie sa na server
  string CONNECTION_STRING = "server=localhost;userid=
      root;password=root;database=cvutdb"
  using (MySqlConnection connection = new MySqlConnection
      (CONNECTION_STRING))
  {
    connection.Open();
    // vytvorenie dotazu na databazu
    using (MySqlCommand command = connection.
      CreateCommand())
    {
      command.CommandText = "GetServedOrders";
      command.CommandType = System.Data.CommandType.
      StoredProcedure;
      command.Parameters.AddWithValue("Count", count);
      // vykonanie dotazu
      using (MySqlDataReader reader = command.
      ExecuteReader())
      {
        // postupne spracovávat prijaté záznamy
        while (reader.Read())
        {
          yield return new Order();
        }
      }
    }
  }
}
```

Obr. 5.18: Volanie MySQL procedúry

textová podoba dotazu sa zapíše do *CommandText*. Pri vykonávaní textového dotazu nie sú potrebné parametre a ich pridávanie sa vynecháva.

5.6 Aplikačný server

Proces operačného systému, ktorý dokážem nastaviť aby sa automaticky spustil po zapnutí počítača alebo servera sa stará o obsluhu terminálov. Obsluhu terminálov budem volať aj podaplikáciou, keďže pre každý terminál je vytvorená nová obslužná trieda v novom vlákne počítača. Tým sa veľmi podobá samostatnej aplikácii spustenej hlavným programom aplikačného servera. Na prístup k dátam sa v implementácii využívajú služby dátovej vrstvy, ktorá bola vysvetlená v kapitole 5.5.

```
<RLI Ver="1.0.0.0">
 <Init StartKey="96" CurrentTime="2018-01-01
    12:05:02.243">
 <StyleSheet>
  QLabel, QPushButton { color: black; }
  QWidget.tableBgStyle { background:rgb(40, 40, 40); }
  QWidget.tableBgStyle QWidget.tableCell { background:rgb
    (175, 167, 218); }
  QWidget.lightPanelStyle { background:rgb(100, 90, 170); }
 </StyleSheet>
 </Init>
</RLI>
```

Obr. 5.19: Definovanie štýlu aplikácie v XML

5.6.1 Spustenie

Podľa implementačných požiadavkách sa po spustení systémovej služby nadviaže komunikácia s terminálmi a prebehne medzi nimi úvodná inicializácia. Textová podoba inicializačného paketu v smere z aplikačného servera do terminálu je zobrazená na obrázku 5.19. V prvom riadku je úvodný tag oznamujúci smer správy. V našom prípade je to „RLI“, teda smer do terminálu. Nasledovaný je verziou XML-ka. Následne vo vnútri tagu je telo inicializácie. Ako parameter nesie číslo znaku, ktorý je používaný ako štartovací znak čítacej hlavy. Ďalším parametrom je systémový čas. Ten je potrebný na synchronizáciu času terminálu a servera. Synchronizácia je potrebná v prípade poruchy. Každé zariadenie si počas práce zapisuje logy. Tie obsahujú aktuálny čas a stav v akom sa nachádza. Keby čas servera a terminálu sa nezhodovali. Dochádzalo by k problému spojiť logy jednotlivých zariadení a tým reverzne vyčítať čo sa v systéme dialo.

Na každej obrazovke terminála je zobrazený čas vo formáte HH:MM. Jeho pravidelné obnovenie je každých 30s, čo využívam aj na kontrolu, či zariadenie správne odpovedá a komunikuje so serverom.

5.6.2 Komunikácia s terminálom

Aplikačný server obsahuje triedu, ktorá reprezentuje jedno zariadenie Raspberry Pi a pre každé je vytvorená nová inštancia tejto triedy. Táto trieda vyžaduje pri inicializácii definovať obsluhu, ktorá ovláda daný terminál. Aby v prípade prijatia udalosti priloženia karty alebo stisku tlačidla z terminálu, dokázala zavolať príslušnú obsluhu. Ďalšie nevyhnutné parametre sú identifikácia terminálu v sieti. Tými sú IP adresa a číslo portu. Vďaka tomu sa po zavolaní funkcie Connect dokáže pripojiť na zariadenie a zahájiť jeho ovládanie. Táto funkcia v sebe obsahuje samotné vytvorenie TCP/IP komunikátoru, kde nastaví obslužnú funkciu pre prijatie správy a pre odosielanie správ vytvorí

```
1  public void SendMessage(IMessage message)
2  {
3    lock (_queue)
4    {
5      // uloženie správy do zásobníka
6      _queue.Enqueue(message);
7    }
8    // oznámenie o novej správe
9    _semaphore.Release();
10 }
```

Obr. 5.20: Pripravenie správy do zásobníka na odoslanie

nové vlákno, ktoré sa bude starať o odosielanie paketov. Týmto postupom je zabezpečená obojstranná komunikácia.

Ovládanie terminálu pozostáva z posielania paketov. Každý paket v sebe obsahuje udalosť, reprezentujúca inštrukcie, čo má terminál vykonať. Ich odosielanie je zabezpečené funkciou *SendMessage*. Ako argument vyžaduje triedu typu *IMessage*, ktorá obsahuje zoznam udalostí. Táto funkcia celú správu uloží do zásobníka a prenechá samotné odosielanie vláknu, na to určenom. Keďže zásobník správ je zdieľaná pamäť, je potrebné toto volanie zapuzdriť do kritickej sekcie. Na záver ešte upozorniť vlákno, ktoré odosiela správy pomocou semaforu, že má v zásobníku novú správu. Táto implementácia je na obrázku 5.20.

V prípade nového vlákna, ktoré sa stará o odosielanie paketov je v nekonečnej slučke. Tá začína pasívnym čakaním na novú správu na odoslanie. Čakanie je ukončené pri vytvorení správy na odoslanie alebo vypnutí aplikácie. To je zabezpečené pomocou _terminatedEvent, kde po jeho prijatí je nekonečná slučka ukončená spolu s volaním funkcie. V prípade že pasívne čakanie bolo prerušené na základe vytvorenia správy v inom vlákne, prejde do kritickej sekcie zdieľanej pamäti, prevezme správu a odošle ju. Ako som spomínal každá správa obsahuje niekoľko udalostí, tie sú postupne spracované a vložené do prislúchajúceho paketu. Na záver každý paket obsahuje funkciu GetData, ktorá vracia bajtovú reprezentáciu udalosti v požadovanej XML podobe. Poslednou časťou, potrebnou na odoslanie dát je využitie funkcie Send u TCP/IP komunikátora, ktorý podporuje odosielanie len bajtov. Vďaka triede reprezentujúcej paket som potrebné dáta získal a môžem ich odoslať. Vďaka TCP/IP protokolu, ktorý sa stará o odoslanie paketu je zaručený príjem na druhej strane a v prípade chyby prenosu je automaticky preposlaný.

5.6.2.1 Reprezentácia udalosti

Každá udalosť je reprezentovaná vlastnou triedou. Tá v sebe obsahuje potrebné parametre, ktoré využíva pri vytváraní XML príkazu pre terminál. Jeden z príkladov je zobrazenie obrazovky. Jeho implementácia je zobrazená

```
namespace ServerApp.Devices.Actions
{
  public class ShowLayoutAction : IAction
  {
    public ShowLayoutAction(string layoutName, params
      ModifyLayoutItem[] modifyLayoutItems)
    {
      ModifyLayoutItems = modifyLayoutItems;
      LayoutName = layoutName;
    }
    public string LayoutName { get; }
    public ModifyLayoutItem[] ModifyLayoutItems { get; }

    public override string ToString()
    {
      return $"ShowLayout {LayoutName} ({
       ModifyLayoutItems.Length} modifiers)";
    }
  }
}
```

Obr. 5.21: Implementacia udalosti zobrazenia obrazovky

na obrázku 5.21. Ako je vidieť na príklade, konštruktor triedy vyžaduje názov rozloženia, ktoré bude zobrazené a jeho úpravy v podobe zoznamu ModifyLayoutItem. V druhej časti je predefinovaná funkcia ToString. Vďaka nej získam porozumiteľný log aká obrazovka je zobrazená a skrátene aspoň počet úprav, ktoré sa na ňu aplikujú. Aby bolo možné pomenovať všetky udalosti jedným meno, dedia všetky rozhranie IAction. Následne, keď chcem pracovať všeobecne s ktoroukoľvek akciou, použijem tento spoločný názov.

5.6.2.2 Reprezentácia paketu

K odoslaniu udalostí do terminálu je potrebné ich zabaliť do triedy paketu ktorá obsahuje spôsob textového kódovania a názov hlavného tagu XML správy. V našom prípade je to kódovanie UTF-8 a tag s názvom „RLI“, keďže je to v smere do terminálu. Trieda paketu ponúka ďalšiu funkcionalitu, potrebnú k odoslaniu. Zabezpečuje správne kódovanie textu podľa interného nastavenia a vytváranie XML tela správy s využitím názvu tagu. Túto funkčnosť má funkcia s názvom GetData. Už bola spomenutá, že vytvára bajtový retazec reprezentujúci obsah správy na odoslanie. Jeho implementácia je na obrázku 5.22.

Najskôr si vytvorí potrebné pole bajtov, ktoré neskôr odošle. Následne pomocou nastaveného kódovania pretransformuje začínajúci tag do bajtov a pridá na začiatok poľa. Za ním nasleduje telo správy, ktoré sa získava pomocou *GetXmlCommand* definované príslušnou triedou na základe jej druhu a

```
1  Encoding = Encoding.GetEncoding("UTF-8");
2  _baseTag = "RLI";
3  public byte[] GetData(Action action)
4  {
5    List<byte> bytes = new List<byte>();
6
7    // otvaraci znak XML spravy
8    byte[] prefix = Encoding.GetBytes($"<{_baseTag}>");
9    bytes.AddRange(prefix);
10
11   // telo XML správy
12   byte[] command = Encoding.GetBytes(action.
         GetXmlCommand());
13   bytes.AddRange(command);
14
15   // uzatvárajúci znak XML správy
16   byte[] suffix = Encoding.GetBytes($"</{_baseTag}>");
17   bytes.AddRange(suffix);
18
19   return bytes.ToArray();
20 }
```

Obr. 5.22: Vytvorenie XML správy z objektu udalosti

ukončím správu zatváracím tagom aby som zabezpečil jej správnosť.

Každý druh paketu vyžaduje implementáciu funkcie *GetXmlCommand*. V opačnom prípade nie je možné vytvoriť textovú podobu príkazu. Keďže každý paket má inú štruktúru, počet argumentov a výstupný tvar. Na jeho vytvorenie využívam knižnicu *System.Xml*. Tá pomocou *XElement* vytvorí XML tag podľa definovaného mena a ako parametre pridá zoznam atribútov definovaných pomocou *XAtribute*. Následne sa tieto triedy môžu ľubovoľne kombinovať. Jeden ukážkový príklad je zobrazený na 5.23. V tomto prípade som vytvoril hlavný element s názvom *elem1* a pridal mu textovú a číselnú podobu atribútu. Následne do neho som pridal ďalší element *elem2*. Po vytvorení štruktúru zanorenia sa po zavolaní funkcie *ToString*, ktorá vytvorí textovú podobu objektu zobrazí jeho reprezentácia pomocou XML značkovacieho jazyka. Tento spôsob som využil aj ja na vytváranie tela paketov.

5.6.2.3 Spracovanie priatych udalosti

Pomocou funkcie, ktorá je volaná pri každej novej správe prijatej od klienta je možné zabezpečené jej okamžité spracovanie. To pozostáva zo spracovania XML textu a transformácia do príkazov. Každá korektná správa podľa definovanej štruktúry v tomto dokumente musí byť zaobalená RLO tagom. Jeho podoba v prípade začiatku správy je $<RLO>$. Toto spojenie piatich znakov je vyhľadané a označené za začiatok správy. Tým preskočím znaky pred správou, ktoré do nej nepatria a nemusia podliehať správnemu XML formátu. Z prípadu

```
XElement root2 = new XElement("elem1",
      new XAttribute("a1", 123),
      new XAttribute("a2", "text"),
        new XElement("elem2"));
Console.WriteLine(root2.ToString());
```

— Výstup —

```
<elem1 a1="123" a2="text">
  <elem2 />
</elem1>
```

Obr. 5.23: Použitie XElement v C#

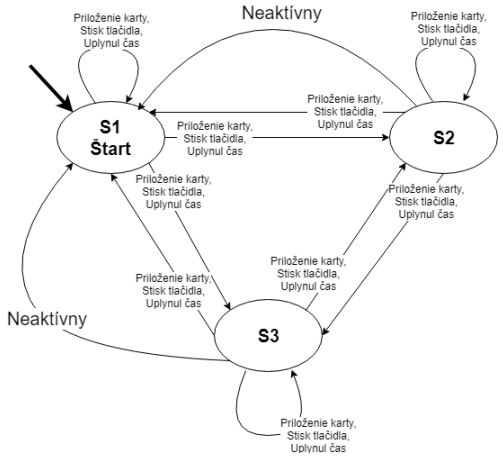
veľkej správy sa môže stať, že systém ju rozdelil na dve a odoslal po častiach. Preto v druhom kroku si program skontroluje či správa obsahuje aj párový ukončujúci znak v podobe *</RLO>*. V takom prípade je prijatá celá správa od terminálu a môže byť spracovaná. Ak sa tento tag nenachádza. Program si uloží prvú časť správy a pri prijatí nových dát ich spojí so starými a vyhľadáva začiatočný a ukončujúci tag rovnakým spôsobom.

Spracovávanie správy po jej kompletnom prijatí je zabezpečené triedou *QTPacketParser* umiestnená v stromovej štruktúre projektu pri implementácii komunikácie s terminálom. Tá obsahuje funkciu *CreateAction*, ktorá vytvára zoznam akcii z korektne definovanej XML správy. Na jej spracovanie používam podobne ako je na vytváranie XML správy knižnicu System.XML. V tomto prípade program používa *XmlReader*, ktorý číta vstupný text a z neho vytvára objektovú reprezentáciu. Ako koreňový tag sa očakáva „RLO“ značka. Keďže je to výstupná správa z terminálu a následne zoznam udalostí s ich parametrami. Medzi udalosti, ktoré môžu nastať na strane terminálu patria:

- ButtonClick
- CardRead
- Error

Okrem týchto parametrov obsahuje každá správa aj presnú časovú stopu, kedy nastala podľa systémového času terminálu. Aj pre toto je dôležitá synchronizácia časov servera a klienta počas inicializácia. Spracovávanie prebieha postupne pomocou volania funkcie *Read* v triede *XmlReader*. Tá postupne číta telo správy až sa dostane na jej koniec. V každej jeho iterácii sa kontroluje názov udalosti. V prípade, že sa zistí zhoda, prečítajú sa jej parametre a vytvorí sa jej inštancia. Zoznam inštancii akcii sa na konci funkcie vracajú ako návratová hodnota a spracovávanie je ukončené.

Po spracovávaní sa dostáva beh programu naspäť, do miesta kde sa spracovával prijatý text. Avšak teraz už disponuje jeho objektovou podobou a



Obr. 5.24: Ukážka stavového automatu

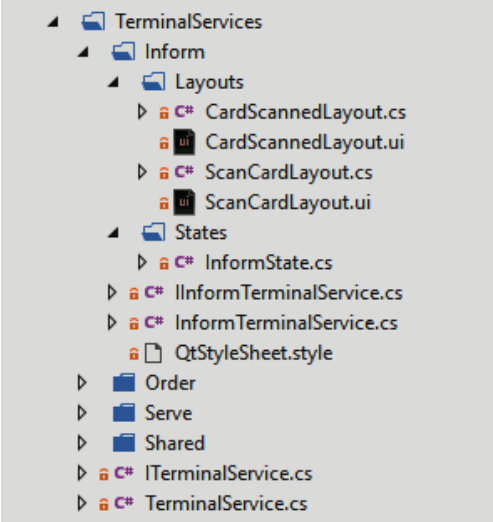
obsahuje tiež ukazovateľ na obsluhu terminálu. Preto postupne prechádza jednotlivé akcie a vykonáva ich obsluhu podľa aktuálneho stavu terminálu.

5.6.3 Štruktúra obsluhy terminálu

Obsluha terminálu je implementovaná deterministickým konečným automatom. Podľa knižky teórie automatov [9] definuje päticu premenných, kde v našom prípade:

- $\sum$ – vstupná abeceda – vypršanie časového limitu, priloženie karty alebo stisk tlačidla, ktoré vyvolajú prechodovú funkciu
- **K** – konečná množina stavov
- **q0** – počiatočný stav – je jeden z množiny **K** určený funkciou *GetInitState*
- δ – prechodová funkcia – v našom prípade je definícia prechodovej funkcie implementovaná v počiatočnom stave. Obsahuje obsluhu $\sum$ na základe ktorej vypočíta nasledujúci stav. Okrem toho automat rozširuje prechodové funkcie o prechod do stavu nečinnosti v prípade straty spojenia s terminálom
- **F** – množina akceptovaných stavov. V našom prípade rovnaká s množinou **K**

Ukážkový konečný automat pre tri stavy S1 až S3 s počiatočným stavom aj stavom nečinnosti v S1 so zobrazením všetkých možných prechodov je znázornený na obrázku 5.24. Zakreslený automatu je nedeterministický a slúži



Obr. 5.25: Stromová štruktúra súborov projektu

ako ukážka, aké prechody môžeme v obsluhe implementovať. Nedeterministickosť je napríklad zo stavu S1, kde je možné ísť po priložení karty do stavu S1, S2 aj S3. Avšak táto funkcia záleží na implementácii, kde sa jednoznačne určí nasledujúci stav.

V prípade vynechania implementácie prechodu je preddefinovaný prechod tiež deterministický s rovnakým počiatočným a koncovým stavom. V prípade implementácie môžeme využiť pri výpočte koncového stavu číslo karty, názov stlačeného tlačidla a stav databázy na presné určenie koncového stavu. Takže pre konkrétny automat s implementáciou bude vždy jednoznačne definovaný prechod a tým získavam deterministický automat.

Na obrázku 5.25 je zobrazená štruktúra obsluhy Informačného terminálu. Umiestnená je v priečinku „TerminalServices“ pod názvom „Inform“, kde obsahuje hlavné nastavenia.

Rozhranie *ITerminalService* opisuje všeobecný automat a jeho funkcie, ktoré musí implementovať. Patria sem funkcie na definíciu počiatočného stavu, stavu nečinnosti, získanie aktuálneho stavu, prechodovú funkciu a udalosti, ktoré sú potrebné na inicializáciu s novým terminálom pri nadviazaní spojenia. Rozhranie je ďalej rozšírené o špecifické funkcie pre daný automat definované v *IInformTerminalService*. Implementácia je rovnako rozdelená na všeobecnú, ktorá je spoločná pre všetky automaty a umiestnená v triede *TerminalService* rozšírená o špecifické funkcie vyžadované informačným automatom v *InformTerminalService*.

Zdieľaná implementácia automatu v súbore *TerminalService* sa stará o dátovú vrstvu, názov obsluhy, opakovanú kontrolu aktívnosti stavu, volanie prechodových funkcii, automatický prechod do neaktívneho stav v prípade

straty spojenia s terminálom a tiež obsahuje informáciu o aktuálnom stave.

Následne implementácia automatu a samotnej obsluhy obsahuje zoznamy terminálou potrebných k pripojeniu, rozloženia obrazoviek používaných počas behu programu, definíciu počiatočného stavu a stavu nečinnosti. K nej je ešte pridaný súbor *QtStyleSheet*, ktorý obsahuje štýl pre terminál. Aby obrazovka bola správne zobrazená. Ten sa do zariadenia posiela v tele inicializačného paketu.

5.6.3.1 Stavy automatu

Sú umiestnené v pod priečinku „States". Na ich identifikáciu je potrebné aby dedili z koreňovej triedy *StatesBase*. Tá zastrešuje všeobecnú obsluhu funkcii potrebných k behu automatu. Avšak každý stav si ich môže ľubovoľne predefinovať. Okrem toho má možnosť si uchovávať akékoľvek informácie, kým sa systém nachádza v danom stave. Avšak po prechodovej funkcii do iného stavu sa všetky interné dáta strácajú. Funkcie používané stavovým automatom, ktoré má užívateľ možnosť ovplyvňovať sú spracovávanie udalostí, ktoré môžu byť priloženie karty, stlačenie tlačidla na obrazovke, vypršanie časovača, výskyt chyby a funkcie volané pri prechode do stavu a odchode z neho. Vypršanie časovača a tým vytvorená periodicky volaná funkcia sa využíva na obnovenie dát zobrazovaných na obrazovke ako je napríklad aktuálny čas. Vďaka aktuálnemu času zobrazovanom na displeji je patrné či terminál funguje, prípadne kedy prestala reagovať na správy z aplikačného servera. Ďalšia možnosť funkcie je vynútiť prechod do iného stavu. To sa však v tomto projekte nenachádza a možno nájde uplatnenie pri rozširovaní funkcionality.

5.6.3.2 Prechodová funkcia

Dá sa rozlíšiť od ostatných funkcii vďaka návratovej hodnote. Tá je typu *IStateBase*. Keďže koreňový stav triedy *StateBase* od neho dedí, je možné funkciou vrátiť ktorýkoľvek stav daného automatu. Ak funkcia vracia inakší stav ako je ten v ktorom sa aktuálne systém nachádza, vyvolá to prechod automatu a za ním skryté volanie funkcie odchodu z pôvodného stavu a funkcie príchodu do nového stavu. V tele funkcie je možné volanie databázových funkcii ale aj iné časovo náročnejšie výpočty na základe ktorých deterministicky vypočíta prechodovú funkciu a nasledujúci stav do ktorého automat prejde. V prípade, že nechcem vykonať žiadny prechod, je možné vrátiť seba samého pomocou príkazu „return this".

5.6.3.3 Rozloženie obrazovky

Každá definícia obrazovky musí obsahovať dva súbory. Jeden s príponou .ui definuje XML súbor so samotným rozložením. Tento súbor dokáže otvoriť a editovať vývojové prostredie QT a umožňuje jednoduchú úpravu pomocou grafického návrhu obrazovky a funkciou chyť a pusti(Drag and Drop).

```
1 public IEnumerable<ModifyLayoutItem> SetDateTimeTo(
      DateTime dateTime)
2 {
3   yield return new ModifyLayoutItem("DateValue", "text",
      dateTime.ToString("d"));
4   yield return new ModifyLayoutItem("TimeValue", "text",
      dateTime.ToString("t"));
5 }
```

Obr. 5.26: Vytvorenie objektov upravujúcich čas a dátum na obrazovke

Následne sa tento grafický návrh zapíše do spomenutého súboru s príponou .ui a vývojár aj bez znalosti XMLka vytvorí reprezentáciu obrazovky pomocou QT vývojového prostredia.

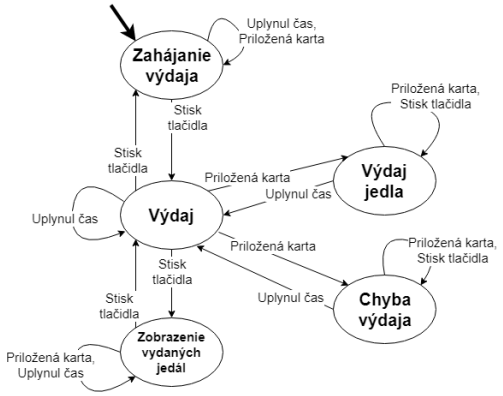
Druhý súbor je implementovaný v jazyku C# a slúži k ovládaniu danej obrazovky čím sa rozumie nastavovanie zobrazených textov, farieb prípadne iných vizuálnych efektov na obrazovke. Keďže na ovládanie obrazovky sa používajú *ModyfyLayou* objekty, preto práve tie sú návratovou hodnotou funkcii v tomto súbore. Ako bolo spomenuté najjednoduchším ukazovateľom či terminál je aktívny je zobrazenie aktuálneho času. Preto v každom okne aplikácii je čas spolu s dátumom zobrazený v pravom hornom rohu. Implementácia funkcie na nastavenia času je zobrazená na obrázku 5.26. Ako parameter príma aktuálny čas a následne vytvorí dva objekty reprezentujúce zmenu na obrazovke pre *DateValue* a *TimeValue* s atribútom *text* na požadovanou hodnotou aktuálneho času a dátumu.

5.6.3.4 Spôsob vytvorenia novej obsluhy terminálu

Každá jedáleň a stravovacie zariadenie môže mať vlastné nároky alebo nápady na vylepšenie. Keďže terminál je vytvorený univerzálne a dokáže zobraziť užívateľom definované rozloženie textov a tlačidiel, tak záleží len na aplikačnom serveri a implementácii obslužnej časti automatu akú funkciu bude vykonávať a čo zobrazovať. Pre vytvorenie novej funkcionality je v projekte už pripravená obsluha terminálu na odosielanie správ a všeobecná implementácia automatu a stavov. Aby som zachoval súborovú štruktúru a prehľadnosť programu, vytvorím si nový priečinok s názvom novej obsluhy respektíve automatu v *TerminalServices*. V nej si vytvorím triedu implementujúcu funkcie automatu, jeho stavy a rozloženia obrazoviek, ktoré budem používať. Následne po implementácii stavov, ich prechodov a chovania je ešte potrebné pridať záznam o novej obsluhe do databázy a môže sa používať.

5.6.4 Implementácia obsluhy terminálov

Každý stravovací systém či jedálna k svojmu fungovaniu potrebujú výdajný terminál, ktorý môže fungovať na princípe okamžitého výdaja. K nemu môžu



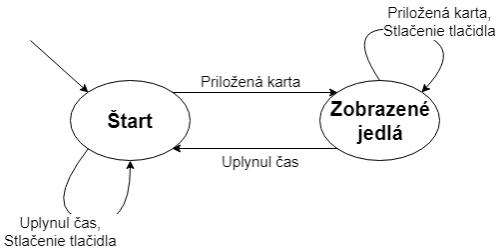
Obr. 5.27: Stavový automat výdajného terminálu

používať systém objednávania, podľa ktorého je navrhnutý aj systém v tejto práci. Preto okrem výdajného terminálu je nevyhnutné implementovať aj objednávací terminál. S ich kombináciou by už jedáleň dokázala fungovať. Následne som diplomovú prácu rozšíril aj o informačný terminál, na ktorom si klienti môžu prezerať vytvorené objednávky. V nasledujúcich kapitolách je vysvetlené ako implementované obsluhy terminálu fungujú. Obrázky displejov terminálov z prevádzky jednotlivých automatov sú súčasťou príloh tejto práce.

5.6.4.1 Výdajný automat

Pozostáva z dvoch zariadení s displejmi, ktoré môžu používať obojstrannú krabičku a hardvérovo vyzerať ako jedno zariadenie, alebo sa môžu použiť dve krabičky s jednostranným displejom. Toto rozhodnutie záleží na jedálni a ich priestoroch. Aby bola zabezpečená komunikácia s oboma zariadeniami, v databáze sa musia k nemu priradiť dva záznamy a oba identifikovať pomocou IP adresy a portu. Obsluha automatu si ku každému z nich vytvorí vlastnú reprezentáciu, ktorá je implementovaná v automate a po jej spustení sa vytvorí vlákno pre každé z nich. To sa stará o komunikáciu s nim. Vďaka použitia viacerých vlákien nie sú prechodové funkcie automatu spomaľované komunikačnými výpadkami a čakaním na odoslanie správ.

Na obrázku 5.27 je zobrazený stavový diagram. Obsahuje päť stavov. Po zapnutí obsluhy automatu sa výdaj jedál automaticky nezapne. Automat začne v stave zahájenia výdaja a obsluha musí manuálne spustiť výdaj, keď je všetko pripravené. Po spustení výdaju prejde automat do stavu Výdaj, kde pri interakcii klienta priložením karty sa mu môže jedlo vydať, alebo nastať chyba. Tou môže byť neobjednané jedlo alebo neznáma karta. Počas výdaju, môže obsluha pozastaviť výdaj a prejsť do štartovacieho stavu, alebo zobraziť vydané jedlá. V oboch prípadoch sa výdaj jedál pozastavuje a ignoruje



Obr. 5.28: Stavový automat informačného terminálu

Obr. 5.29: Zobrazenie objednaných jedál

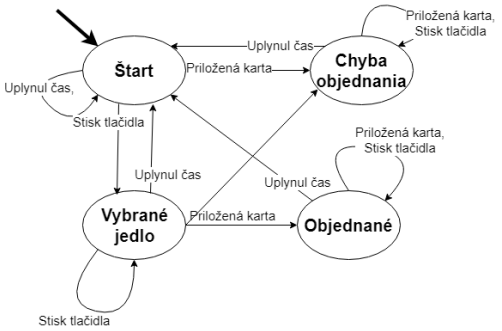
priloženie kariet.

5.6.4.2 Informačný automat

Po zapnutí obsluhy je potrebné spojenie s jedným terminálom, ktorý umožňuje zobraziť nevydané objednávky na displeji. Funkčnosť vyžaduje aby terminál disponoval čítačkou kariet a na druhú stranu nevyžaduje dotykovú plochu.

Stavový diagram obsluhy je zobrazený na obrázku 5.28, obsahuje dva stavy. Ani jeden z nich nevyužíva interakciu s užívateľom pomocou stlačenia tlačidla, ktorú neimplementuje. V prípade, že by táto udalosť nastala, automat ostáva v pôvodnom stave a javý sa ako by danú udalosť ignoroval. Prvý zo stavov je stav nečinnosti. Zobrazuje aktuálny čas a očakáva priloženie karty. V prípade jej priloženia prechádza automat do druhého stavu. V prípade rozpoznaného klienta, zobrazí jeho objednávky. Ak karta nie je registrovaná, zobrazí upozornenie o neznámej karte. Následne sa automat vracia do stavu nečinnosti po uplynutí časového intervalu.

Dizajn aplikácie je definovaný dvomi XML súbormi. Kde prvý je použitý počas stavu nečinnosti s informáciou „Priložte kartu“ a po priložení karty v prípade, že zobrazenie objednávok klienta sa nepodarilo tak zobrazí mu chybovú hlášku na rovnakej obrazovke. Druhé zobrazenie obsahuje meno klienta, jeho stav na účte a zoznam objednaných jedál. Jeho ukážka je na obrázku 5.29.



Obr. 5.30: Stavový automat objednávacieho terminálu

5.6.4.3 Objednávací automat

Rovnako ako informačný automat vyžaduje jedno zariadenie s čítačkou kariet. To však musí disponovať aj dotykovou plochou. Terminál využíva interakciu s užívateľom, ktorý si musí v prvom kroku vybrať jedlo z ponuky dotykom na obrazovke. Následne po priložení karty sa systém pokúsi o jeho objednanie. V prípade úspešnej objednávky sa zobrazí informácia „Objednané“. Objednávka sa samozrejme môže nepodariť. Dôvodom môže byť neplatná karta alebo nedostatok financií na účte klienta. Stavový automat zobrazený na obrázku 5.30 obsahuje štyri stavy. Počiatočným stavom je zobrazenie klientovi správu o pripravenosti systému, v prípade výberu jedla sa stav zmení a jedlo sa na obrazovke označí. V prípade priloženia karty k čítačke sa vykoná prechod. Ten záleží, či objednávka prebehla v poriadku. V takom prípade je ďalším stavom úspešne objednané jedlo. a po uplynutí časovača sa automat vráti do počiatočného stavu. V prípade neúspechu systém získa informáciu, ktorá spôsobila chybu, automat prejde do chybového stavu, zobrazí chybovú správu a po uplynutí časovača sa vráti do štartovacieho stavu. Tam sa klient môže opäť pokúsiť o vytvorenie objednávky.

Obsluha objednávacieho terminálu počas behu programu využíva len dve obrazovky. Jedna zobrazuje zoznam dostupných jedál k objednaniu, kde po kliknutí na jedno z nich sa označí. Využíva ho štartovací stav a stav s označeným jedlom. Druhým je obrazovka na zobrazenie správy klientovi. Tá je zdieľaná medzi chybovým a úspešným stavom. Rozdielna je len informácia, ktorá sa na displeji zobrazí.

5.7 Aplikácia pre správu systému

Potreba stálych zmien, vývoja a správy systému vyžaduje vytvorenie aplikácie, ktorá bude sledovať jeho stav a udržiavať bezchybný chod stravovacieho zariadenia. Pre intuitívnosť aplikácie je použité grafické prostredie. To je implementované pomocou XML-ka a frameworku *Windows Presentation Founda-*

tion. Použitý je programovací jazyk C# a už existujúca databázová vrstva ku komunikácii s databázou. Obrázky aplikácie sú obsiahnuté v prílohách.

5.7.0.1 Podaplikácie

Zobrazuje zoznam vytvorených podaplikácii v prevádzke. Po dvojitom kliknutí sa otvorí nové okno reprezentujúce podaplikáciu obsahujúcu názov, typ a zoznam pripojených terminálov s ich IP adresami a portami. Každú obsluhu terminálu dovoľuje program aktivovať alebo deaktivovať aby sa nemusel podaplikácia v prípade potreby vypnutia odstraňovať.

5.7.0.2 Správa terminálov

Umožňuje SSH a SFTP pripojenie na terminál. Vďaka tomu dokáže spravovať systémové nastavenia terminálov ako je IP adresa, doménové meno, aktualizácia programu, stiahnutie logov a sledovanie stavu.

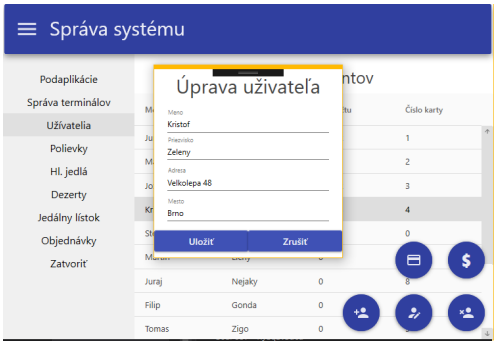
Ak by správca systému chcel využiť inú funkcionalitu cez SSH alebo SFTP, ktorá nie je implementovaná. V okne správy terminálov je možnosť automatického otvorenia a pripojenia sa na terminál cez protokol SSH s programom *Putty*, a cez protokol SFTP s programom *WinSCP*. Ich integrovanie do aplikácie je povolené, keďže v aplikácii nemodifikujem ich zdrojové kódy.

5.7.0.3 Stravníci

Zoznam zobrazujúci všetky dostupné informácie o užívateľoch systému obsahujúce jeho interný identifikačný údaj, meno, priezvisko, stav na účte a číslo karty. Tabuľkové zobrazenie umožňuje užívateľov zoraďovať či vyhľadávať medzi nimi. Podľa návrhového vzoru *Material Design* [10], pôvodne smerovaného pre Android zariadenia, sú zobrazené tlačidlá v pravom dolnom rohu s možnosťou vytvoriť, upraviť, vymazať stravníka a nastaviť mu bezkontaktnú kartu s vkladom na účet v riadku nad nimi. Ukážka obrazovky je na obrázku 5.31. V prípade úpravy alebo vytvorenia nového stravníka sa otvorí nové okno aplikácie s detailnými informáciami o ňom. Obsahuje jeho meno, priezvisko, adresu a mesto. Jednoducho sa dajú rozšíriť o rodné číslo či iné potrebné údaje v prípade potreby. Avšak v tomto smere treba byť opatrný v prípade ochrany osobných údajov. Ďalšou možnosťou je zmena bezkontaktnej karty. Kde sa otvorí nové okno a aplikácia čaká na jej načítanie. Poslednou funkciou ktorú môže obsluha vykonávať je vklad alebo výber peňazí z účtu. Rovnako sa obsluhe zobrazí užívateľské meno a s dialógom na vloženie sumy, ktorú chce na kartu vložiť. V prípade zadania záporného čísla znamená výber z karty.

5.7.0.4 Jedlá ktoré sa v jedálni varia

Každé menu môže obsahovať tri druhy jedál. V našom prípade sa volajú polievka, hlavné jedlo a dezert. Aby sa mohol vytvoriť jedálny lístok. Je po-

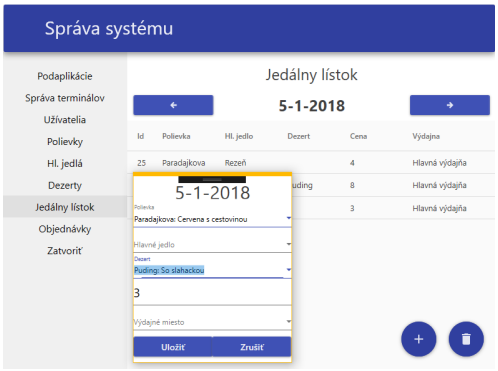


Obr. 5.31: Aplikácia správy systému - zobrazenie užívateľov

trebné mať toto jedlo v ponuke. Všetky tri druhy majú rovnakú štruktúru. Obsahujú identifikačné číslo, meno a podrobný opis. Po kliknutí v menu na jeden z nich sa zobrazí jeho zoznam. Tam je možné jedlo pridať, upraviť alebo vymazať. Keďže jedlo môže byť použité v jedálnom lístku, tak operácia vymazať sa z pohľadu databáze nemôže vykonať. Keďže je viazaná na existujúce menu. Preto je to v aplikácii zabezpečené skrytým parametrom „vymazané“. V prípade, že ho obsluha vymaže, v skutočnosti sa len označí za vymazané a skryje zo zoznamu. Taktiež ho už nebude možné vybrať pri vytváraní nového jedálneho lístka. U modifikácii jedla nie je povolená zmena jeho mena a povolené je zmeniť len jeho opis. Ak obsluha zmení opis jedla, spôsobí to zmenu vo všetkých jedálnych lístkoch aj objednávkach klientov, ktoré sú s ním prepojené. Preto pri tejto operácii treba byť opatrný.

5.7.0.5 Jedálny lístok

Zobrazí jedálny lístok na aktuálny deň. Pomocou šípok v hornej časti dovoľuje prechod medzi dňami. Na spodnej strane obrazovky ponúka tlačidlá k vytvoreniu novej alebo vymazanie aktuálneho záznamu z objednávok. Rovnako ako pri jedlách sa pri vymazaní neuskutoční vymazanie z databázy, ale sa len označí skrytým príznakom. Aby v prípade, že už bola vytvorená objednávka naň, dala sa dohladať. V prípade vytvorenia novej objednávky sa vytvára na deň, ktorý je vybratý a otvorí sa nové okno. Tam užívateľ vyberie, čo bude daný záznam obsahovať a jeho cenu. Medzi jedlami sa mu ponúkajú polievky, hlavné jedlá a dezerty, ktoré sa v prevádzke varia. Po pridaní sa automaticky zobrazí záznam s novou ponukou medzi alternatívami v objednávacom termináli a umožní klientom jeho objednanie. Obrázok s jedálnym lístkom je zobrazený na



Obr. 5.32: Aplikácia správy systému - nastavenie jedál. lístka

5.7.1 Objednávky

Systém myslí aj na zábudlivých stravníkov. Každému sa môže stať, že si zabudne svoju bezkontaktnú kartu. V takom prípade je možné v záložke objednávky vyhľadať požadovaného klienta a vytvoriť mu novú objednávku, alebo vydať už existujúcu objednávku. Taktiež umožňuje vymazávať a prezerať, ktoré objednávky si objednal ale neboli mu vydané prípadne o koľkej si jedlo prevzal. Keďže objednávky sú zobrazené riadkovo, aplikácia umožňuje ich filtráciu a zoraďovanie.

5.7.1.1 Správa zariadenia Raspberry Pi

V okne správy aplikácii, je potreba definovať IP adresu zariadenia s ktorým idem pracovať. Následne v ďalšom riadku vyberiem funkciu z rolovacieho okna, ktorú chcem použiť a na základe funkcie vyplním voliteľné argumenty. Ako napríklad pri sťahovaní logov je potrebné zadať umiestnenie v počítači, kam sa majú súbory uložiť. Na komunikáciu s mikropočítačom som využil knižnicu *Renci.SshNet*, ktorá mi ponúka možnosť vzdialeného pripojenia.

Použitie knižnice a vzdialená práca so zariadením Terminál podporuje len šifrované protokoly k pripojeniu. Tie vyžadujú prihlasovacie údaje k zariadeniu ako je meno, heslo a identifikáciu zariadenie pomocou IP adresy. Na príklade 5.33 je zobrazené vytvorenie informácii o spojení, ktoré obsahujú všetky tieto údaje. Následne pomocou *SshClient* vytvorím spojenie a pripojím sa. Nad vytvoreným spojením je možné vytvárať príkazy, kde ako parameter je jeho textová reprezentácia. Funkciou *Execute* sa príkaz vykoná a textový výstup z príkazu je vrátený danou funkciou. Po ukončení práce cez SSH je potrebné odpojenie. V prípade prenosu súborov môžem použiť rovnaké informácie pripojenia. Ale namiesto triedy *SshClient* využijem triedu *SftpClint* zabezpečujúcu SFTP pripojenie. Po pripojení môžem využívať jeho dve hlavné funkcie, ktoré sú *DownloadFile* na stiahnutie súboru a *UploadFile* na posla-

```
protected string ExecuteRemotly()
{
  string output;
  ConnectionInfo connInfo = new ConnectionInfo("192.168.0.10
      ", "login", new PasswordAuthenticationMethod("login", "
     heslo"));
  using (SshClient sshClient = new SshClient(connInfo))
  {
    sshClient.Connect();
    using (SshCommand cmd = sshClient.CreateCommand("cat
       ReadMe.md"))
    {
      output = cmd.Execute();
    }
    sshClient.Disconnect();
  }
  using (var client = new SftpClient(connInfo))
  {
    client.Connect();
    using (var localFileStream = File.Create(localFileName))
    {
      client.DownloadFile(remoteFileName, localFileStream);
      client.UploadFile(localFileMemoryStream,
       remoteNewFileName);
    }
    client.Disconnect();
  }
  return output;
}
```

Obr. 5.33: Použitie knižnice Renci.SshNet

nie súboru do zariadenia. V oboch prípadoch je potrebné definovať súbor v lokálnom počítači a na vzdialenom zariadení.

S využitím SSH protokolu na vzdialené vykonávanie príkazov sú implementované nastavovania operačného systému mikropočítača Raspberry Pi

- Doménové meno
- Použitie DHCP servera
- IP adresu
- IP adresu východzej brány
- DNS adresu

Modifikácie nastavení sa neaplikujú okamžite, ale je potrebné zariadenie reštartovať. To je taktiež možné vzdialene, cez príkazovú riadku. Aplikácia vykonáva reštart vždy automaticky, preto si treba dať pozor na správnosť údajov. V prípade nesprávne zadaných hodnôt sa môže stať, že mikropočítač

stratí spojenie do lokálnej siete a nebude sa dať vzdialene ovládať. V takom prípade je nevyhnutné sa priamo pripojiť na zariadenie pomocou USB klávesnice a opraviť nastavenia systému. Na pripojenie sa k USB je potrebné otvorenie krabičky, keďže USB porty sú umiestnené vo vnútri. Do druhej skupiny nastavení patria funkcie, ktoré vyžadujú prenos súborov a s tým spojené pripojenie cez SFTP protokol:

- Sťahovanie logov
- Aktualizácia softvéru v zariadení

Keďže sa jedná o presúvanie súborov, aplikácia vyžaduje od užívateľa väčšiu interakciu. V prípade sťahovania logov je potrebné zadať umiestnenie v počítači, kam sa stiahnuté súbory majú uložiť. U aktualizácie softvéru je potrebné zadať cestu v počítači, kde je uložená binárna podoba programu a o zvyšok sa program postará sám.

V prípade aktualizácie softvéru musí aplikácia na pozadí vykonať niekoľko menších krokov, keďže SFTP nepodporuje priame nahradzovanie starého súboru. Preto sa aktualizácia skladá z troch fáz. V prvej fáze vymaže pôvodný program pomocou SSH, následne nakopíruje nový program cez SFTP a na záver, aby bolo možné program spustiť sa pomocou SSH nastaví jeho mód ako spustiteľný. Rovnako ako u nastavovaní IP adresy, na aplikáciu zmien je potrebný reštart zariadenia. Pri novom spustení operačného systému sa už spúšťa s novým binárnym súborom a na obrazovke si môžem všimnúť novú verziu programu.

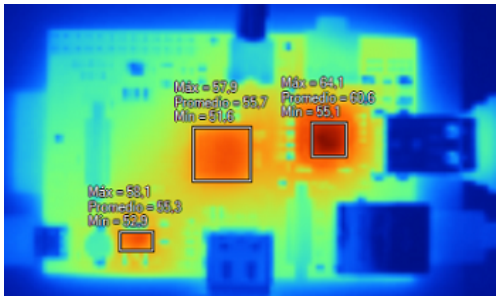
Kapitola 6

Testovanie, nasadenie a prevádzka

6.1 Terminál

Záťažové testy kontrolujú správanie zariadenia v prípade maximálnej záťaže, to som zabezpečil vytvorením aplikácie, ktorá každú sekundu posiela inštrukciu do terminálu na prekreslenie celej obrazovky. Ak by posielala inštrukciu na zobrazenie stále rovnakej obrazovky, interný softvér terminálu by to rozpoznal a ignoroval. Preto každá správa obsahuje inú obrazovku spolu s ďalšími dátami na úpravu. Ako som si všimol, mikropočítač je vstavaný na výrazne vyšší výkon a problém s prekresľovaním nemá. Dokonca jeho využitie procesora sa pohybovalo do 30%. Menej prijemná situácia bola s jeho teplotou. Tá sa dokázala vyšplhať až na 67°C. Podľa výrobcu je beh zariadenia odporúčaný v rozmedzí 0-70°C. To síce spĺňa, ale budem si dávať na teplotu pozor v prevádzke po nasadení. Operačný systém mikropočítača obsahuje tiež kontrolu hardvéru a dokáže zobraziť upozornenie v pravej hornej časti displeja. V prípade poklesu napätia sa zobrazí blesk, pri teplote 80-85°C teplomer s ortuťou v polovici a pri prekročení 85°C sa zobrazí rovnaký teplomer ale s ortuťou až po vrch. Okrem mikropočítača sa v krabičke nachádza aj micro SD karta. Tá je tiež zariadením, ktoré vyžaduje správne pracovné prostredie. Preto som prešiel z pôvodných micro SD kariet na tie, u ktorých výrobca zaručoval vyššiu pracovnú teplotu. Dokonca počas testovania sa stalo, že karta sa zlomila a stala sa nečitateľnou. Dôvod mi nie je známy a usudzujem, že sa to mohlo stať na základe zmeny teploty a rozťažnosti materiálu.

Počas dlhodobého testovania sa stávalo, že operačný systém postupne vypínal služby až vypol aplikáciu terminálu. Následne sa už dostal do prevádzkového režimu. Ako som zistil problém bol vo vymazávaní premenných, kde postupne zapĺňalo operačnú pamäť. Počas vývoja som tento problém nezbadal, keďže sa prejavoval v priemere po 5tich dňoch pri štandardnej obsluhe. Avšak pri použití zátažového testu, kde je potrebné stále prekreslovanie obra-



Obr. 6.1: Teplota mikropočítača Rapsberry Pi 3

zovky sa chyba dala spozorovať už po niekoľkých hodinách. Našťastie oprava tejto chyby netrvala dlho. Podarilo sa ju opraviť a pokračoval som ďalej s testovaním pri použití opraveného softvéru.

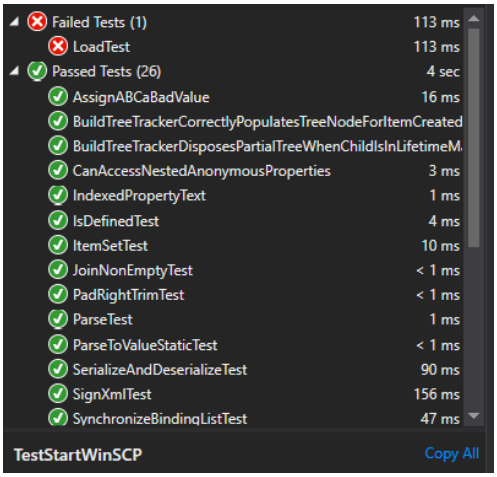
Ohľadom teploty mikropočítača som sa dozvedel, že hlavným zdrojom prehrievania sú stabilizátor napätia, procesor a radič internetového pripojenia. Dokonca z obrázku 6.1 sa dá vidieť, že najvyššiu teplotu dosiahol spomínaný radič, umiestnený na fotke na pravej strane. Nasledoval procesor v strede obrazovky a najnižšiu teplotu dosiahol stabilizátor napätia zvýraznený malým obdĺžnikom v ľavom dolnom rohu. Vďaka kompatibilite mikropočítačov rady Raspberry Pi verzie 2 a 3 som pri testovaní otestoval oba z nich. Informácie z internetu sa overili. U novšej verzie s označením 3 je prehrievanie väčším problémom a pri rovnakých podmienkach sa rozdiel teploty v uzatvorenej krabičke po jednom dni používania líšil o približne 10°C.

Vďaka skúsenostiam z testovania som poskladal výsledné riešenie terminálu z oceľovej krabičky, zabezpečujúcej lepšie pasívne chladenie a z mikropočítača Raspberry Pi verzie 2. Tým som sa dostal na prijateľnú teplotu v prevádzke, ktorá je v rozmedzí od 40 do 50°C. To by malo zabezpečovať bezproblémové fungovanie terminálov a celého systému.

6.2 Aplikačný server

Vývoj, to je len začiatok života programu. Ešte pred jeho samotným nasadením je potrebné jeho testovanie aby sa predišlo čo najviac problémom. Projekt obsahuje niekoľko testov a zainteresovaný boli aj samotný budúci užívatelia. Tí mali možnosť si aplikáciu vyskúšať, navrhnúť úpravy prípadne odhaliť chyby.

Unit testy určené ku kontrole funkcionality malých častí alebo funkcii. Vývoj programu prebiehal vo vývojovom prostredí Visual Studio, ktoré v sebe obsahuje podporu pre tento druh testovania. Dokáže vytvoriť unit testy, spustiť ich jednotlivo, po skupinách alebo všetky a následne zobrazuje priebeh,



Obr. 6.2: Unit testy

ktoré úspešne skončili a kde nastala chyba. Taktiež podporuje ladenie testov rovnako ako sa ladia programy. V prípade chyby zobrazí miesto výskytu, zanorenie programu a hodnoty jednotlivých premenných. To uľahčuje hľadanie chýb. Hlavnou výhodou je v prípade pokračovania vývoja. Ak by sa funkcionalita rozširovala, tak unit testy mi zaručia, že program nezmenil svoje správanie v existujúcej funkcionalite. V aplikačnom servery sú tri podskupiny Unit testov. Prvá je zameraná na databázovú vrstvu. Kontroluje získavanie dát z databázy.

Ďalšia skupina testov je zameraná na testovanie automatov. Kontroluje prechodové funkcie a zmenu stavov na základe vyvolanej akcie. V inicializačnej fáze testu sa vytvorí virtuálna databázová vrstva a nastaví sa automat na daný stav. Vtedy test vyvolá udalosť, tá pri spracovávaní môže využívať databázovú vrstvu. Avšak dáta z nej sú kontrolované testom a po ukončení prechodovej funkcie sa kontroluje stav, v ktorom sa automat nachádza. Týmto spôsobom viem izolovať priebeh funkcie od vplyvov aplikácie či databázy a zamerať sa výlučne na potrebnú funkcionalitu.

Poslednou skupinou je funkčnosť, ktorá sa stará o vytváranie paketov. Tá dostáva na vstup udalosť, napríklad zobrazenie obrazovky a výstupom sa očakáva textový súbor reprezentujúci danú funkčnosť v XML formáte. V tomto prípade program nepotrebuje spoluprácu s ďalšími objektami ako je databáza a výsledok je výlučne v implementácii danej triedy a použitej knižnice. Po spustení testu sa inicializuje udalosť, ktorá má byť pretransformovaná. Vyvolá sa potrebná obsluha a keďže test vie, akú udalosť očakáva, porovnáva výstupný text so vzorovým. V prípade zhody končí úspechom. Výsledky unit testov sú zobrazené na obrázku 6.2.

Integračné testy Kontrolujú komunikáciu a spojenie medzi menšími časťami programu. Väčšinou tými, ktoré boli testované Unit testami. Ich využitie je hlavne vo väčších spoločnostiach, kde sa na vývoji softvéru podieľa niekoľko tímov. V takom prípade by kontrolovali spoluprácu medzi celkami, ktoré vytvorili rôzne vývojové tými.

V našom prípade môžem kontrolovať integritu medzi troma skupinami, ktoré boli rozdelené v Unit testovaní. Projekt obsahuje niekoľko testov, ktoré kontrolujú integráciu databázovej vrstvy s automatom. Tým implementácia je veľmi podobná tej, ktorá kontrolovala automat. Avšak v tomto prípade sa miesto virtuálnej databázovej vrstvy použije reálna. Problém v tomto prípade vzniká v kontrole výsledku. Pri spúšťaní testov väčšinou vieme, čo by som mal na výstupe dostať. Avšak v tomto prípade výstup záleží na stave databázy. Preto v inicializačnom kroku testov sa inicializuje aj databáza a nastavuje sa v závislosti na danom teste. Tu si však tiež treba dávať pozor, aby sa predišlo spusteniu viacerých testov naraz. To by spôsobilo viacnásobné inicializovanie databázy a výsledky by neboli správne. Tým by testy neuspeli aj napriek tomu že implementácia by bola správna.

Ďalšou možnosťou je integrácia automatu a odosielania paketov. V tomto prípade simulujem priloženie karty a očakávam paket zvukovej signalizácie na výstupe. Podobných možností je nespočetné množstvo a niektoré z nich sú implementované v projekte.

Smoke testy Opisuje test programu v celku, kde osoba, ktorá sa nemusí vyznať do problematiky sa pomocou návodu a intuície snaží program používať. Vďaka tomuto testu získa vývojár iný pohľad na systém, kde niektoré veci sa mu zdali samozrejmé ale v skutočnosti je to naopak. Okrem toho prináša aj otestovanie hlavných funkcii. Avšak test nejde príliš hlboký a ostáva pri najčastejšie sa vykonávaných akciách.

Keďže tento test už nepatrí do rady automatických a je potreba zainteresovať ľudí. Vybral som sa na Mendlovu univerzitu v Brne a oslovil pár študentov. Celkovo sa zapojili 5ti, kde traja z nich našli aj chybu programu. Tá sa následne samozrejme opravila. Pre detailnejšie testovanie by sa mali použiť ľudia rôznych generácii, zamerania či vzdelania.

Black box testy Sú veľmi podobné Smoke testom. Rovnako nie sú automatizované a je potrebné aby ich vykonával iný človek ako vývojár. Avšak v tomto prípade by mal byť znalý problematiky. Pred začatím testu sa program pripraví a spustí. Následne sa tester snaží detailne prejsť každú funkcionalitu, ktorú by dané zariadenie malo zabezpečovať. Z názvu vyplýva, že tester nemá prístup do vnútra programu, nepozná jeho štruktúru ani spôsob implementácie. Preto nemôže smerovať testy na pomyselné slabé miesta implementácie. K tomuto testovaniu som využil dvoch mojich spolužiakov z oboru Informatika. Pred začiatkom testu som im vysvetlil o čo ide a čo je úlohou

klienta a obsluhy. Následne skúšali objednávať, klikať po obrazovke, zahajovať a pozastavovať výdaj. Tým sa im podarilo odhaliť ďalších niekoľko chýb a tým ukončiť časť testovania programu. Ďalšie testovanie bude prebiehať pri rozšírení funkcionality alebo iného zásahu do zdrojového kódu.

6.3 Aplikácia na správu systému

Pozostáva z menších funkcii, ktoré predovšetkým komunikujú s databázovou vrstvou. Väčšina z nich je samostatná a nezávisí na iných celkoch. Preto integračné testy v tomto prípade nemajú význam. Miesto nich rozšírim automatické testovanie o testy grafického rozhrania.

Unit testovanie Keďže aplikácia neobsahuje veľkú funkcionalitu a väčšinou sa jedná o zobrazenie upravenie a následné uloženie dát. Tak unit testy obsahujú v sebe súčasne konvertovanie vložených dát z textu na čísla a volania databázových funkcii. Čím sa spoločne testuje aplikácia s dátovou vrstvou. Druhou oblasťou sú unit testy na správu zariadenia. Starajú sa o funkčnosť, nadviazanie spojenia a vykonanie potrebných inštrukcii na operačnom systéme v termináli. Keďže sú to jednoduché testy, každý obsahuje len jednu funkcionalitu, kde nastaví na zariadení napríklad IP adresu a následne ju skontroluje.

UI testovanie Visual Studio vo verzii Enterprise ponúka vytvorenie projektu na testovanie užívateľského rozhrania. Je potrebné ju doinštalovať, keďže nie je v základnom balíčku programu. Pridanie projektu je rovnaké ako u iných, ale typ projektu sa zvolí GUI Test. Ten vytvorí prázdnu testovaciu triedu. Tá sa od ostatných líši parametrom [Test Method]. Na vytvorenie GUI testu som použil generátor. Ten zaznamenáva prácu s aplikáciou a každú interakciu s nou. Po dokončení zadávania príkazov sa vygeneruje ich zoznam. V mojom prípade je zoznam príkazov zobrazený na obrázku 6.3. Vďaka označeniu metódy za testovaciu, sa v okne zoznamu všetkých testov zobrazia aj tieto a môžu sa opakovane spustiť. Po spustení testu je potrebné zapnúť testovanú aplikáciu a následne sledovať, ako sa myš hýbe po obrazovke a zadávajú sa nahrané údaje.

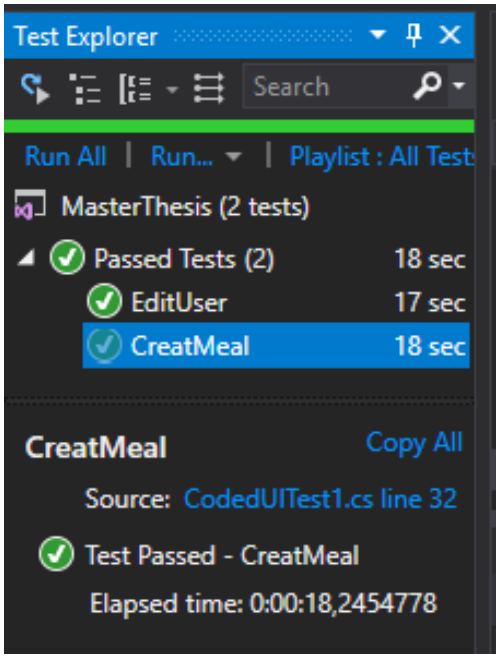
Výsledok testov je zobrazený v okne testov a označený červenou pre neúspešné, modrou nové testy, ktorých výsledok zatiaľ nie je známy. A poslednou možnosťou sú zelené úspešné testy. Výstupu testov grafického rozhrania je zobrazný na obrázku 6.4.

6.4 Význam testovania

Výhodou automatických testov je rýchle otestovanie. Pri každej zmene programu nie je problém ich opätovné spustenie. Síce sa zdá, že je to plytvanie časom na vytváranie testov a venovať čas tomu, čo je už hotové. Ale výhody sa

```
1  public void CreateMeal(string mealName, string descrtiption)
2  {
3    WpfButton uIHljedlaButton = this.UIMainWindowWindow1.
         UIHljedlaButton;
4    WpfButton uIPridatButton = this.UIMainWindowWindow1.
         UIFramePane.UIPridatButton;
5    WpfEdit uIItemNameEdit = this.UIMainWindowWindow1.
         UIItemNameEdit;
6    WpfEdit uIDescriptionEdit = this.UIMainWindowWindow1.
         UIDescriptionEdit;
7    WpfButton uIUlozitButton = this.UIMainWindowWindow1.
         UIUlozitButton;
8    WpfButton uIZatvoritButton = this.UIMainWindowWindow1
         .UIZatvoritButton;
9
10   Mouse.Click(uIHljedlaButton, new Point(105, 16));
11   Mouse.Click(uIPridatButton, new Point(38, 23));
12   uIItemNameEdit.Text = mealName;
13   Keyboard.SendKeys(uIItemNameEdit, this.
         CreateMealParams.UIItemNameEditSendKeys,
         ModifierKeys.None);
14   uIDescriptionEdit.Text = descrtiption;
15   Mouse.Click(uIUlozitButton, new Point(75, 13));
16   Mouse.Click(uIZatvoritButton, new Point(114, 23));
17 }
```

Obr. 6.3: Implementácia GUI testu



Obr. 6.4: Výsledok GUI testov

prejavia hlavne u väčších projektov a v početnejších vývojových tímoch. V mojom prípade mi Unit testy pomohli aj pri vývoji tejto aplikácie. Celý čas som používal lokálnu databázu nainštalovanú na vlastnom počítači s operačným systémom Windows. Následne ako bol program skoro hotový som sa rozhodol použiť vzdialenú databázu dostupnú pomocou verejnej IP adresy. Na tento server som nainštaloval operačný systém Linux s distribúciou Ubuntu. Po inštalácii som nastavil MySQL server a skopíroval dáta z lokálnej databázy. Po spustení aplikácie som nemal problém s pripojením ale žiadna z funkcii nefungovala. Problém bol v rozlišovaní veľkých a malých písmen. To nebol problém a rýchlo som funkcie prepísal na malé písmená. Aplikácia už fungovala bez problémov ale po pár dňoch, keď som pustil unit testy jeden z nich neprešiel. Jednalo sa o zabudnutý príkaz, ktorý obsahoval veľké písmená v názve tabuľky. Bola to funkcia, ktorá sa nepoužíva často, preto som ju neodhalil. Ale po spustení testov mi vývojové prostredie presne ukázalo kde sa chyba nachádza.

6.5 Odozva od používateľov

Úspešne sa podarilo časť systému nasadiť do prevádzky. Jedná sa o terminál v oceľovom zapuzdrení. Medzi spoločnosti, pre ktoré sa tento terminál nasadil patria Masarykova univerzita v Brne, Slovenská technická univerzita v Bratislave, prevádzka Slovnaft v Bratislave a Fakultná nemocnica Olomouc. Aby sa zabezpečila kompatibilita a komunikácia s ich ďalšími zariadeniami, sa použil z tejto diplomovej práce len terminál, ktorý zabezpečuje zobrazovanie a interakciu so stravníkmi a obsluhou. Následne upravili serverovú časť z pôvodnej, ktorá odosielala informácie do sedem segmentového displeja na komunikáciu pomocou XML správ a mojím terminálom. Tým sa môj terminál stal súčasťou ich komplexného systému, ktorý ponúka aj ďalšiu funkcionalitu ako sledovanie alergénov stravníkov, nákup potravín s uprednostnením regionálnych dodávateľov, prepojenie s ubytovacím systémom a ďalšie.

Pri prvej realizácii som asistoval a pomáhal k hladkému priebehu. Vďaka dobrým testom sa nevyskytli väčšie problémy a všetko sa dalo operatívne vyriešiť. Následne som niekoľko dní sledoval stav zariadenia a prevádzkovú teplotu. Ukázalo sa že všetko išlo v poriadku. Jediným problémom bolo neustále zapisovanie informácii o behu bez toho aby ich aplikácia premazávala. Napriek tomu, že je v zariadení inštalovaná 16 gigabajtová micro SD karta, bolo potrebné chybu opraviť a po pár mesiacoch aktualizovať ich starú verziu na novú, ktorá obsahuje argument počtu uložených logovacích súborov a ich spätné vymazávanie. Okrem toho sa vyskytli problémy ako nefunkčná zvuková signalizácia na jednom z dvoch zariadení v obojstrannom termináli. Následne niektoré zvukové signalizácie fungovali, iné čo boli posielané na problémový displej zas nefungovali. Táto chyba sa zisťovala dlhšie, keďže testovací program na zvukovú aplikáciu sa automaticky spúšťal na funkčnom displeji. A

najnovším problémom, ktorý sa dostal do vedomia až teraz je v prípade pádu aplikácie. U chyby Fatal nie je možné ďalej pokračovať. Preto ešte program vytvorí informáciu o chybe ale pred tým ako ju zapíše do súboru sa aplikácia zavrie. Aktuálne je už chyba opravená, avšak je ešte veľa zariadení, ktoré majú zastaralú verziu.

Obsluha po nasadení dotykovej obrazovky mala problémy s ovládaním. Ešte stále je veľa ľudí, ktorý nemajú skúsenosti s dotykovými zariadeniami. Sú zvyknutý na stláčanie tlačidiel miesto displeja. Dokonca bola sťažnosť, že mali problém zahájiť výdaj i keď sa na obrazovke zobrazovalo len jedno tlačidlo s nápisom štart, ktoré stačilo stlačiť. Ale to boli začiatky, pravdepodobne si obsluha na ovládanie zvykla a dokonca dostávam návrhy na vylepšenie či ďalšiu implementáciu funkcii, ktoré by chceli využiť.

Kapitola 7

Pokračovanie vývoja

V jednoduchosti je krása ale rozšírenie funkcionality je vždy vítané. Keďže som strávil na projekte veľa času, objavilo sa veľa možností, ktoré by sa dali použiť a pridať do projektu. Určite sa dajú rozšíriť informácie o užívateľovi, ktoré budú ukladané v databáze, rozšíriť počet druhov jedál, ktoré obsahuje menu. Ale to sú väčšinou rozšírenia závislé na požiadavkách danej jedálne a nevyžadujú veľa času na implementáciu. Avšak existuje veľké množstvo ďalších rozšírení, ktoré by boli použiteľné vo väčšine stravovacích jednotkách.

7.1 Tlačenie pokladničných blokov

Systém je možné využiť len v prevádzkach, kde nie je potrebné tlačiť potvrdenku o predaji jedla. To je možné z histórie získať a prípadne vytlačiť formou faktúry na konci mesiaca ručne. Ale je možnosť implementovať komunikáciu terminálu s pokladničnou tlačiarňou. Tá by sa tiež mohla využiť na výdaj jedál pri zabudnutí karty. Kde stravník by dostal lístok s objednávkami na daný deň z aplikácie na správu systému. A na základe lístka by mu v jedálni mohli vydať jedlo.

7.2 Terminály

Vďaka univerzálnosti zariadenie dokáže vykonávať akúkoľvek funkcionalitu. Avšak jeho veľkosť je obmedzujúca. Preto by určite bolo výhodou disponovať terminálmi s väčšími displejmi. Prípadne vytvorenie stavanej aplikácie a použitie zariadenia ako pokladňu. Na trhu je veľké množstvo tabletov a stolových obrazoviek obsahujúce funkčné zariadenie. Po softvérovej stránke väčšinou využíva Linux, Windows alebo Android. Vzhľadom na to použiť aktuálnu aplikáciu prípadne inú vstavanú aplikáciu ktorá by dokázala fungovať so systémom, databázou klientov a ich účtami. Mohla by to byť napríklad už spomínaná pokladňa v školskom bufete.

7.3 Výdaj neobjednaných jedál

Návrhom od aktuálnych používateľov výdajných terminálov je výdaj bez objednávky. Aktuálne ak chcú zákazníci dáke jedlo, je potrebné si ho najskôr objednať a následne vyzdvihnúť. Preto možnosť implementácie týchto dvoch zariadení do jedného, kde by pri výdaji mohol klient vybrať jedlo z ponuky a po priložení karty by sa mu objednalo a následne vydalo. Už teraz terminál obsahuje obe obrazovky dotykové, tak je škoda, že dotyk zo strany klientov sa nevyužíva. Toto by mohla byť jedna z možností uplatnenia.

7.4 Váha

Do populárnosti sa pomaly dostáva predaj jedál na váhu. Kde zákazník si sám naservíruje jedlo na tanier a následne podľa váhy sa mu vypočíta cena. Toto rozšírenie by sa dalo použiť pri implementácii pokladne. Na trhu sa ponúkajú váhy so vstavaným USB pripojením prípadne sériová linka. Pre náš mikropočítač by žiadna z týchto možností nerobila problém, keďže podporuje väčšinu známych protokolov.

7.5 Nabíjacie stanice

Stanica vybavená platobným terminálom prípadne mincovníkom. Umožňoval by klientovi vložiť peniaze na jeho konto.

7.6 WEB a mobilná aplikácia

Aktuálnym trendom v oblasti informatiky je mať všetko online. Existuje niekoľko pamäťových úložísk, ktoré zálohujú lokálne súbory na vzdialené servery. Alebo cloudové riešenia, kde sa aplikácie už neinštalujú lokálne do počítača, ale využíva sa webový prehliadač. Internet sa stáva stále viacej dostupnejší a nie je problém sa pripojiť do siete internetu takmer kdekoľvek. Či už s využitím mobilného internetu alebo verejných Wi-Fi sieti. To otvára ďalšiu veľkú kapitolu možností vývoja. Pomocou webu alebo aj mobilnej aplikácie by sa dalo vyriešiť objednávanie. Kde sa užívateľ prihlási a môže vytvárať objednávky či prezerať jedálny lístok.

7.7 Rozšírenia terminálu

Najdôležitejšou dierou v ponuke je väčšie množstvo zariadení. Ktoré by sa medzi sebou líšili hlavne veľkosťou. Mať len jedno zariadenie s veľkosťou displeja podobou mobilnému telefónu určite nestačí. No okrem toho aj aktuálne zariadenie sa dá ešte vylepšiť

7.7.1 Využitie technológie PoE

Vďaka nenáročnosti zariadenia na dodávanú elektriku, mám možnosť využiť napájanie zdieľané s internetovým pripojením. Za normálnej prevádzky je odber mikropočítača spolu s displejom okolo 0,8A s nárazmi do maximu 1 ampér. To znamená maximálny odber 5 wattov. Podľa štandardu IEEE 802.3af [5]. Je možné pomocou PoE získať až 60W. To by pokrylo dokonca aj obojstranný displej. Táto možnosť bola spomenutá aj v analýze2, kde sa výkon dvoch zariadení vypočítal na maximálne 25W. Avšak zatiaľ ešte nie je súčasťou projektu.

7.7.2 Raspberry Pi - compute module

Ako je v analýze 2 opísané, Raspberry Pi je vyrobený na študijné účely a nie je smerovaný na celodennú prevádzku. I keď to cieľom nebolo, už niekoľko mesiacov fungujú moje terminály v bezproblémovom stave. Dokonca mikropočítač našiel aj iné využitia, kde jeho celodenné fungovanie je nevyhnutné. To nasmerovalo firmu Raspberry k vytvoreniu rovnakého modulu, avšak do nepretržitých prevádzok. Je ním Raspberry Pi compute modul. Jeho výhodou je malý dizajn. Avšak nevýhodou, že k jeho fungovaniu je potrebné vyrobiť dosku. Samotné zariadenie neobsahuje žiadne konektory a je potrebné ho zapojiť do rozhrania rovnakého ako je použité pre pamäte RAM. Výhody, ktoré by priniesol v mojom termináli je nižšia cena, stavaná pamäť, menšie rozmery a integrované USB, napájanie aj internet do jedného konektoru. Dokonca podľa výrobcu disponuje dvomi výstupnými konektormi na displej. To by umožnilo použitie jedného zariadenie v prípade obojstranného terminálu. Avšak z doterajších informácii je konektor displeja s označením 0 vybavený len 2 kanálovou komunikáciou. Ktorá pravdepodobne nepostačuje na použitie 7 palcového displeja.

Záver

Hlavným cieľom diplomovej práce bolo vytvoriť funkčný stravovací systém. Výsledkom je databázový server, C# natívna aplikácia na správu systému, klientov a terminálov, C# systémová služba reprezentujúca aplikačný server s obslužnými programami pre terminály. A zobrazovací terminál využívaný klientami na interakciu so systémom. K základnému fungovaniu stravovacieho systému je potrebný serverový počítač a minimálne jeden jednostranný a jeden obojstranný terminál. Na počítači môže bežať databázový, aplikačný server a aplikácia na správu systému. Jednostranný terminál by sa použil na objednávanie a druhý na výdaj. Toto riešenie sa môže rozšíriť a použiť viacej objednávacích či výdajných terminálov súčasne. Prípadne rozšíriť funkcionalitu systému o informačný terminál.

Obmedzenia na výsledný produkt neboli, tak som mohol použiť všetky dostupné možnosti na naplnenie cieľov. V prípade databázového servera som použil voľne dostupnú databázu MySQL, čím znižujem náklady na prevádzku systému. U správy systému som použil WPF framework. Výsledná aplikácia ponúka jednoduchú prácu s intuitívnym ovládaním na vytváranie objednávok a jedálneho lístka. Táto aplikácia spolu s aplikačným serverom sú naprogramované v jazyku C# a zabezpečujú bežnú funkcionalitu.

Hlavná časť analýzy a vývoja bola zameraná na terminály. Tam som implementoval softvér a vyrábal kryt terminálu, aby sa mohol použiť v prevádzke. Informačný systém obsahuje viaceré terminály, preto dôležitosť lacnej výroby na jeden kus bola dôležitá. Jeho obsahom je mikropočítač Raspberry Pi, 7 palcový dotykový displej, bzučiak na zvukovú signalizáciu a čítačka kariet. Mikropočítač obsahuje operačný systém Raspbian a beží na nom moja aplikácia implementovaná v programovacom jazyku C++ s využitím QT knižníc. Na toto zariadenie som vytlačiť plastnovú krabičku na 3D tlačiarni. Po spojení som získal konečný výsledok terminálu. Jeho cena sa bohybuje okolo 200 eur v prípade jednostrannej a okolo 300 u obojstrannej krabičky. Avšak vo väčšine prevádzok je plastové krabička nedostačujúca. Pri použití oceľovej varianty sa náklady na terminál zvýšia. Tým sa dostávame na konečnú hodnotu 300

respektíve 400 eur u obojstranného displeja. Ceny sú orientačné, keďže cena zariadení nie je stabilná. Pri porovnaní s konkurenčnými zariadeniami, ako bolo spomenuté v kapitole 1.1.2, je cena jednostranného terminálu od firmy AV factory 89 298,00 Kč.

Vďaka spolupráci s firmou sa podarilo nasadenie môjho terminálu v prevádzke. Ten využíva ich existujúci aplikačný server. Použitie mojej implementácie serverov nebolo možné, keďže ich systémy sú prepojené na viacej služieb. Vďaka výberu QT frameworku, sa mi podarilo splniť celý plán. Podporuje komunikáciu s USB aj UART čítacou hlavou, zvukovú signalizáciu aj zobrazovanie potrebných dát.

Systém plne zabezpečuje objednávanie aj výdaj, čím napomáha k fungovaniu jedálne. V prípade použitia je možná náhrada stravovacích lístkov týmto systémom a mojím ďalším cieľom je osloviť stravovacie zariadenia stredných škôl a nájsť uplatnenie vytvoreného stravovacieho systému.

Literatúra

[1] Národná rada Slovenskej republiky: ZÁKONNÍK PRÁCE. 2001. Dostupné z: https://www.slov-lex.sk/pravne-predpisy/SK/ZZ/2001/311/

[2] Mbed: *mbed NXP LPC1768 prototyping board*. Dostupné z: https://www.nxp.com/docs/en/brochure/LPC1768.pdf

[3] Mbed: *Mbed OS 5*. Dostupné z: https://os.mbed.com/docs/v5.8/introduction/index.html

[4] Campbell, J.: *C Programmer's Guide to Serial Communications*. Sams, 1993, ISBN 0672302861. Dostupné z: https://www.amazon.com/C-Programmers-Guide-Serial-Communications/dp/0672302861?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0672302861

[5] Eisen, M.: *Introduction to PoE and the IEEE802.3af and 802.3at Standards*. Marcum Technology, 2009. Dostupné z: https://ieee.li/pdf/viewgraphs/introduction_to_poe_802.3af_802.3at.pdf

[6] Microsoft: *What's new in the .NET Framework*. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/

[7] Raspberry Pi Foundation: *Raspberry Pi 3 Model B+*. Dostupné z: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf

[8] Barrett, D. J.; Silverman, R. E.; Byrnes, R. G.: *SSH, The Secure Shell: The Definitive Guide*. O'Reilly Media, 2005, ISBN 0596008953. Dostupné z: https://www.amazon.com/SSH-Secure-Shell-Definitive-Guide/dp/0596008953?SubscriptionId=

0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0596008953

[9] Hopcroft, J. E.; Motwani, R.; Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison Wesley, 2006, ISBN 0321462254. Dostupné z: https://www.amazon.com/Introduction-Automata-Theory-Languages-Computation/dp/0321462254?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0321462254

[10] Clifton, I. G.: *Android User Interface Design: Implementing Material Design for Developers (2nd Edition) (Usability)*. Addison-Wesley Professional, 2015, ISBN 0134191404. Dostupné z: https://www.amazon.com/Android-User-Interface-Design-Implementing/dp/0134191404?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0134191404

Dodatok **A**

Zoznam použitých skratiek

RPi Raspberry Pi

GUI Graphical user interface

XML Extensible markup language

Telnet Teletype network

SSL Secure Sockets Layer

FTP File Transfer Protocol

SFTP SSH File Transfer Protocol

TLS Transport Layer Security

EER Enhanced Entity–Relationship

UTF Unicode Transformation Format

RAM Random Access Memory

DHCP Dynamic Host Configuration Protocol

IP Internet Protocol

WPF Windows Presentation Foundation

CSS Cascading Style Sheets

LAN local-area network

HDMI high-definition video devices

DSI Display Serial Interface

3D 3-dimensional

Dodatok **B**

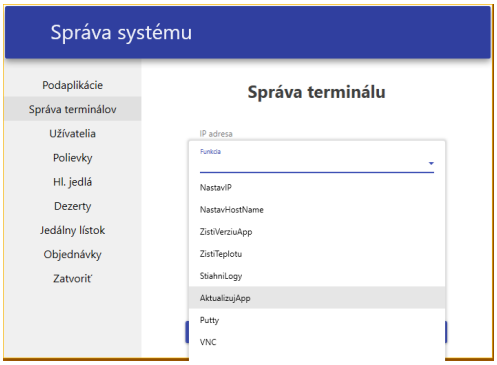
Obsah priloženého CD

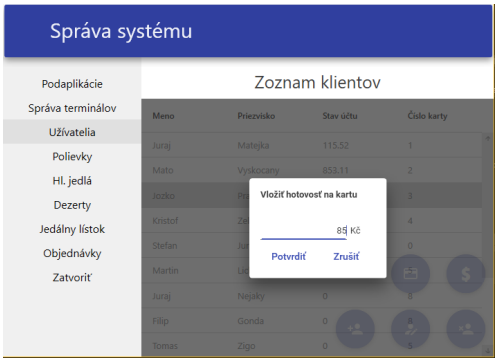
- `exe` ..... adresár so spustiteľnou formou implementácie
  - `AplikacnyServer` ..... Služba aplikačného servera
  - `SpravaSystemu` ..... Aplikácia na správu systému
  - `TerminalLinux` ..... Aplikácia terminálu pre Linux
  - `TerminalWindows` ..... Emulátor terminálu pre Windows
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `3Dmodely` ..... projekt 3D modelu na tlacenie
    - `QtApplication` ..... aplikácie pre terminál
    - `ServerAndAdmin` ..... aplikačný server a správa systému
    - `Database.sql` ..... skrypt na vytvorenie databázy
  - `DP_Kello_Tomas_2018.zip` ..... práca vo formátu LaTeX
- `text`
  - `DP_Kello_Tomas_2018.pdf` ..... text práce vo formáte PDF
- `readme.txt` ..... stručný popis obsahu CD

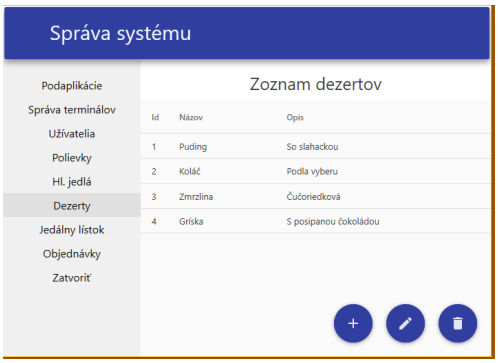
Dodatok C

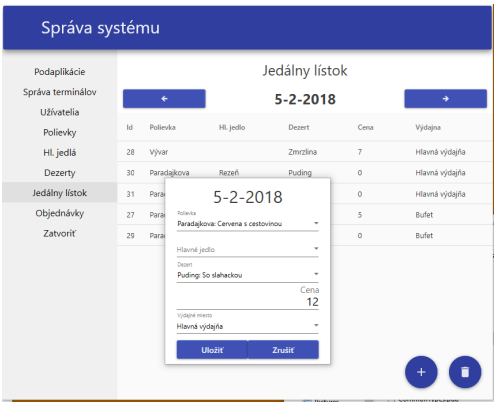
Prílohy

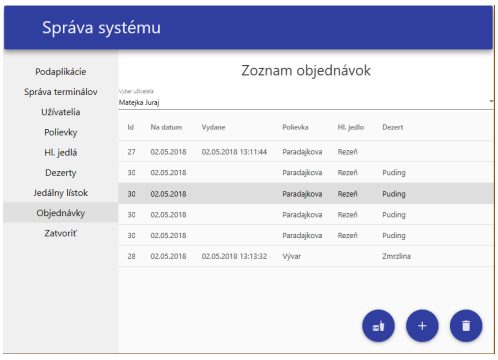
Aplikácia na správu systému



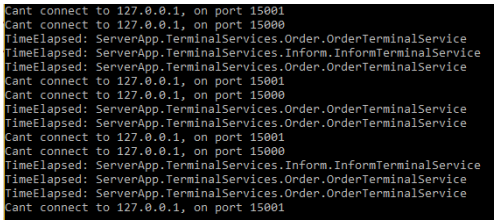

Správa systému
Podaplikácie
Správa terminálov
Užívatelia
Polievky
Hl. jedlá
Dezerty
Jedálny lístok
Objednávky
Zatvoriť
Zoznam klientov
Meno
Priezvisko
Stav účtu
Číslo karty
Vložiť hotovosť na kartu
85 Kč
Potvrdiť
Zrušiť


Správa systému
Podaplikácie
Správa terminálov
Užívatelia
Polievky
Hl. jedlá
Dezerty
Jedálny lístok
Objednávky
Zatvoriť
Zoznam dezertov
Id
Názov
Opis
1 Puding So slahackou
2 Koláč Podla vyberu
3 Zmrzlina Čučoriedková
4 Griska S posipanou čokoládou


Správa systému
Podaplikácie
Správa terminálov
Užívatelia
Polievky
Hl. jedlá
Dezerty
Jedálny lístok
Objednávky
Zatvoriť
Jedálny lístok
5-2-2018
5-2-2018
Paradajkova: Cervena s cestovinou
Hlavné jedlo
Puding: So slahackou
Cena
12
Hlavná výdajňa
Uložiť
Zrušiť

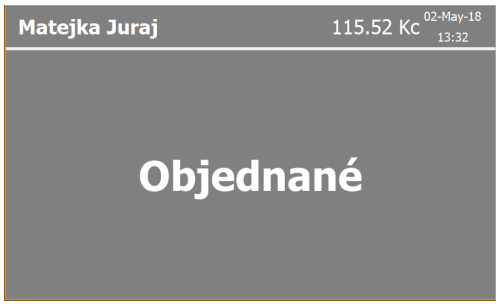

Správa systému
Podaplikácie
Správa terminálov
Užívatelia
Polievky
Hl. jedlá
Dezerty
Jedálny lístok
Objednávky
Zatvoriť
Zoznam objednávok
Matejka Juraj

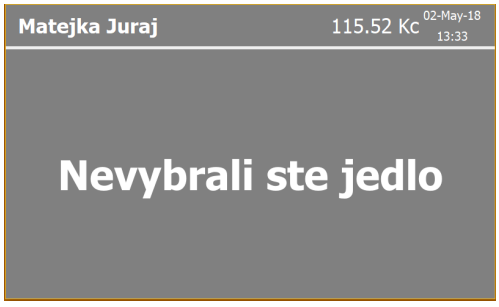
Výpis z konzoly aplikačného servera



Informačný terminál

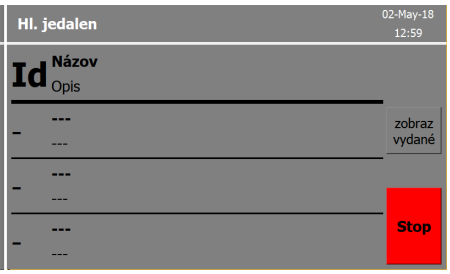
Objednávací terminál

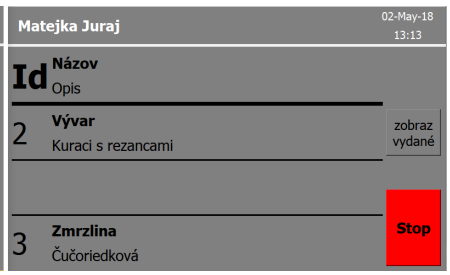


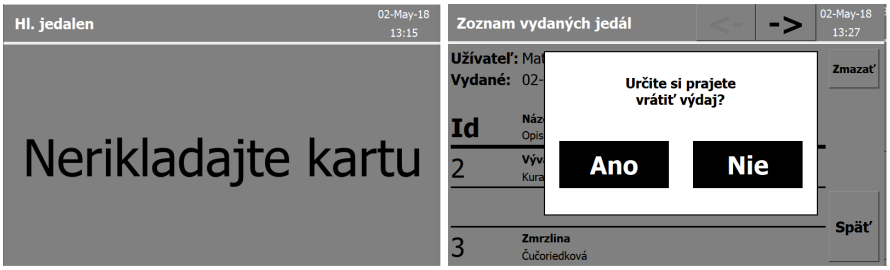


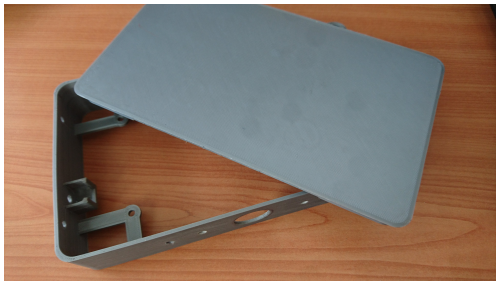
Výdajný terminál

Ľavá strana je zo strany klienta, pravá zo strany obsluhy









Jednostranná krabička z plastu

Obojstranná krabička z plastu