FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Návrh a implementace řídící aplikace technologie Pick to light ve skladu
Student:	Bc. Michael Olšavský
Vedoucí:	Ing. Martin Beránek
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je návrh efektivního procesu vyzvedávání objednávek ze skladových lokací používajících technologii Pick to light (dále PTL) a jeho následná implementace. Očekávaným výstupem je samostatná aplikace komunikující se skladovým systémem a PTL hardwarem.

1) Analyzujte a navrhněte efektivní a pro uživatele intuitivní proces vyzvedávání objednávek pomocí systému PTL. Zohledněte existující skladové procesy a předem známá hardwarová omezení.
2) Implementujte TCP/IP komunikaci s kontrolními prvky PTL.
3) Navrhněte a implementujte HTTP komunikaci se skladovým systémem, která bude odpovídat existujícím standardům systému.
4) Navrhněte a implementujte výslednou aplikaci komunikující s oběma stranami (kontroléry PTL a API skladového systému), která respektuje navržený proces vyzvednutí objednávky.
5) Aplikaci automaticky otestujte a navrhněte manuální testovací scénáře s důrazem na souběžné užívání systému více uživateli.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 9. února 2023 v Praze.

Diplomová práce

NÁVRH A IMPLEMENTACE ŘÍDÍCÍ APLIKACE TECHNOLOGIE PICK TO LIGHT VE SKLADU

Bc. Michael Olšavský

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Bc. Martin Beránek
4. května 2023

Obsah

Poděkování vii

Prohlášení viii

Abstrakt ix

Úvod 1
0.1 Motivace a cíl práce . . . 1
0.2 Struktura práce . . . 2

1 Teoretický kontext 3
1.1 Popis problémové domény . . . 3
1.1.1 E-commerce fulfillment . . . 3
1.1.2 Skladovací procesy . . . 4
1.1.3 Vychystávání zboží . . . 6
1.1.4 Pick to Light . . . 7
1.2 Analýza současných skladovacích procesů . . . 8
1.2.1 Testovací sklady . . . 8
1.2.2 Plánování práce . . . 8
1.2.3 Proces doplňování zboží . . . 8
1.2.4 Proces vychystávání zboží . . . 9
1.2.5 Proces inventarizace . . . 11
1.2.6 Shrnutí v kontextu *PTL* . . . 11

2 Metodika vývoje 13

3 Požadavky na systém 15
3.1 Funkční požadavky . . . 15
3.2 Nefunkční požadavky . . . 15

4 Analýza technologie Pick to Light 17
4.1 Hardware . . . 17
4.1.1 TW2209 – Kontrolér . . . 19
4.1.2 TW2210 – Junction box . . . 19
4.1.3 Moduly . . . 20
4.2 Komunikace s externím systémem . . . 21
4.3 Rozložení a zapojení . . . 22
4.4 Existující implementace komunikační vrstvy . . . 23

5 Návrh procesů 25
5.1 Proces vychystávání objednávek . . . 25
5.2 Proces doplňování zboží . . . 28
5.3 Proces údržby . . . 28

6 Implementace **31**
6.1 Výběr technologií . . . 31
6.2 Návrh architektury řídící aplikace . . . 33
6.3 Modul komunikace s *PTL* . . . 34
6.3.1 PickToLightAPI . . . 34
6.3.2 PickToLightConnection . . . 35
6.3.3 Odchozí *PTL* příkazy . . . 36
6.3.4 Příchozí *PTL* příkazy . . . 37
6.4 Business logika řídící aplikace . . . 37
6.4.1 Warehouse Event . . . 40
6.4.2 PTL fasáda . . . 41
6.5 Komunikace s WMS . . . 42
6.5.1 Configuration API . . . 43
6.5.2 Picking API . . . 44
6.5.3 Maintenance API . . . 48
6.5.4 Klienti . . . 49

7 Testování **51**
7.1 Automatizované testy . . . 51
7.1.1 Komunikace s *PTL* . . . 51
7.1.2 Komunikace s WMS . . . 51
7.1.3 Business logika . . . 52
7.2 Manuální testy . . . 52
7.3 Uživatelské testy . . . 52

8 Sestavení a nasazení **55**

9 Závěr **57**

A Seznam zkratek **59**

B Příloha – Testovací scénáře **61**

Obsah přiloženého média **71**

Seznam obrázků

1.1 Základní skladovací procesy dle studie [7] 4
1.2 Pick to Light 7

2.1 Agilní model vývoje softwaru podle [16] 13

4.1 Vizualizace ukázkového propojení komponent *PTL* systému 18
4.2 Lightning Pick TW2209 controller – hardware interface 19
4.3 Lightning Pick TW2210 junction box – hardware interface 19
4.4 Lightning Pick regál s lištami pro připojení modulů 20
4.5 Lightning Pick MW series – single module 21
4.6 Komponenty Proglove skeneru 21
4.7 Rozložení a zapojení *PTL* v jednom ze skladů zadavatele 22
4.8 Ukázka *PTL* instalace v jednom ze skladů zadavatele 23

5.1 Případy užití *PTL* systému 25
5.2 Flow diagram procesu vychystávání z pohledu jednoho pracovníka *PTL* systému. 27
5.3 Flow diagram procesu údržby. 29

6.1 Ilustrace smyčky událostí (Event loop) v prostředí NodeJS 32
6.2 Moduly řídící aplikace 33
6.3 Diagram tříd pro komunikaci s *PTL* 34
6.4 Diagram tříd *PTL* zařízení 35
6.5 Formát příkazu pro komunikaci s *PTL* 36
6.6 Popis jednotlivých polí formátu příkazu pro komunikaci s *PTL* 36
6.7 Stavy a přechody uživatele *PTL* systému 38
6.8 Diagram aktivit pro část procesu vychystávání objednávky 39
6.9 Diagram tříd business logiky řídící aplikace 42

7.1 Zaznamenaná latence z reálného provozu operace *pickProduct* 53

Seznam tabulek

1.1 Způsoby transportace zboží uvnitř skladu. Zdroj: [11] 9
1.2 Srovnání standardních technologií pro vychystávání objednávek. Zdroj: [11] . . . 10

Seznam výpisů kódu

1 Configuration API – operace getConfiguration 44
2 Picking API – operace login 45
3 Picking API – operace retrievePicks 46
4 Picking API – operace pickProduct 47
5 Picking API – operace skipPick 48
6 Picking API – operace fullTote 49
7 Maintenance API – operace configureLightModule 50
8 Maintenance API – operace verifyPermission 50

Děkuji Ing. Bc. Martinovi Beránkovi za podnětné konzultace a cenné rady při tvorbě této práce. Dále bych rád poděkoval své přítelkyni a rodině za podporu nejen při psaní práce, ale v průběhu celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 4. května 2023 .....

Abstrakt

Tato diplomová práce se zabývá návrhem procesu vychystávání objednávek s využitím technologie Pick to Light (*PTL*) ve skladech a následnou implementací řídící aplikace *PTL* systému na základě navrženého procesu. V první části práce je popsána obecná problematika skladování se zaměřením na *e-commerce fulfillment*, která je následně konkretizována analýzou současných procesů ve skladech zpřístupněných k pozorování. Dále je provedena analýza technologie Pick to Light a proveden sběr požadavků na výslednou aplikaci.

V další části práce je pak na základě získaných poznatků navržen proces vychystávání (a údržby systému Pick to Light), přičemž je kladen důraz na jeho efektivitu, intuitivnost, ale i kompatibilitu s jinými již zavedenými skladovými procesy. Dále se práce zabývá implementací navrženého procesu v podobě řídící aplikace *PTL* systému napsané v jazyce TypeScript. Aplikace byla iterativně testována, vylepšována, a nakonec i nasazena v produkčním provozu.

Klíčová slova Pick to Light, PTL, sklad, skladové systémy, WMS, logistika, fulfillment

Abstract

This thesis deals with the design of a picking process using Pick to Light (*PTL*) technology in warehouses and the subsequent implementation of a *PTL* system control application based on the designed process. The first part of the thesis describes the general warehousing issues with a focus on *e-commerce fulfillment*, which is then concretised by analysing the current processes in the warehouses made available for observation. Furthermore, an analysis of the Pick to Light technology is performed and the requirements for the resulting application are collected.

In the next part of the thesis, the picking process (and maintenance of the Pick to Light system) is then proposed based on the findings, with emphasis on its efficiency, intuitiveness, and compatibility with other already established warehouse processes. Furthermore, the thesis deals with the implementation of the proposed process in the form of a *PTL* system control application written in TypeScript. The application was iteratively tested, improved and finally deployed in production.

Keywords Pick to Light, PTL, warehouse, warehouse management system, WMS, logistics, fulfillment

Úvod

0.1 Motivace a cíl práce

Trh e-commerce v poslední dekádě raketově roste a hranice pro vstup do něj se díky novým technologiím a online platformám stále snižuje. Mnoho malých a středních podniků ale nemá kapacity řešit vlastní logistiku a využívá proto tzv. *3PL* (third party logistics), tedy služeb externích poskytovatelů. Očekávání zákazníků na rychlé a bezchybné dodání zboží jsou však stále vyšší a společnosti poskytující *3PL* služby tak musí neustále inovovat, aby byly schopné poptávku uspokojit při rozumných nákladech.

Mnoho společností nabízí dodání a zprovoznění kompletních logistických řešení od hardwaru po Warehouse Management System (*WMS*, 1.1.2.0.5). Tato řešení jsou však často zaměřena na tradiční odvětví logistiky a neodpovídají požadavkům skladů zaměřených na *e-commerce fulfillment* (1.1.1).

Mezi nejnákladnější operace ve skladové logistice patří proces výběru naskladněného zboží pro konkrétní objednávky (tzv. vychystávání) [1]. Jednou z technologií zaměřených na zrychlení vychystávání při současném snížení chybovosti je technologie *Pick to Light* (*PTL*, 1.1.4).

Tato práce se zabývá návrhem procesu vychystávání objednávek s využitím technologie *PTL* a následnou implementací řídící aplikace *PTL* systému na základě navrženého procesu komunikující s *WMS* skladů zpřístupněných za účelem testování.

Hlavní cíle této práce jsou následující:

- navrhnout intuitivní a pro obsluhu jednoduchý proces vychystávání s technologií *PTL*, který udržuje kompatibilitu s již zavedenými skladovacími procesy,
- analyzovat konkrétní hardwarovou realizaci *PTL* systému a implementovat komunikační vrstvu pro řídící aplikaci,
- implementovat řídící aplikaci *PTL* systému respektující navržený proces s ohledem na potřeby skladového prostředí,
- a řešení otestovat v reálném provozu.

0.2 Struktura práce

Rešeršní část práce má za cíl popsat problematiku skladových procesů týkajících se zpracování objednávek se zaměřením na sklady typu *e-commerce fulfillment*, a analyzovat technologii *PTL* a její typické implementace. Očekávaným výstupem této části je návrh procesu vychystávání zboží ze skladových lokací s využitím technologie *PTL*, který bude kompatibilní s již zavedenými skladovacími procesy, a seznam požadavků na systém.

Cílem praktické části je pak implementace řídící aplikace *PTL* systému (včetně případných podpůrných systémů) na základě navrženého procesu, která bude splňovat všechny zmapované požadavky, a její následné testování a nasazení v reálném provozu.

Kapitola 1
Teoretický kontext

1.1 Popis problémové domény

V této kapitole jsou shrnuty základní pojmy z domény skladovací logistiky, které jsou významné pro pochopení problematiky a efektivní návrh *PTL* procesu. Pojmy a vysvětlení vychází z obecné literatury[1], ale odkazují se i na konkrétní implementace ve skladech poskytnutých k analýze a testování v rámci této práce.

1.1.1 E-commerce fulfillment

E-commerce fulfillment je souhrnný termín pro obstarání kompletní logistiky internetovým obchodům, online tržištím a dalším hráčům na poli e-commerce. Pod *e-commerce fulfillment* spadá (nejen):

- synchronizace objednávek z různých zdrojů (např. e-shopy),
- příjem a skladování zboží,
- inventura (udržování stavu zásob),
- vychystání (zkompletování) objednávek,
- expedice zboží (balení, tisk štítků, předání dopravci),
- vrácení zboží,
- doručení informací kupujícímu (stav přepravy).

Realizace vlastního e-commerce řešení je velmi nákladná záležitost a vyžaduje značné investice do technologií, skladových prostor a zaměstnanců. V drtivé většině případů je výhodné využít hotových řešení, které pomohou standardizovat a nastavit procesy, ať už ve formě softwarových balíků (např. *Oracle NetSuite Order Fulfillment* [2]) nebo SaaS (Software as a Service) platforem (např. *Deposco*, *ShipStation* [3]).

Často výhodnější volbou, obzvláště pro rychle rostoucí společnosti, jejichž hlavní doménou není logistika, je outsourcing denní operativy e-commerce fulfillmentu na specializované společnosti, tzv. *3PL* (third-party logistics), které mají prostor do technologií a infrastruktury investovat ve velkém a nabídnout tak lepší služby, ve výsledku často za nižší cenu.[2]

[1] Některé pojmy nemají v českém jazyce ustálený překlad, proto jsou uvedeny v angličtině.

[2] Pod pojem 3PL patří i tradiční logistické společnosti (DHL, Fedex), které se zaměřují na řešení na míru pro velké společnosti a mají tedy významně jinou klientelu i podmínky.

Distribuční centra zajišťující 3PL pro *e-commerce fulfillment* mají svá specifika – typicky musí být optimalizována na zpracování **vysokého počtu malých objednávek** v **minimálním čase** (v řádu hodin, maximálně jednotek dní), jelikož právě doba a cena doručení zboží patří mezi nejzásadnější faktory spokojenosti zákazníka s online nákupy [4]. Zároveň musí podporovat širokou škálu produktů lišících se ve fyzických dispozicích, ale i v požadavcích na skladování.

V porovnání s tradiční skladovou logistikou musí být *e-commerce fulfillment* distribuční centra značně flexibilnější i v plánování práce a skladovacích kapacitách [5]. Zatímco u tradičních zásobovacích skladů pro jednoho konkrétního prodejce lze predikovat četnost prodeje konkrétních položek nejen na základě historických dat, ale i dalších vnitro-firemních dat (např. na základě marketingu a plánovaného rozpočtu), v případě *e-commerce fulfillment* skladů, obsluhujících stovky až tisíce různých zákazníků, je situace výrazně složitější. I tak se ale prováděné skladovací procesy nevymykají tradiční kategorizaci popsané v následující sekci 1.1.2.

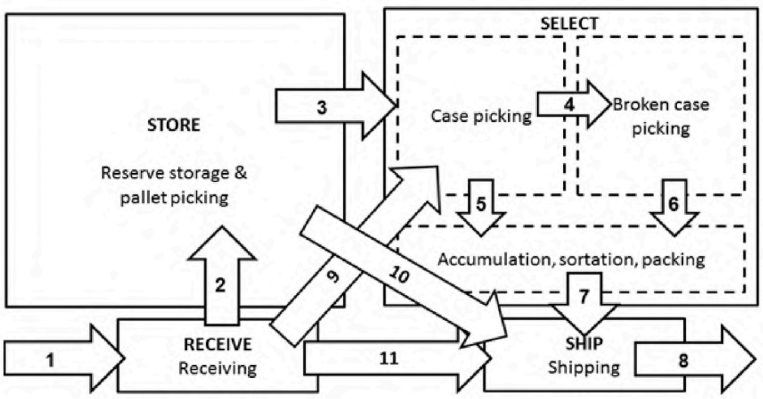
1.1.1.0.1 Zákazník a kupující Z pohledu *3PL* poskytovatele je zákazníkem jakýkoliv subjekt, který vědomě využívá služeb *e-commerce fulfillment*, nikoliv kupující (koncový zákazník objednávající produkt na prodejním kanále). Dále v textu je tedy pojem zákazník vždy zastupovat tuto definici, zatímco koncový zákazník bude označován jako *kupující*.

1.1.2 Skladovací procesy

Pro vysvětlení role technologie *PTL* je třeba znát obecné skladovací procesy a pro její efektivní využití je nutné vycházet i z konkrétní implementace procesů ve skladech.

Ačkoliv se rozložení skladů a implementace skladovacích procesů může zásadně lišit v závislosti na povaze skladovaného zboží a účelu skladového komplexu, obecně lze skladovací procesy rozdělit do následujících 4 základních kategorií:[3]

- příjem zboží (**naskladnění**),
- evidence zboží (**inventarizace**),
- zkompletování objednávky (**vychystávání zboží**),
- odeslání objednávky (**expedice**).



Obrázek 1.1 Základní skladovací procesy dle studie [7]

Graf 1.1 ilustruje návaznost jednotlivých procesů. Obvyklý životní cyklus zboží ve skladě na *e-commerce fulfillment* vypadá následovně.

[3] Jednotlivých procesy jsou popisuje například [6].

Zákazník (1.1.1.0.1) objedná zboží u dodavatele a notifikuje svého *3PL* poskytovatele. Poté, co dodavatel přiveze zboží do skladu (typicky po velkých dávkách na paletách), je provedena jeho identifikace (1.1.2.0.3) a kontrola kvality, a na základě odhadů poptávky jsou vybrány skladové lokace (1.1.2.0.1), kam a v jakých kvantitách má být zboží umístěno. V případě, že je evidována velkoobchodní objednávka, může být paleta namísto naskladnění rovnou přebalena a odeslána kupujícímu.

Naskladněné zboží je následně periodicky kontrolováno, evidováno a přemisťováno do výhodnějších lokacích na základě dalších analýz, bez vědomí zákazníka. Skladový systém musí počítat s tím, že zboží může být poškozeno, ztraceno nebo ukradeno a musí být nastaveny procesy pro řešení těchto situací. Informace o stavu zboží jsou zpřístupněny zákazníkovi a zpětně synchronizovány s jeho prodejním kanálem. Jakmile je v prodejním kanále zákazníka zaznamenána objednávka, je požadovaná kvantita zboží vychystána z naskladněných lokací (1.1.3), objednávka je zabalena a expedována kupujícímu.

1.1.2.0.1 Skladová lokace je fyzické místo ve skladě, ve kterém je možné skladovat určitý typ zboží. Lokace je jednoznačně identifikovatelná, typicky pomocí alfanumerického identifikátoru reprezentovatelného čárovým kódem, aby bylo možné provádět systémovou evidenci zboží. Mezi základní typy skladových lokací patří paletové lokace, na kterých je umístěno zatím nerozbalené zboží na paletách, a lokace pro vychystávání zboží, ze kterých lze zboží vychystat (1.1.3) po jednotlivých kusech.

1.1.2.0.2 Zóny identifikují části skladu, které jsou vyhrazeny pro konkrétní skladové procesy. Mezi základní zóny patří zóna pro příjem zboží, zóna pro vychystávání zboží, zóna balení a zóna pro expedici. Dále se ve skladech často vyskytují speciální zóny pro křehké zboží, zóny s vysokou bezpečností (pro dražší položky), zóny s potravinami apod.

1.1.2.0.3 SKU (Stock keeping unit, česky skladová jednotka) je unikátní identifikátor konkrétní varianty produktu, obvykle ve formě čárového kódu. *SKU* se může lišit na základě typu, velikosti, barvy, balení atd. Právě na úrovni jednotlivých *SKU* může probíhat evidence a výběr zboží ze skladových lokací. Pro některé produkty je ale třeba produkt identifikovat kombinací *SKU* a číslem šarže (1.1.2.0.4).

1.1.2.0.4 Číslo šarže je identifikátor skupiny produktů se stejným *SKU*, které byly vyrobeny ve stejném časovém období a podle stejné receptury. Číslo šarže je využíváno pro sledování kvality, dodržování expirační lhůty a zpětné svolání vadných produktů.

1.1.2.0.5 Warehouse Management System (*WMS*) je software sloužící k řízení skladovacích procesů. Mezi nezbytné funkcionality *WMS* patří evidence zboží a skladových lokací. Kvalitní WMS by pak měl zahrnovat kompletní plánování a řízení skladovacích procesů popsaných v grafu 1.1. Podstatnou součástí *WMS* je také schopnost reportingu a komunikace s dalšími externí systémy, ať už s hardwarem v rámci skladu, tak s vyššími vrstvami, např. *ERP* (Enterprise resource planning) systémem [8]. Mezi známé zástupce *WMS* patří *Oracle Warehouse Management*, *SAP Extended Warehouse Management*, či *Extensive 3PL Warehouse Manager*.

1.1.3 Vychystávání zboží

Picking (standardní pojem, česky vychystávání zboží) je proces výběru a sběru zboží ze skladových lokací na základě požadavků konkrétní objednávky. Dle studie [1] se jedná se o nejdražší aktivitu co se operačních nákladů (cca. 55 % z celkových operačních nákladů) týče prováděnou ve skladech, a tedy i aktivitu s největším potenciálem minimalizace času a nákladů. Pro efektivní zpracování objednávek je zásadní rozložení skladových lokací a následné efektivní rozmístění zboží. Cílem procesu vychystávání zboží je minimalizace času a nákladů potřebných k vychystání objednávky při zachování minimální chybovosti – výsledný proces je tak často kompromisem.

1.1.3.0.1 Objednávka k vychystání v kontextu procesu vychystávání nemusí nutně reprezentovat objednávku kupujícího z prodejního kanálu. Může se jednat pouze o část objednávky, kdy zbývající zboží bude vychystáno později či z jiných sekcí nebo naopak o skupinu objednávek (1.2.4.0.1), kdy je po vychystání třeba provést jejich zpětné rozdělení (*sorting*).

1.1.3.0.2 Zóna vychystávání zboží je zóna skladu optimalizovaná na vychystávání produktů po jednotlivých kusech. Lokace v této sekci typicky obsahují již vybalené zboží, nikoliv palety. Zboží, u kterého se na základě předchozích analýz očekává zájem od zákazníků a bude ho tedy potřeba vyskladnit v řádu hodin, maximálně dní, je typicky doplňováno kontinuálně.

Některé objednávky, typicky velkoobchodní či objednávky seskupené v rámci *batching* procesu (1.2.4.0.2), mohou sekci vychystávání zboží přeskočit a přejít do balící zóny rovnou na paletách ze skladových zásob.

1.1.3.0.3 Skladové zásoby Zóna skladových zásob je povinnou součástí každého skladu. Obsahuje lokace, do kterých je zboží umisťováno při naskladnění, a ze kterých následně probíhá doplňování zboží do *sekce vychystávání*. Zboží v těchto lokacích je typicky uložené na paletách nebo v krabicích po větším množstvím kusů, přičemž lokace jsou rozloženy do regálů o několika vertikálních úrovních.

1.1.4 Pick to Light

PTL je technologie založená na principu světelných ukazatelů, která se používá ve skladech pro efektivní vychystávání objednávek (1.1.3). V porovnání s plně manuálním vychystáváním objednávek vykazuje zkrácení doby zpracování objednávky a výrazně menší chybovost [9].

PTL systém se skládá ze světelných modulů přiřazených ke konkrétním skladovým lokacím, typicky rozmístěných do mřížky. Pracovník skladu načte objednávku do systému, typicky naskenováním čárového kódu přepravky přiřazené k objednávce, načež *PTL* systém rozsvítí světelné moduly, které obsahují požadované položky a na displejích modulů zobrazí požadované kvantity. Pracovník světelné signály následuje, sbírá položky z jim přiřazených lokací do přepravky a potvrzuje výběr tlačítkem na modulech.

Obrázek 1.2 Pick to Light

1.2 Analýza současných skladovacích procesů

V této kapitole jsou popsány skladovací procesy implementované v testovacích skladech (1.2.1), které (přímo i nepřímo) ovlivňují návrh a použití technologie *PTL*. Termíny jsou nejprve vysvětleny samostatně, na konci kapitoly je pak shrnutí jejich vztahu k *PTL*.

1.2.1 Testovací sklady

Pro účely této práce byly zpřístupněny dva skladové komplexy zaměřené na *e-commerce fulfillment*, ve kterých byla dostupná, do té doby nevyužívaná, instalace *PTL* systému. Při obvyklém provozu k listopadu 2022 každý ze skladů odbavoval desítky tisíc objednávek denně. Zavedené procesy tak bylo možné pozorovat na velkém vzorku dat z reálného provozu získaných z interně vyvíjeného *WMS*.

V obou skladech byly k dispozici kontaktní osoby z operativy, které poskytovaly informace o procesech a odpovídaly na dotazy v průběhu analýzy.

1.2.2 Plánování práce

Zásadní kvalitou, a tedy i potenciální konkurenční výhodou *WMS* je efektivní plánování práce, obzvláště pak v dynamickém prostředí *e-commerce fulfillmentu*. V ideálním světě z pohledu operativy a minimalizace nákladů by bylo nejvýhodnější se striktně držet plánu, ze kterého by bylo možné exaktně odvodit potřebný počet pracovníků a časové odhady zpracování objednávek.

V reálném skladovém prostředí však do plánování vstupuje mnoho neznámých – zboží se ztrácí, objednávky se mění, přicházejí nové, výkonnost pracovníků není konzistentní, některé procesy se přehlcují atd. Prakticky je navíc i v případě fixních vstupů hledání globálně optimálního řešení nemožné. I jednotlivé podúlohy jako plánování cesty při vychystávání objednávky o více produktech nebo shlukování objednávek (1.2.4.0.2) patří mezi NP-těžké úlohy [10].

Na základě míry dynamiky skladového prostředí je možné plánování práce organizovat shora dolů, kdy je odhadnutý rámcový objem práce, kterou je třeba odbavit na základě SLA[4], kapacity procesů apod. a následně jsou konkrétní podčásti řešeny jako samostatné úlohy, ať už pomocí heuristik nebo s přihlédnutím na specifická umělá omezení [10], nebo zdola nahoru, kdy je primární snaha systému maximalizovat výkonnost jednotlivých podúloh v reálném čase. Kvalitní *WMS* typicky kombinuje oba přístupy.

1.2.2.0.1 *WMS* řídící testovací sklady vytváří rámcový plán práce den předem, při kterém jsou vybrány objednávky, které mají být odbaveny v následujícím dni. V návazném procesu je zboží z těchto objednávek předem alokováno na konkrétní lokace s cílem minimalizovat parametry jako celková vzdálenost či počet přemístění napříč zónami. Zboží dostupné pouze ve skladových zásobách je doplněno do zóny vychystávání (1.1.3.0.2). Alokace zboží, ale i odbavované objednávky jsou přehodnocovány v průběhu dne v reakci na změny stavu skladu.

1.2.3 Proces doplňování zboží

Doplňování zboží je proces řešící přemísťování skladových zásob tak, aby na lokacích bylo k dispozici dostatečné množství zboží konkrétního typu odpovídajícího parametrům (technologickým, rozměrovým) lokace. Kvalitní proces doplňování zboží je stěžejní pro následné efektivní vychystávání. Doplňování zahrnuje primárně přesun zboží z paletových zásob do lokací v sekci vychystávání, ale může docházet i k přesunům v rámci jedné zóny s cílem snížit celkovou vzdálenost při vychystávání – zpravidla platí, že skupinové operace (např. pohyb palety, boxu) jsou výrazně

[4]SLA (Service-level agreement) udává dohodnutou míru dostupnosti systému a garantované služby v daném časovém úseku.

efektivnější než operace jednotkové. Doplňování zboží by měla předcházet komplexní analýza poptávky a následná optimalizace (či rovnou simulace) návazných procesů.

1.2.3.0.1 V testovacích skladech probíhá doplňování zboží kontinuálně v průběhu dne a je tedy důležité, aby neblokovalo ostatní procesy. Strategie výběru lokací pro produkty je velmi komplexní a liší se napříč skladovými zónami. Zpravidla je kombinování různého zboží v jedné lokaci je povoleno pouze v lokacích obsahující zabalené produkty (palety, boxy), zatímco v sekci vychystávání (1.1.3.0.2) jsou lokace obsazeny nejvýše jedním typem produktu identifikovaným kombinací *SKU* a *číslem šarže*.

1.2.4 Proces vychystávání zboží

Jak už bylo uvedeno v kapitole 1.1.3, vychystávání zboží je jednou z nejnákladnějších operací a poskytuje proto velký prostor pro optimalizaci. Optimalizovat proces vychystávání lze v mnoha různých dimenzích.

V každém skladě je třeba vybrat (či nakombinovat) následující parametry [11]:

- způsob transportu zboží uvnitř skladu,
- výběr technologie pro vychystávání z lokací,
- směrování vychystávaných objednávek napříč zónami.

Mezi základní způsoby transportu zboží uvnitř skladu patří řešení využívající dopravníkový pás (*Pick to Conveyor*) a řešení pomocí manuálně přesouvaných vozíků (*Pick to Cart*), jejichž výhody a nevýhody shrnuje tabulka 1.1.

	Commonly Used Operations	Benefits	"Draw Backs"
Pick to Cart	▪ Large number of items with low movement per item ▪ Full case and piece picks operations with little system support to split out the orders	▪ No conveyor cost ▪ Highly flexible ▪ Multiple pickers per zone, if required	▪ Low pick rate due to typically long travel paths
Pick to Conveyor	▪ Low number of items ▪ High volume items ▪ Large number of very small items (i.e. jewelry)	▪ High pick rate due to small pick zones	▪ Typically only one picker per zone ▪ Conveyor cost

Tabulka 1.1 Způsoby transportace zboží uvnitř skladu. Zdroj: [11]

Tabulka 1.2 ze stejného článku pak shrnuje výhody a nevýhody technologií vychystávání z lokací. **Paper picking** reprezentuje způsob vychystávání, kdy je pracovníkovi předán seznam produktů potřebných pro zkompletování objednávky (*pick list*) spolu s lokacemi, na kterých

se tyto produkty nacházejí. Pracovník následně prochází sklad a postupně sbírá produkty do přepravky vyhrazené konkrétní objednávce. Tato technika je ale v porovnání s ostatními výrazně náchylná na chyby a pomalá.

Pokročilejší variantu reprezentuje v tabulce **Radio Frequency picking**, což je souhrnný název pro zařízení, které bezdrátově komunikují s *WMS*, navigují pracovníka skrze sklad a poskytují interaktivní vazbu – typicky jsou taková zařízení vybavena čtečkou čárových kódů a navštívené lokace i produkty je třeba skenovat pro kontrolu správnosti. Poslední srovnávanou variantou je pak právě **Pick to Light**.

	Commonly Used Operations	Benefits	"Draw Backs"
Paper Picking	▪ Small operations with little systemic support	▪ Low technology cost ▪ Low risk	▪ Low pick rate ▪ Low pick accuracy ▪ Long order cycle time
Radio Frequency Picking	▪ All types of operations	▪ High accuracy ▪ Paperless ▪ Ability to capture item specific information (i.e. serial numbers)	▪ Low pick rate (often the same as paper picking)
Pick to Light	▪ Low # of items with high volumes	▪ High accuracy ▪ High pick rate ▪ "Hands Free"	▪ High hardware costs

Tabulka 1.2 Srovnání standardních technologií pro vychystávání objednávek. Zdroj: [11]

Dále lze vychystávání optimalizovat například plánováním cesty pracovníka, vychystáváním zboží přímo do odesílaných krabic namísto přepravek (tzv. *Pick to Carton*), či velmi efektivním shlukováním objednávek [1].

1.2.4.0.1 Shlukování objednávek (angl. *cluster picking*) je technika, při které pracovník v jednom průchodu skladem vychystává více objednávek najednou. Objednávky jsou oddělené a mohou obsahovat rozdílné zboží, při efektivním výběru shluku se však významně ušetří čas strávený průchodem skladu.

1.2.4.0.2 Dávkové zpracování objednávek (angl. *batch picking*) je speciální varianta shlukování, při které jsou objednávky rozděleny do skupin na základě společných vlastností, například jde o jednokusové objednávky nebo objednávky obsahující identické kombinace zboží, a všechny procesy tak lze provést najednou, čímž se zásadně šetří čas.

1.2.4.0.3 Roztřídění (angl. *sorting*) V závislosti na implementaci procesu shlukování objednávek může být nutné společně vychystané objednávky znovu rozdělit. Tento proces se nazývá *sorting* a může být prováděn manuálně při balení nebo pomocí automatizovaných zařízení, např. *Sure Sort* [12] či inverzní varianty *PTL*, tzv. *Put to Light*.

1.2.4.0.4 Mezi modernější způsoby optimalizace vychystávání stavějících na uvedených technologiích pak patří roboticky asistované techniky (např. Locus [13]) či plně automatizované

skladové systémy, které jsou ale zpravidla velmi nákladné a fungují jen na omezené množině zboží.

1.2.4.0.5 Zásadní roli hraje také strategie umisťování zboží do lokací. Mezi základní dělení patří strategie dedikovaných regálů a strategie smíšených regálů [14]. V případě dedikovaných lokací jsou lokace předem rezervované pro konkrétní *SKU*, a lokace se stejnými *SKU* jsou umístěny blízko u sebe, zatímco strategie smíšených lokací rozmísťuje zboží dynamicky do libovolných lokací napříč celou zónou. Varianta smíšených lokací nabývá na popularitě zejména v posledních letech díky většímu potenciálu pro optimalizaci práce – problematikou efektivity a komplexitou plánování u obou strategií se zabývají například studie [14] a [15].

1.2.4.0.6 V testovacích skladech jsou implementovány prakticky všechny varianty výše zmíněných technik. V obou skladech jsou skladové zóny propojeny dopravníkem, který se stará o inteligentní směrování přepravek napříč zónami. Větší zásoby zboží jsou ale přepravovány manuálně.

Objednávky se stejným obsahem se *WMS* pokouší shlukovat a odbavovat najednou v rámci *batch procesu*, kdy je zboží navíc staženo přímo ze skladových zásob a přeskakuje se tak klasické vychystávání. Jednokusové objednávky (různého zboží) jsou také odbavovány skupinově a následně roztříděny až při balení.

Ve vychystávacích zónách se používá kombinace **Radio frequency picking** v podobě bezdrátového skenovacího zařízení komunikujícího s *WMS* a roboticky-asistovaného vychystávání technologií *Locus*. V obou případech se může odbavovat více objednávek najednou v rámci jednoho průchodu. Velmi zřídka však může být prováděn i čistě manuální **paper picking**.

Plánování cesty je prováděno pouze triviálně, větší důraz je na optimalizaci umístění zboží v průběhu vychystávání. Využívána je strategie smíšených regálů, zboží tedy může být náhodně rozmístěno v rámci celé zóny.

1.2.5 Proces inventarizace

Posledním z procesů, které potenciálně přímo ovlivňují práci s technologií *PTL*, je proces inventarizace, konkrétně jeho podčást *kontrola lokací*. *Kontrola lokací* je prováděna za účelem zjištění stavu zboží na konkrétní lokaci a je obvykle vyžádána *WMS* systémem, a to v pravidelných intervalech nebo na základě detekce chyby v průběhu jiného procesu. Pověřený pracovník skladu má možnost přepsat evidovanou kvantitu zboží na lokaci ve *WMS*, případně zboží přemístit jinam.

1.2.5.0.1 V testovacích skladech je kontrola lokací součástí každodenní operativy. Přednostně jsou kontrolovány lokace, u kterých byl detekován problém při provádění jiných procesů, jelikož ty jsou až do vyřešení problému ostatními procesy nepoužitelné. Při kontrole není možné s lokací manipulovat, jelikož je nutné zjistit přesný stav zboží, nicméně frekvence kontrol by z podstaty řešeného problému měla být prakticky zanedbatelná v porovnání s ostatními procesy.

1.2.6 Shrnutí v kontextu *PTL*

PTL je jednou z mnoha technologií využívaných pro urychlení a minimalizaci chybovosti procesu vychystávání. Primárními metodami optimalizace vychystávání jsou techniky dávkového zpracování (1.2.4.0.2), které již jsou využívány v obou testovacích skladech.

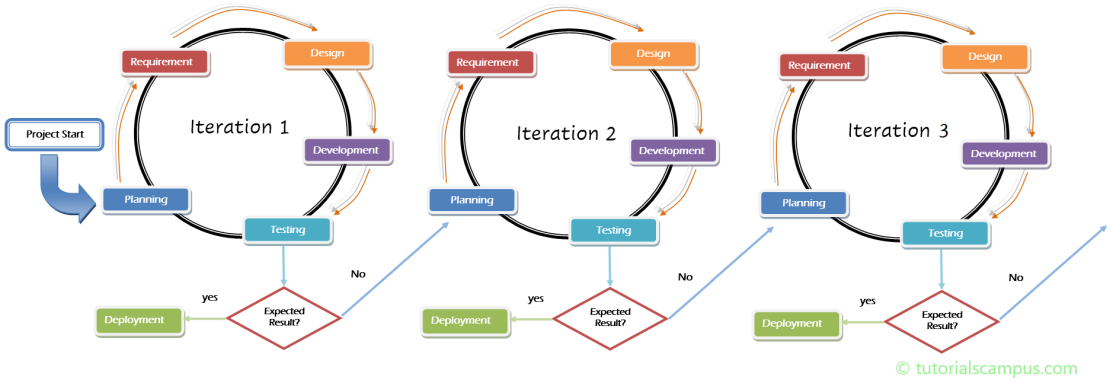
Navržené *PTL* řešení musí umět pracovat s libovolným typem objednávek nehledě na aplikované optimalizace. Objednávky mohou před odbavením technologií *PTL* navštívit libovolný počet skladových zón a jejich požadovaný a očekávaný obsah se může v průběhu času měnit, systém tedy musí pracovat se vždy aktuálními daty.

V rámci snahy nezavádět nekonzistence do původního chodu skladu je třeba řešení navrhnout tak, aby se z lokací dalo zboží vychystávat i tradičními metodami *paper picking* a *RF picking*. Dále musí navržené řešení zvládat odbavit objednávky nehledě na způsob jejich transportu po skladu a balení. V neposlední řadě je pak třeba zajistit možnost standardizovaným způsobem reportovat chyby týkající se zboží a lokací v průběhu vychystávání, a řešení navrhnout takovým způsobem, aby bylo možné *PTL* lokace kontinuálně doplňovat bez nutnosti pozastavení vychystávání.

Kapitola 2

Metodika vývoje

Vývoj řešení včetně návrhu procesů probíhal podle agilního modelu [16] ve 14denních sprintech. Každá iterace byla konzultována s kontaktní osobou produktového týmu pracující v testovacích skladech. Pozdější iterace pak byly testovány pracovníky skladu pro získání zpětné vazby (7).



Obrázek 2.1 Agilní model vývoje softwaru podle [16]

Samotný vývoj se držel praktik *Continuous Integration* [17] – na každodenní bázi byly zapracovávány menší části systému do hlavní vývojové větve nástroje Git [18]. Nad každou změnou kódu se spouštěla sada automatizovaných testů (7) a probíhala kontrola nástroji statické analýzy kódu (6.1.0.0.2). Praktiky *Continous Integration* jsou orientovány zejména na efektivní práci v týmu, ale velmi dobře jdou ruku k ruce s agilním přístupem i při solo vývoji. Pravidelné zapracovávání změn umožňuje rychlejší detekci problémů (a případný návrat zpět), a to jak v kódu, tak v návrhu procesu.

Kapitola 3
Požadavky na systém

Před začátkem návrhu procesu a implementace systému bylo uspořádáno několik schůzek se zástupci produktového a vývojového týmu zadavatele za účelem zmapování požadavků na systém. Požadavky byly následně v každé vývojové iteraci validovány a upřesňovány (příp. doplňovány, viz. 2). V této kapitole jsou výsledné požadavky sepsány.

3.1 Funkční požadavky

Funkční požadavky jsou rozděleny do dvou kategorií: požadavky na proces vychystávání (`F*`) a požadavky na údržbu systému (`M*`).

F1 Řídící aplikace systému *PTL* musí být napojena na existující interně vyvíjený skladový systém (1.1.2.0.5), přičemž je možné zavést nové *API* pro komunikaci, nikoliv však měnit existující.

F2 Systém musí zvládat spravovat tisíce skladových lokací se světelnými moduly.

F3 Systém musí být možné škálovat přidáním dalších pracovníků, v rámci praktických mezí (desítky pracovníků na tisíce lokací).

F4 Lokace dedikované pro *PTL* musí dodržovat standardní značení a umožňovat provádění standardních skladových procesů, mj. evidenci a doplňování zboží.

F5 Zboží je možné doplňovat do lokací dedikovaných pro *PTL* bez nutnosti pozastavení procesu vychystávání objednávek.

F6 Pracovník má možnost reportovat chybějící zboží v lokacích v průběhu vychystávání do *WMS*.

F7 Proces bude navržen tak, aby pro transport vychystávaných objednávek využíval dopravníkový pás prostupující celou zónou (1.1.3.0.2).

M1 Systém musí být připraven na přidávání dalších skladových lokací.

M2 Lokace s technickými problémy lze identifikovat a problémové hardwarové komponenty nahradit s minimálním zásahem ze strany vývojového týmu.

3.2 Nefunkční požadavky

N1 Dodavatelem hardwarového řešení systému *PTL* je společnost *Lightning Pick, Matthews International Corporation* (viz. 1.1.4).

N2 Vybraná technologie implementace systému musí odpovídat aktuálně zavedeným technologiím ve vývojovém týmu zabývající se skladovou logistikou, aby bylo možné projekt předat k dalšímu vývoji a údržbě.

N3 Doba odezvy systému při standardních podmínkách nesmí citelně negativně ovlivňovat výkon pracovníků skladu.

N4 V případě výpadku systému nesmí dojít ke ztrátě dat o již dokončených akcích.

N5 V případě výpadku systému se systém musí umět restartovat bez nutnosti ručního zásahu IT pracovníka.

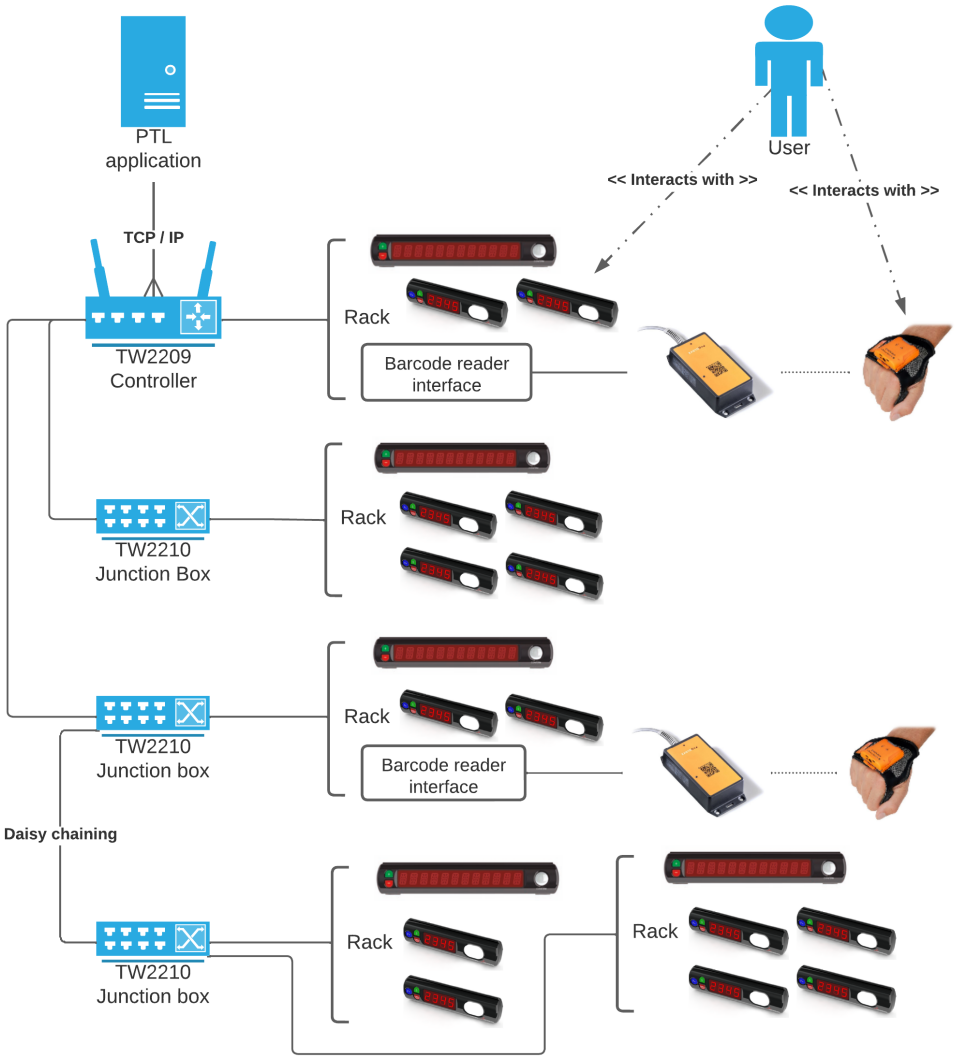
Kapitola 4

Analýza technologie Pick to Light

V této sekci je popsán hardware a komunikační rozhraní technologie *PTL* dodávané společností *Lightning Pick, Matthews International Corporation* (viz. požadavek **N1**). Dodavatel byl vybrán na základě analýzy externích expertů zadavatele s ohledem na kvalitu řešení ve skladovém prostředí a finanční náklady. Následující text vychází z technické dokumentace [19] a [20].

4.1 Hardware

Všechny níže uvedené hardwarové komponenty jsou vzájemně kompatibilní a cíleně určeny k industriálním použití. Ukázkové propojení všech komponent *PTL* systému použitých v testovacích skladech je vyobrazeno na obrázku 4.1. V jednotlivých podkapitolách níže jsou pak komponenty popsány detailněji.

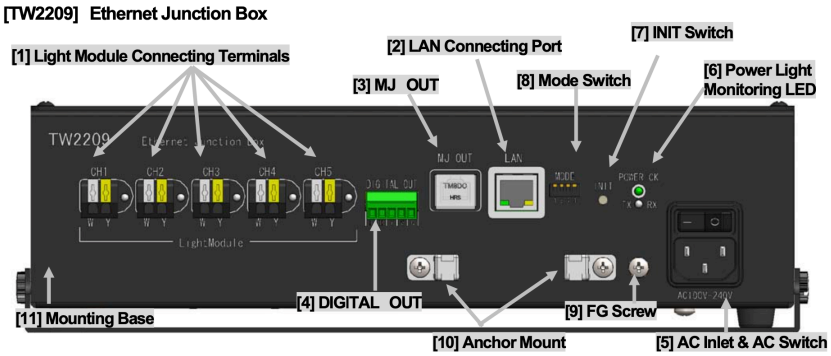


Obrázek 4.1 Vizualizace ukázkového propojení komponent *PTL* systému

4.1.1 TW2209 – Kontrolér

Kontrolér je hlavním komunikačním prvkem *PTL* systému. Je zodpovědný jak za komunikaci s ostatními *PTL* zařízeními, tak za komunikaci s externími systémy skrze proprietární protokol postavený nad *TCP/IP*.

Jak ilustruje obrázek 4.2, kontrolér je vybaven 5 výstupními kanály pro připojení modulů (až 64 modulů na kanál) (ukazatel [1]), *LAN* portem pro připojení k síti (ukazatel [2]) a nestandardním *RJ45* portem (ukazatel [3]) pro připojení *TW2210* (4.1.2).

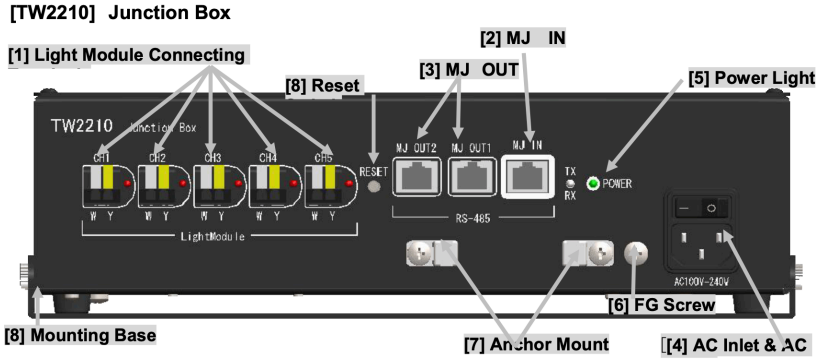


Obrázek 4.2 Lightning Pick TW2209 controller – hardware interface

Základní konfiguraci embedovaného systému (např. sítě, času, …) v kontroléru je možné provést pomocí sady příkazů dostupných přes *TELNET* protokol. Přes toto rozhraní je také možné spustit kontrolu stavu připojených modulů. Výstup kontroly, stejně jako veškeré další logy aktivity kontroléru a k němu připojených zařízení, je vystaven na lokálně běžícím *FTP* serveru.

4.1.2 TW2210 – Junction box

Junction box je přepínač mezi kontrolérem a světelnými moduly operující na linkové vrstvě *OSI* modelu. Z pohledu řídící aplikace jde o kompletně transparentní prvek, který nijak neovlivňuje formát komunikace a není třeba jej manuálně konfigurovat. Junction box je vybaven 5 vstupními kanály pro připojení modulů (až 32 modulů na kanál) (ukazatel [1]), vstupním *RJ45* portem (ukazatel [2]) pro připojení k *TW2209* (4.1.1) či *TW2210* (4.1.2) a dvěma výstupními porty (ukazatel [3]) pro zřetězené připojení dalších *TW2210* (4.3). Kompletní konektivita je zobrazena na obrázku 4.3.



Obrázek 4.3 Lightning Pick TW2210 junction box – hardware interface

4.1.3 Moduly

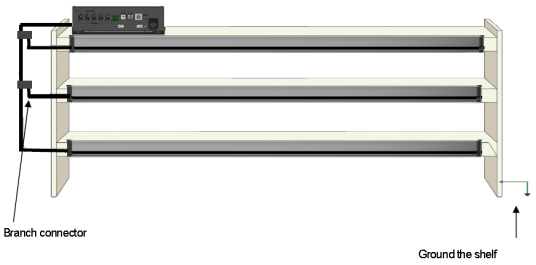
Moduly *PTL* systému lze rozdělit do tří kategorií:

- světelné moduly,
- informační moduly (*Batch display*),
- vstupní zařízení (*Barcode reader interface*).

Každý modul má přiřazený čtyřmístný číselný identifikátor (lze změnit, defaultně nastavený z výroby), přičemž rozsahy jsou vyhrazeny následovně:

- 0–7999 – světelné moduly,
- 8100–8199 – informační moduly,
- 9100–9199 – vstupní zařízení.

Při komunikaci s moduly skrze kontrolér jsou pro adresování používány právě tyto identifikátory nebo jejich *wildcard* varianta – libovolnou číslici v identifikátoru lze při adresování nahradit znakem `?`, čímž dojde k zavolání akce pro všechny moduly obsahující namísto symbolu otazníku libovolnou číslici. Moduly jsou k TW2209 (4.1.1) / TW2210 (4.3) připojeny buď napřímo pomocí modulárního konektoru anebo skrze regálovou lištu (4.4).



Obrázek 4.4 Lightning Pick regál s lištami pro připojení modulů

4.1.3.0.1 Světelné moduly jsou k dispozici v mnoha variantách lišících se primárně rozměry a garantovanou odolností. Rozhraní všech modulů je identické, některé však zvládají složitější konfigurace (např. vícebarevné diody, zvukové signály, …). Pokud modul danou funkcionalitu nepodporuje, je problémová konfigurace ignorována a dál neovlivní chování modulu. Moduly jsou připojeny a napájeny skrze regálovou lištu (viz. obrázek 4.4).

Ve skladu zadavatele byly vybrány moduly řady MW, a to ve dvou variantách:

- single module – modul určený k přiřazení na jednu lokaci,
- double module – dvojitý modul určený k přiřazení na dvě lokace. Z pohledu *PTL* systému vystupuje jako dva samostatné moduly (má dva identifikátory).

Každý světelný modul řady MW je vybaven LED světlem [1], displejem [2], a zvukovou signalizací. Na modulech jsou k dispozici tři tlačítka – hlavní tlačítko sloužící k potvrzení aktuální akce (např. potvrzení vychystání), tlačítko minus a speciální funkční tlačítko Fn. Tlačítka Fn a minus jsou částečně konfigurovatelná, v defaultním nastavení slouží tlačítko Fn k označení prázdné lokace a tlačítko minus cyklicky dekrementuje požadovanou kvantitu zboží na lokaci.

[1] Single moduly podporují RGB, double moduly jsou omezeny na jednu barvu.
[2] Single moduly jsou omezeny na 4 alfanumerické znaky, double moduly pouze 2.

Obrázek 4.5 Lightning Pick MW series – single module

4.1.3.0.2 Batch display je speciální světelný modul, který není určený k vychystávání zboží na lokaci, nýbrž k poskytování dodatečných informací pracovníkům skladu. Typicky je takový modul umístěn jeden v každém regálu, a pracovník skladu na něm může zjistit, v jakém kroku procesu se nachází, a tedy co má provést za další akci.

4.1.3.0.3 Barcode reader interface je modul určený pro připojení čtečky čárových kódů skrze *RS-232C* sériový protokol, který se, navzdory svému stáří, i v dnešní době pro skenery standardně používá. Rozhraní je kompatibilní s libovolnou čtečkou podporující tento sériový protokol. Ve skladu zadavatele byly vybrány moderní bezdrátové čtečky *Proglove Mark 2* [21] spolu s přístupovým bodem (*Proglove Access Point*, obrázek 4.6), který na jedné straně zajišťuje komunikaci se čtečkami skrze Bluetooth a na druhé komunikaci s barcode reader interface přes *RS-232C*.

(a) Proglove Mark 2 – čtečka čárových kódu

(b) Proglove Access Point

Obrázek 4.6 Komponenty Proglove skeneru

4.2 Komunikace s externím systémem

Komunikace probíhá skrze jedno otevřené *TCP* spojení. Pokud je s *PTL* kontrolérem otevřeno spojení nové, předchozí je násilně ukončeno. Přenášené zprávy reprezentují bezstavové příkazy, přičemž každý příkaz musí být druhou stranou potvrzen nebo zamítnut odesláním zprávy se

stejným sekvenčním číslem. Jednotlivé typy příkazů jsou odlišeny pomocí alfanumerických kódů, obsah zprávy je pak závislý příkaz od příkazu, může být buď textový nebo binární.

Celá komunikace je detailně zdokumentovaná v příloze [20], přičemž příkazy použité v této práci jsou popsány v kapitole zabývající se implementací (6.3). Obecně je však lze rozdělit do dvou kategorií: aktivní a jednorázové[3]. Zatímco pasivní příkazy pouze mění interní stav připojených zařízení a dále neovlivňují kontrolér, v případě aktivních příkazů (například aktivace modulu na lokaci) si kontrolér událost pamatuje a naslouchá na změny stavu zařízení pomocí tzv. pollingu. Polling je pro kontrolér relativně náročná operace a v případě hromadných operací je tak vhodnější volit pasivní příkazy (například příkaz na pouhé rozsvícení světla namísto příkazu na kompletní aktivaci modulu). V případě aktivace vyšších desítek modulů na jednom kontroléru naráz hrozí zpomalení celého systému [19].

4.3 Rozložení a zapojení

Pro lepší představu o fyzických možnostech a omezeních při návrhu procesu vychystávání je na diagramu 4.7 zobrazeno fyzické rozložení *PTL* systému v jednom z testovacích skladů.

Obrázek 4.7 Rozložení a zapojení *PTL* v jednom ze skladů zadavatele

[3]Tato terminologie není součástí dokumentace, byla zvolena autorem práce.

Každý růžový obdélník reprezentuje jeden *PTL* regál (4.4), obsahující přibližně 200 lokací rozmístěných do mřížky s 5 vertikálními úrovněmi a jeden batch displej (4.1.3.0.2) umístěný nahoře uprostřed regálu. Každá lokace má přiřazený identifikátor světelného modulu. Lokace ve spodním patře sdílí jeden double modul vždy s lokacemi o patro výše, ostatní úrovně pak klasické single moduly. Regály jsou sdružené do trojic (značení `L01` až `L34`), přičemž každá trojice (tzv. *PTL* sekce) má k dispozici jeden *barcode reader interface*.

V sekcích `L01`, `L11`, `L21` a `L31` jsou umístěny kontroléry (4.1.1), zatímco v ostatních sekcích jsou pouze junction boxy (4.3). Každá vertikální linie sekcí je pak řetězově propojena s kontrolérem. Fakticky se tak z pohledu HW jedná o 4 izolované *PTL* systémy.

4.3.0.0.1 *PTL* lokace jsou standardní skladové lokace opatřené světelným modulem a mají tedy standardní značení odpovídající následujícímu formátu:

`WHA-ZB-LXX-YYY-ZZ`

kde **WHA** je identifikátor skladu, **ZB** identifikátor zóny, **LXX** odpovídá číslu *PTL* sekce, **YYY** je číslo lokace na horizontální úrovni a **ZZ** je číslo lokace na vertikální úrovni. Ukázková lokace tak může vypadat například takto: `FL1-Z1-L01-001-02`. Jde o lokaci v *PTL* sekci `L01`, která se nachází v 1. sloupci a 2. patře. Každá lokace má svůj identifikátor vytištěný ve formě čárového kódu.

Konkrétní realizaci instalace v jednom z testovacích ukazuje obrázek 4.8.

Obrázek 4.8 Ukázka *PTL* instalace v jednom ze skladů zadavatele

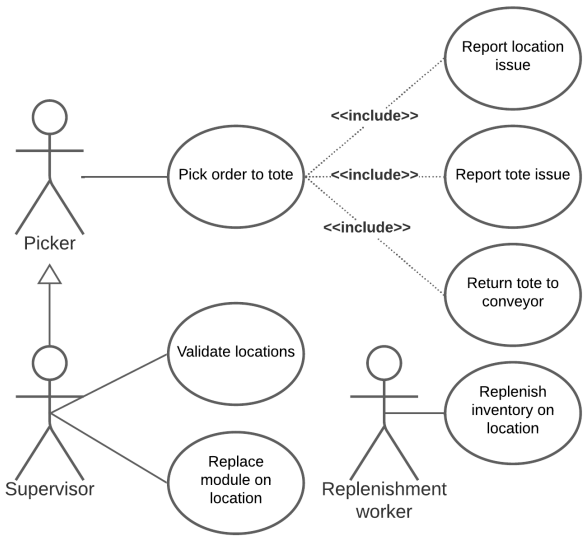
4.4 Existující implementace komunikační vrstvy

Vzhledem k nefunkčnímu požadavku **N1** byl výběr hotových řešení omezen na systém *LP Management Console* [22] poskytovaný právě dodavatelem technologie *PTL*. Tento systém však poskytuje kompletní řešení pro vedení skladu, které je v rozporu s požadavkem **F1**, a navíc je implementován v technologiích (platforma *.NET*), které vývojový tým zadavatele nepoužívá, tedy nesplňuje ani požadavek **N2**.

Žádnou jinou alternativu, ať už ve formě kompletního systému pro vychystání zboží nebo jen *SDK* pro komunikaci, dodavatel hardwaru neposkytuje a nepodařilo se ji najít ani v podobě *open-source* komunitní implementace.

Kapitola 5
Návrh procesů

Na základě analýzy požadavků byly identifikovány následující případy užití popsané diagramem 5.1. Každý z aktérů na diagramu je dále rozpracován jako samostatný proces. Podkapitoly níže zachycují finální podobu navržených procesů spolu s argumentací hlavních rozhodnutí.



Obrázek 5.1 Případy užití *PTL* systému

5.1 Proces vychystávání objednávek

Tato kapitola se zabývá návrhem procesu vychystávání objednávek splňující funkční požadavky z kapitoly 3.1 s přihlédnutím na nastavení existujících procesů v testovacích skladech. Nejprve jsou popsána omezení vycházející z požadavků, poté je zodpovězeno několik zásadních otázek týkajících se návrhu procesu a následně je navržený proces popsán.

Zásadním parametrem při návrhu procesu byla jeho **jednoduchost a podobnost s jinými skladovými procesy**, jelikož pracovníci skladu jsou často brigádníci a je třeba je co nejrychleji zaškolit. Ze stejného důvodu je třeba dbát na to, aby proces dával pracovníkovi **minimální**

prostor pro lidskou chybu. Hlavní flow procesu by tedy mělo být jednoznačné a jednotlivé kroky by měly být návodné. V problémových situacích by měl systém vyžadovat minimum vstupů od pracovníka, přičemž rozhodnutí o další akci s takovou objednávkou by mělo být ideálně přenecháno na specializovaných pracovnících.

Dalším důležitým parametrem návrhu bylo zajištění **kompatibility s existujícími skladovými procesy**. Navržený proces musí pracovat se standardními přepravkami a lokacemi, resp. jejich identifikátory. Ačkoliv na základě požadavku F7 mají být přepravky k vychystání přepravovány pomocí dopravníku, v některých situacích může být třeba odbavit objednávku využívající libovolný jiný způsob přepravy (např. *Pick to Cart*) či balení (*Pick to Carton*). S lokacemi v *PTL* zóně také musí být možné provádět standardní operace inventarizace, tedy přesun zboží, kontrolu lokace apod.

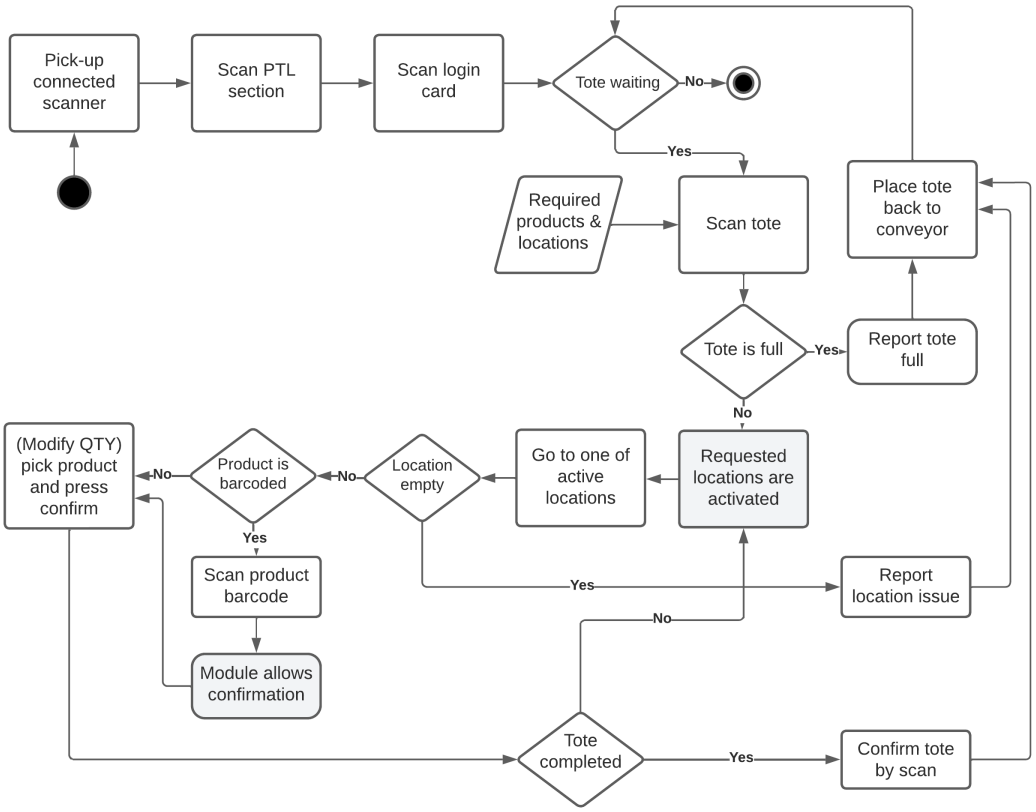
S cílem dosáhnout maximální efektivnosti bylo již při instalaci hardware rozhodnuto o rozdělení *PTL* zóny na sekce vždy po 3 regálech (viz. 4.3), což zajišťuje přirozené oddělení pohybu pracovníků a zároveň ideálně pokrývá průměrnou oblast viditelnosti pracovníka, v rámci které je možné jednoduše identifikovat světelné i zvukové signály bez nutnosti přílišného pohybu. Na základě tohoto rozhodnutí byly navíc instalovány i zastávky na dopravníku, které přesně korespondují s těmito sekcemi. Směrování přepravek je tak efektivní – přepravka zastaví jen v sekcích, ve kterých má alokovanou práci.

Zásadním rozhodnutím při návrhu procesu byla **forma souběžné práce** více pracovníků (požadavek F3). Hlavním problémem souběžné práce na *PTL* systému je schopnost rozlišit stavy a akce jednotlivých pracovníků. Teoreticky by toho šlo docílit například použitím různých barev světelných modulů, kdy by každý z pracovníků následoval pouze sobě přiřazenou barvu. Takové řešení by nicméně bylo velmi náchylné na lidské chyby, navíc světelné moduly ve variantě *double* (4.1.3.0.1) změnu barvy neumožňují. Další variantou je možnost nechat více pracovníků vystupovat v systému pod jedním uživatelem. Toho je možné dosáhnout například připojením více skenerů k jednomu *Proglove Acces Point* zařízení, případně nechat jednoho pracovníka bez skeneru interagovat jen s moduly aktivovaných lokací. Tato varianta je sice technicky funkční, výsledná paralelní efektivnost by ale vzhledem k velikosti sekcí (3 regály) byla velmi nízká – většina objednávka obsahuje nižší jednotky produktů a další pracovník by tak většinu času čekal. Vzhledem k výše uvedeným problémům bylo rozhodnuto, že v **jedné *PTL* sekci může pracovat nejvýše jeden pracovník**. Každý z pracovníků pak **odbavuje objednávky ze své sekce nezávisle na ostatních**.

Stejné omezení bylo zavedeno i na počet sekcí přiřazených k pracovníkovi – vzhledem k tomu, že jedna sekce akorát pokrývá viditelnou oblast, nedává smysl povolovat pracovníkům práci na více sekcích naráz. Kdykoliv v průběhu procesu může pracovník přepnout na jinou sekci a začít vychystávat objednávky z ní.

Další ze zásadních otázek bylo, jakým způsobem má probíhat **komunikace ve směru od systému k pracovníkovi**. Základní variantou bylo využití batch displejů pro zobrazení informací o další očekávané akci ve stylu jednoznačných krátkých příkazů. Takové řešení je ale limitující zejména v situacích, kdy je namísto standardního flow třeba řešit výjimečné situace – například chybějící zboží, přeplněnou přepravku apod. Alternativou je pak přidání externích displejů do každé sekce, zobrazující detailní informace o aktuálním stavu na sekci a případně poskytující možnost interakce. Po několika iteracích návrhu procesu bylo rozhodnuto, že řešení nebude komplikováno zaváděním dalších technologií, v rámci procesu vychystávání bude možné pouze **reportovat chyby** a samotné řešení problémů bude delegováno na tzv. *Problem solving* tým, který je naučený interagovat přímo s *WMS* a který v testovacích skladech v rámci zóny vychystávání již funguje.

Výsledný navržený proces z pohledu pracovníka *PTL* znázorňuje flow diagram 5.2. Standardní průchod je dále textově popsán níže.



Obrázek 5.2 Flow diagram procesu vychystávání z pohledu jednoho pracovníka *PTL* systému.

1. Pracovník přijde do *PTL* zóny se svým skenerem a vybere si sekci
2. Pracovník připojí bezdrátový skener k systému
3. Pracovník naskenuje čárový kód identifikující sekci – *PTL* systém v dané sekci začne na **batch displeji** napovídat ohledně dalších akcích
4. Pracovník se **přihlásí** do *PTL* systému pomocí své identifikační karty [1]
5. Pro každou z přepravek, které zastaví v jeho sekci:
   a. Pracovník **naskenuje identifikátor přepravky**
   b. **Rozsvítí se** všechny **moduly** na lokacích, ze kterých je třeba vychystat zboží. Moduly zobrazí počet kusů, které je třeba vychystat.
   c. Pro každou z rozsvícených lokací:
      i. Pokud je produkt označen čárovým kódem, modul nedovolí vychystání. Pracovník musí nejprve **naskenovat čárový kód produktu**. Až poté se modul přepne do interaktivního stavu.

[1] Identifikační karta může být libovolný řetězec znaků reprezentovaný čárovým kódem. V rámci testovacího skladu byly použity identifikační karty používané pro všechny ostatní skladovací procesy.

ii. Pracovník modifikuje kvantitu dostupnou k vychystání – je možné, že se na lokaci nebude vyskytovat dostatečný počet kusů. Zvolený počet kusů následně přesune do přepravky a potvrdí akci stiskem hlavního tlačítka na modulu.

d. Pracovník znovu naskenuje čárový kód přepravky pro ověření a přepravku vrátí na dopravník.

V rámci hlavního flow byly identifikovány 3 výjimečné situace, které je třeba řešit:

- **chybějící zboží** – v případě, že se na lokaci nenachází žádný produkt, není možné naskenovat čárový kód produktu pro aktivaci modulu,
- **plná přepravka**,
- **problém s čárovým kódem produktu** – kód nesedí na očekávaný produkt nebo je nečitelný.

Pro všechny tři situace jsou vyhrazeny speciální čárové kódy, které může pracovník v průběhu procesu naskenovat, na základě čehož řídící aplikace oznámí problém včetně kontextuálních informací *WMS* systému. Typicky je pak přepravka přesměrována do sekce *Problem solving* a je vyvolán proces kontroly lokace. Kontrola lokace je vyvolána i v případě, kdy na lokaci nějaké zboží je, ale ne v dostatečné kvantitě – pak je vychystána všechna dostupná kvantita, zbývající kvantita objednávky je přealokována jinam a lokace je následně zkontrolována.

5.2 Proces doplňování zboží

Co se doplňování zboží do lokací týče, *PTL* systém neklade žádné specifické požadavky a proces tak může být téměř identický s již zavedeným skladovým procesem doplňováním libovolné lokace v zóně vychystávání zboží.

Doplňování zboží do systému za běhu (požadavek **F5**) je možné ze zadní strany regálů – regálové lokace jsou navržené tak, aby se do nich dalo zasunout několik krabic standardizovaných velikostí se zbožím za sebou. Prázdné krabice se odeberou a recyklují z přední strany v průběhu procesu vychystávání.

Navržená řídící aplikace *PTL* s tímto procesem musí počítat při implementaci, z procesního pohledu jde ale o proces na *PTL* zcela nezávislý.

5.3 Proces údržby

Proces údržby pokrývá funkční požadavky M1 a M2, přičemž údržbu by mělo být nutné provádět jen velmi zřídka, a to speciálně vyškolenými zaměstnanci. Proto byl při návrhu brán ohled zejména na hardwarové možnosti systému *PTL* za cenu menší intuitivnosti procesu.

Obecně lze údržbu v systému *PTL* provádět na třech úrovních:

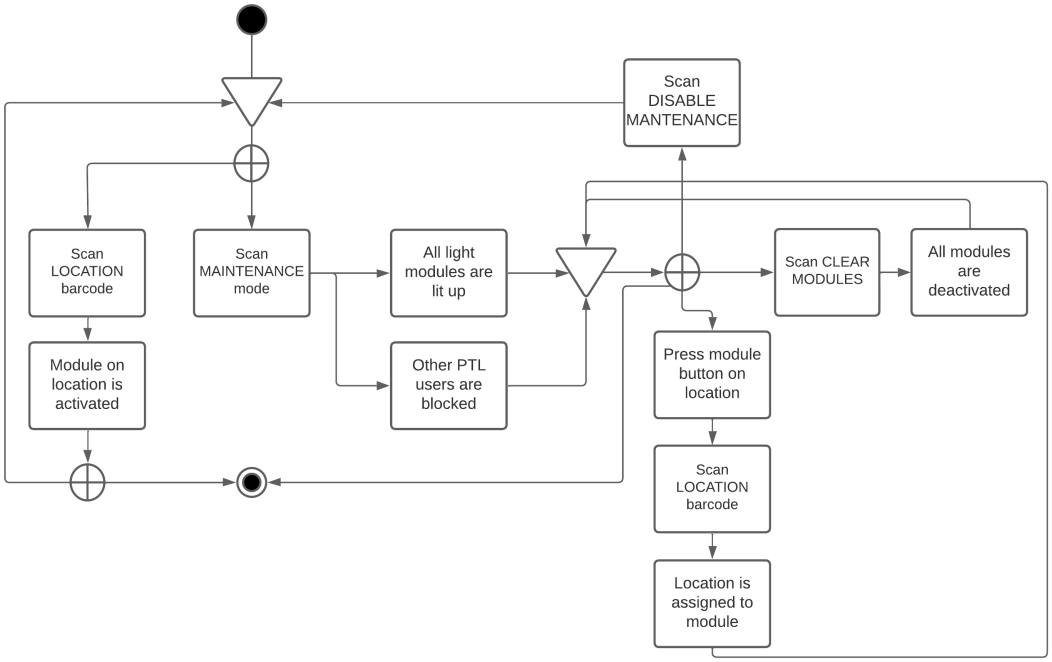
- aplikační – přepnutí aplikace do údržbového módu, který je ovladatelný pomocí standardních skenerů a čárových kódů,
- hardwarová – přepnutí do údržbového módu pomocí hardwarového přepínače na jednotlivých kontrolérech (4.2),
- programová – úpravy kódu řídící aplikace, které jsou prováděny vývojáři.

přičemž jednotlivé úrovně od shora dolů vyžadují více a více odborné znalosti a v případě problémů by tedy měly být zkoušeny v tomto pořadí. Aplikační úroveň pokrývá pouze základní údržbu, a to detekci nefunkčních modulů a jejich výměnu.

Vzhledem k hardwarovým možnostem *PTL* systému je uživatelský vstup i při aplikační údržbě omezen na skenování čárových kódů. Základní detekci vadného modulu lze provést v rámci standardního módu vychystávání, prostým naskenováním čárového kódu reprezentující identifikátor lokace (viz. 4.3). V případě, že je lokace funkční, rozsvítí se jí přiřazený modul[2]. Pokud se modul nerozsvítí nebo dojde k rozsvícení modulu na jiné než skenované lokaci, je třeba přejít k dalšímu kroku.

Pro úpravu modulů přiřazených k lokacím je třeba naskenovat speciální čárový kód povolený pouze vyškoleným pracovníkům, který celou řídící *PTL* aplikaci přepne do údržbového módu, při kterém je interakce ostatních uživatelů se systémem zablokována. Světelné moduly lze následně vyměňovat a přiřazovat k lokacím přidržením tlačítka na modulu a naskenováním čárového kódu příslušné lokace.

Celý údržbový proces je znázorněn flow diagramem 5.3. Složitější operace údržby, jako přidání dalších skenerů, není součástí standardního procesu a je třeba je provádět pomocí speciálních skriptů využívajících modul komunikace řídící aplikace za pomoci IT pracovníka.



Obrázek 5.3 Flow diagram procesu údržby.

[2]Nejde o standardní aktivaci modulu, pouze o rozsvícení světla.

Kapitola 6
Implementace

Tato část práce popisuje implementaci řídícího software *PTL* systému na základě procesů navržených v kapitole 5, který vyhovuje požadavkům stanoveným v kapitole 3.

6.1 Výběr technologií

Jedním z požadavků na systém (N2) byl výběr takových technologií a nástrojů, které je zadavatel schopen provozovat a udržovat, a to nejen s ohledem na licenční a finanční nároky, ale také schopností a zkušeností jeho dosavadního vývojového týmu.

Na základě diskuze s vývojovým týmem byly mezi hlavními kandidáty programovací jazyky PHP, JavaScript/TypeScript (NodeJS) a případně Python.

Jelikož je aplikace z povahy problému silně řízená událostmi (event-driven) [23], a jedním ze zásadních požadavků je schopnost systém škálovat navyšováním počtu aktivních uživatelů (pracovníků), je důležité systém navrhnout tak, aby jednotlivé akce nebyly vzájemně blokující.

Toho lze docílit buď použitím asynchronního programovacího modelu, nebo použitím více vláken (příp. procesů). Všechny tři výše uvedené jazyky a jejich běhová prostředí (alespoň ve standardní konfiguraci) mají podporu pro práci s vlákny, avšak ne ve standardním slova smyslu.[1]

Naopak asynchronní programování je nativně podporováno v jazycích JavaScript (NodeJS) i Python a je možné jej dosáhnout i v PHP, například pomocí rozšíření *ReactPHP*.

Z důvodů popsaných v následujícím odstavci byl pro realizaci vybrán jazyk TypeScript s běhovým prostředím NodeJS.

Jazyk PHP byl kvůli omezenému výběru knihoven podporujících asynchronní I/O a výrazně menší komunitě asynchronní I/O využívající z výběru vyřazen. V případě jazyka Python lze blokující knihovny adaptovat na neblokující s použitím vláken, ale Python je ve vývojovém týmu zadavatele využíván týmem zabývajícím se primárně datovou analýzou, nikoliv vývojem *WMS*, čímž je zadavatelem silně defavorizován.

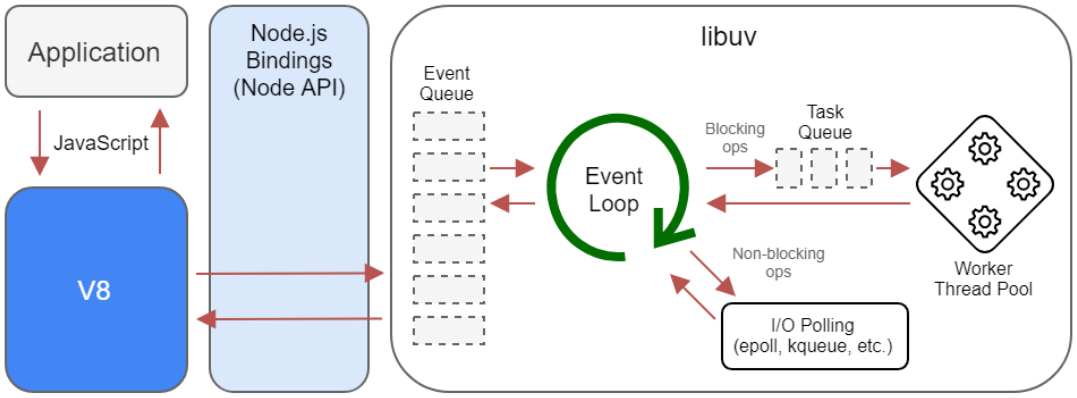
6.1.0.0.1 TypeScript je nadstavba jazyka JavaScript přidávající primárně syntax pro typový systém, nad kterým je následně možné provádět statickou typovou kontrolu [24]. Kód napsaný v jazyce TypeScript je obvykle source-to-source kompilátorem (transpiler) převedený do jazyka JavaScript, který je následně interpretován v prohlížeči nebo jiném běhovém prostředí – na serveru (a tedy potenciálně v případě řídící *PTL* aplikace) standardně NodeJS).

[1] NodeJS poskytuje pouze tzv. Worker Threads, což je spíše obdoba procesů, PHP je potřeba zkompilovat v tzv. Zend Thread Safe módu spolu s rozšířením *ext-parallel* a v případě standardního Python interpreteru je omezením tzv. GIL (Global Interpreter Lock), který zabraňuje paralelnímu běhu kódu v rámci jednoho procesu.

Spouštěný uživatelský kód běží v prostředí NodeJS v jednom vlákně. Paralelního zpracování lze docílit pomocí *Worker threads*, které však nereprezentují vlákna v tradičním smyslu – jde o izolované podprocesy, se kterými lze komunikovat pouze pomocí techniky *Message Passing*.

Tento nedostatek kompenzuje nativní podpora pro asynchronní I/O zajištěná pomocí tzv. *event loop* [25], což je technika kooperativního multitasking, kdy běhové prostředí registruje události do smyčky událostí, která se nějakým způsobem [2] postará o jejich zpracování a informaci o výsledku předá zpět – více informací viz. diagram 6.1 a [23].

Velkou výhodou oproti jazykům PHP i Python je fakt, že naprostá většina standardní knihovny i knihoven třetích stran asynchronní I/O využívá, zatímco pro Python a PHP obzvlášť je nabídka knihoven připravených na neblokující volání omezená.



Obrázek 6.1 Ilustrace smyčky událostí (Event loop) v prostředí NodeJS

6.1.0.0.2 Statická analýza kódu Statická analýza kódu umožňuje odhalovat chyby programu před spuštěním programu (např. v době kompilace). Nejtypičtější statickou analýzou je typová kontrola, která je v jazyce TypeScript nativní. V rámci realizace této práce byl kompilátor jazyka TypeScript nakonfigurován na nejvyšší možnou úroveň striktnosti. Pro zajištění konzistentního kódu napříč projektem a zamezení častých chybných vzorů v kódu byl dále použit nástroj *ESLint* [26] s přídavnou podporou pro TypeScript.

6.1.0.0.3 API pro komunikaci s WMS V rámci této práce bylo třeba definovat *API* (dále popsané v kapitole 6.5), které bude řídící aplikace *PTL* provolávat pro komunikaci s obecným *WMS*.

Na základě komunikace s vývojovým týmem *WMS* systému z testovacích skladů byl zvolen návrh *API*, které je bezstavové, využívá protokol *HTTP* a data přenáší ve formátu *JSON*, ale nedrží se přímo konvencí *REST* [27]. V prostředí *WMS* jsou totiž modelovány především akce a procesy, které je složité popsat pomocí zdrojů (resources), jak je definuje *REST* a omezené sady *HTTP* metod. Endpointy *API* jsou zároveň vytvářeny pro konkrétní klienty (aplikace) na základě potřeby a například užitečnost cachování výsledků na úrovni *HTTP* je v rychle měnícím se prostředí *WMS* zanedbatelná. API tedy respektuje konvence *HTTP*, ale ne všechny konvence *REST*.

Pro implementaci *API* se nabízí dvě varianty – rozšíření existujícího API *WMS* nebo vytvoření samostatné aplikace, která se bude starat o překlad *API* definovaného *PTL* systémem na volání existujících *WMS* akcí.

[2] pomocí neblokujících systémových volání nebo vláken, z pohledu běhového prostředí jde o implementační detail

Výsledná (referenční) implementace tohoto *API* v rámci *WMS* zadavatele není přímou součástí rozsahu této práce, nicméně pro dovedení práce do funkčního stavu ji bylo nutné provést.

Referenční implementace byla vytvořena v jazyce PHP, s použitím frameworku Symfony, knihovny Doctrine ORM a MySQL databáze.

6.2 Návrh architektury řídící aplikace

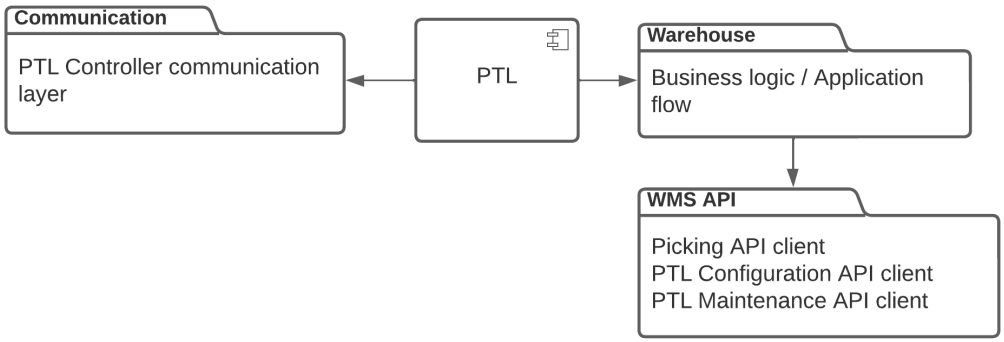
Aplikaci lze rozdělit na následující 3 moduly, jejichž provázání znázorňuje diagram 6.2:

- komunikaci s *PTL* Hardware (6.3),
- business logika řídící aplikace (6.4),
- komunikaci s *WMS* systémem (6.5).

Při návrhu modulu komunikace s *PTL* Hardware byl brán ohled na případnou podporu dalších typů/poskytovatelů *PTL* Hardware bez velkých zásahů do business logiky řídící aplikace. Jelikož je ale pravděpodobnost změny poskytovatele *PTL* Hardware vzhledem k počáteční investici velmi malá, v některých částech kódu se upřednostnila jednoduchost před flexibilitou. Do návrhu mezimodulové komunikace to však prakticky nezasahuje. Zásadnějším parametrem pro tento modul byla testovatelnost.

Komunikace s *WMS* systémem už je ze strany řídící aplikace více spjatá s procesem vychystání objednávek (podstatná část stavu systému je držena na straně *WMS*, viz. kapitola 6.5) a případné nutnosti napojit řídící aplikaci na jiný *WMS* systém by předcházelo doimplementování API buď přímo do *WMS* nebo samostatné aplikace řešící pouze překlad mezi API *WMS* a očekávaným API řídící aplikace.

Aplikace je navržena tak, aby komunikovala s celým *PTL* systémem, který je složen z jednoho či více *PTL* kontrolérů (4.1.1), stovek až tisíců připojených modulů a desítek pracovníků. Zároveň byl kladen důraz na rychlou odezvu, separaci chování systému pro jednotlivé uživatele a schopnost zotavit se z (nejen) síťových problémů.



Obrázek 6.2 Moduly řídící aplikace

6.2.0.0.1 Návrhový vzor *Failable<R, E>* (implementován knihovnou *ts-failable*) je návrhový vzor (patřící mezi monády), známý (primárně z domény funkcionálního programování) také pod názvy *Result* nebo *Either*. Jedná se o alternativní přístup k propagaci výjimek napříč

kódem, který je kontrolovaný typovým systémem jazyka, a to i v asynchronním kódu, ve kterém je velmi obtížné provádět statickou analýzu vyhazovaných výjimek.

Výjimky jsou v aplikaci použity pouze v situacích, ze kterých se nelze zotavit, zatímco v situacích, kdy špatný výsledek operace lze ošetřit, je vrácen *Failable<R, E>* objekt.

6.3 Modul komunikace s *PTL*

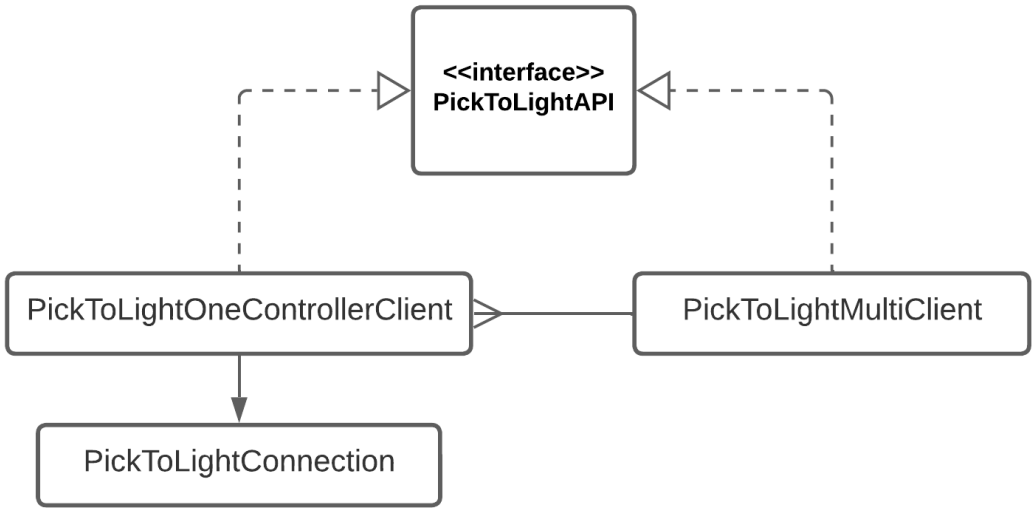
V této sekci jsou popsány základní implementační koncepty komunikace s *PTL*. Detailní popis celého komunikačního protokolu je k dispozici v příloze [20].

Modul komunikace s *PTL* kontrolérem je stěžejní součástí výsledné řídící aplikace, ale je možné jej používat i samostatně například pro účely testování, konfigurace skenerů a jiných zařízení apod (viz. složka *src/scripts* v přílohách).

Důležitou otázkou při návrhu bylo, zda má aplikace komunikovat vždy pouze s jedním *PTL* kontrolérem a *PTL* systém o více kontrolérech tak bude obsluhován více instancemi aplikace nebo zda má aplikace obsluhovat celý *PTL* systém. Výhodou provozu více samostatných instancí je, že v případě pádu jedné instance (např. z důvodu selhání hardwaru kontroléru) zůstane zbytek systému funkční. Mezi nevýhody naopak patří složitější proces nasazování a monitorování aplikace a omezení zapojení *PTL* hardwaru – v rámci jedné *PTL* sekce by musela všechna zařízení spadat pod jeden kontrolér.

S přihlédnutím na výše zmíněné body bylo navrženo řešení, které podporuje oba způsoby provozu. Jedna řídící aplikace může obsluhovat více *PTL* kontrolérů najednou a zároveň je možné provozovat více instancí aplikace v rámci jednoho skladu – například v situacích, kdy se ve skladě nachází více fyzicky oddělených *PTL* systémů.

6.3.1 PickToLightAPI



Obrázek 6.3 Diagram tříd pro komunikaci s *PTL*

Základním prvkem modulu pro komunikaci s *PTL* je rozhraní *PickToLightAPI*, které definuje obecné operace proveditelné nad *PTL*, mimo jiné:

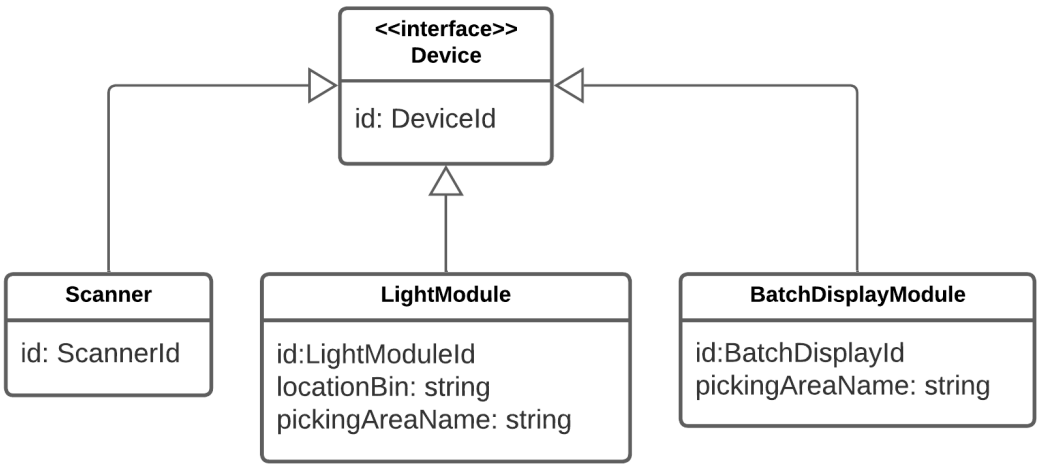
- připojení a inicializace *PTL*,

- aktivace modulu,
- deaktivace modulu,
- zobrazení textu na pomocném displeji (Batch Display),
- naslouchání na uživatelské vstupy.

Konkrétními implementacemi tohoto rozhraní (viz. diagram 6.3) jsou třídy *PickToLightOneControllerClient* a *PickToLightMultiClient*. Instance třídy *PickToLightOneControllerClient* si drží informace o zařízeních připojených k jednomu *PTL* kontroléru a spojení s ním reprezentované třídou *PickToLightConnection*.

PickToLightMultiClient je dekorátor, který umožňuje operovat s několika instancemi *PickToLightAPI* (a tedy několika fyzickými kontroléry) najednou skrze původní *PickToLightAPI* rozhraní – implementace se stará o směrování operací na konkrétní instance a mapování výsledků operací do původní formy.

PTL systém rozlišuje 3 typy zařízení vyobrazených na grafu 6.4. Informace o připojených zařízeních je třeba



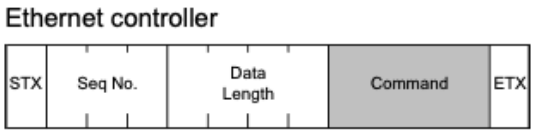
Obrázek 6.4 Diagram tříd *PTL* zařízení

6.3.2 PickToLightConnection

Třída *PickToLightConnection* řeší obecné odesílání a přijímání zpráv z *PTL* kontroléru přes persistentní TCP/IP socketovou komunikaci. Socket je reprezentována nativní třídou *net.Socket* obalenou do třídy *PromiseSocket* (z balíčku *promise-socket*), která přidává podporu pro rozhraní využívající Promise API.

PTL kontrolér drží nejvýše jedno otevřené spojení na vyhrazeném portu pro komunikaci s řídící aplikací. V případě, že je navázáno nové spojení, je třeba celé komunikační flow začít od znovu.

Komunikace probíhá ve formě příkazů. Každý příkaz se skládá z počáteční kontrolní sekvence (0x02), hlavičky, datového bloku a koncové kontrolní sekvence (0x03). Hlavička obsahuje sekvenční číslo příkazu a informaci o délce datového bloku. Formát příkazu je znázorněn na obrázku 6.5 s přidruženými popisky polí v tabulce 6.6.



Obrázek 6.5 Formát příkazu pro komunikaci s *PTL*

Command	Command
STX	Beginning of the command: 02h
ETX	At the end of the command: 03h
Seq No.	Sequence number: 000 ~ 999
Data Length	Command length: 1 ~ 1015 (byte)
SM	Checksum
CR	End of command: 0Dh

Obrázek 6.6 Popis jednotlivých polí formátu příkazu pro komunikaci s *PTL*

Jednotlivé příkazy jsou identifikovány pomocí kódu na začátku datových bloků, přičemž zprávy přicházející od *PTL* kontroléru do řídící aplikace jsou vždy identifikovány malými písmeny, zatímco zprávy odesílané do *PTL* kontroléru používají velká písmena.

Každý odeslaný příkaz musí být potvrzen zprávou obsahující ‘O‘ (potvrzení)/ ‘N‘ (odmítnutí), resp. ‘o‘ / ‘n‘, která v hlavičce obsahuje stejné sekvenční číslo jako odeslaný příkaz. Zatímco aplikace komunikující s *PTL* kontrolérem musí potvrzení odeslat ihned po přijetí zprávy, *PTL* kontrolér může začít odesílat své vlastní zprávy před odesláním potvrzení.

Kvůli tomuto omezení bylo třeba komunikaci naimplementovat tak, aby byly jednotlivé příkazy v rámci jednoho spojení serializovány a odesílány synchronně (ale bez blokování zbytku aplikace), čehož bylo dosaženo pomocí balíčku *async-lock*, který implementuje zámky velmi jednoduše pomocí *Promise API*.

Vedle odesílání zpráv je třída *PickToLightConnection* schopna registrovat callback volaný pro každou přijatou zprávu. V situacích, kdy dojde k výpadku spojení nebo je nutný restart z jiných důvodů poskytuje třída *PickToLightConnection* podporu pro kompletní restart spojení.

6.3.3 Odchozí *PTL* příkazy

Níže jsou vypsány příkazy odesílané z řídící aplikace do *PTL* kontroléru, které jsou nutné pro proces vychystávání. Další příkazy, včetně příkazů specifických pro mód údržby, jsou popsány v dokumentaci *PTL* kontroléru.

6.3.3.0.1 Inicializace – Z Příkaz, který je třeba poslat po navázání spojení – kontrolér po přijetí resetuje všechny moduly do základního nastavení a připraví se na příchozí příkazy. Všechny moduly jsou deaktivovány a skenery zatím neposílají příkazy.

6.3.3.0.2 Aktivace skeneru – G Příkaz, který aktivuje skener s daným identifikátorem. *PTL* kontrolér následně začne přeposílat vstupy načtené pomocí skeneru připojeném k danému *barcode reader interface*.

6.3.3.0.3 Aktivace světelného modulu – PP5 Příkaz, který aktivuje modul s daným identifikátorem (PP5) a na displeji zobrazí zadaný text o maximální délce 5 alfanumerických

znaků. Pro korektní funkcionalitu všech tlačítek modulu je očekáváno, že text bude složen pouze z číslic.

6.3.3.0.4 Deaktivace světelného modulu – D Příkaz, který deaktivuje modul s daným identifikátorem, nehledě na příkaz, kterým byl aktivován.

6.3.3.0.5 Konfigurace světelného modulu – Am1 Příkaz, který umožňuje nakonfigurovat chování modulu v rámci jednoho spojení. Definuje barvy diod, zvukové prvky, chování tlačítek. Pro sestavení konfigurace je k dispozici *DeviceConfigurationBuilder*.

6.3.3.0.6 Zobrazení textu na Batch Display modulu – X Příkaz, který zaktivuje Batch Display modul a zobrazí zadaný text o maximální délce 12 alfanumerických znaků.

6.3.3.0.7 Změna identifikátoru modulu – Ac Příkaz, který umožňuje přepsat identifikátor modulu. Součástí vstupu je nový identifikátor modulu. Příkaz nehlídá duplicity a povolí přiřazení stejného identifikátoru několika modulům. Lze použít na všechny typy modulů, včetně *barcode reader interface*. Příkaz není adresovatelný – *PTL* systém dohledá modul se stisknutým hlavním tlačítkem (4.1.3.0.1) v momentě přijetí příkazu a přepíše jeho identifikátor. Speciálním případem jsou double moduly, jejichž identifikátory jsou adresovány vždy společně. Nehledě na stisknuté tlačítko je vždy pravé polovině modulu přiřazen identifikátor o 1 vyšší než levé.

6.3.4 Příchozí *PTL* příkazy

6.3.4.0.1 Vstup ze skeneru – t91 Příkaz, který je odesílán při každém načtení čárového kódu pomocí skeneru. Obsahuje identifikátor skeneru, a text, který skener načetl.

6.3.4.0.2 Stisk tlačítka – t Příkaz, který je odesílán při stisku tlačítka světelném modulu, který byl předtím aktivován pomocí příkazu *PP5*. Obsahuje identifikátor modulu, stav – OK / Nedostatek produktu / Vadný modul a v případě OK stavu kvantitu zobrazovanou na displeji.

6.4 Business logika řídící aplikace

Veškerá business logika řídící aplikace implementující navržené procesy (5) se nachází v modulu *src/warehouse*.

Základními cíli při návrhu aplikace byla jednoduchá implementace jednotlivých akcí procesu, jejich testovatelnost a atomicita provedených akcí tak, aby byl systém vždy v konzistentním stavu s *WMS*.

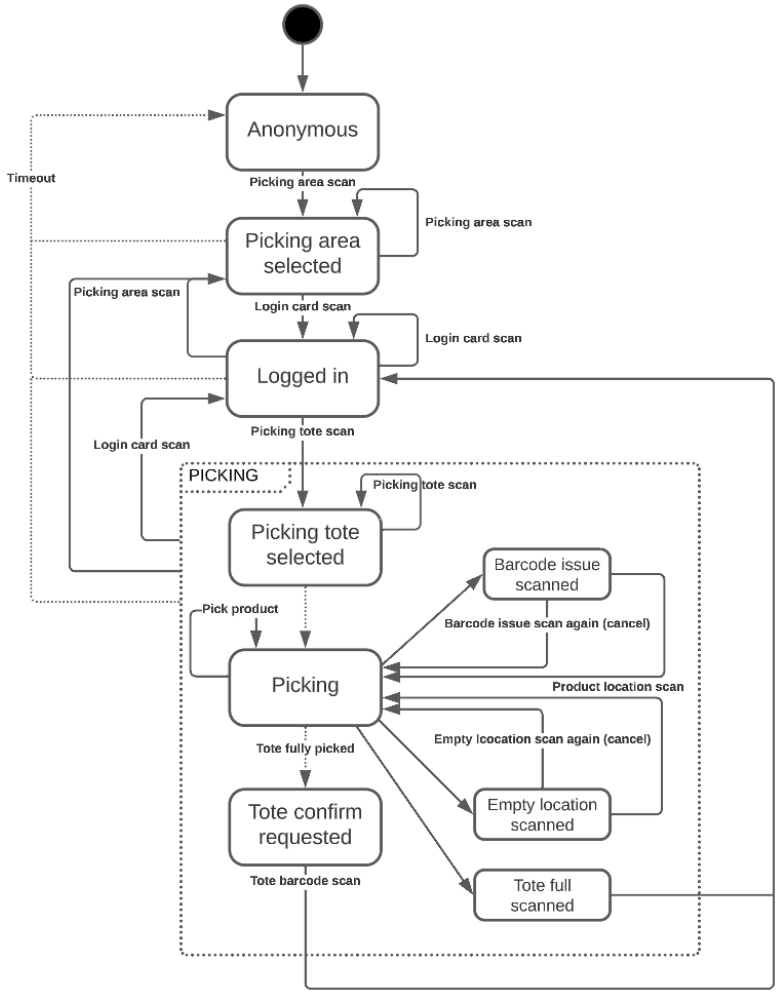
Z tohoto důvodu byla architektonická rozhodnutí směřována na maximální bezstavovat systému, s výjimkou dvou entity: `PickToLightUser` a *PTL*.

6.4.0.0.1 Třída PickToLightUser reprezentuje stav jednoho uživatele *PTL* systému. Stavy uživatele a jejich přechody jsou reprezentovány stavovým automatem vyobrazeným na grafu v 6.7 a odpovídají flow diagramu vychystání objednávek, viz. graf 5.2.

Následující flow by se mohlo zdát relativně komplikované, ale díky návodným zprávám na batch displejích umístěných v každé sekci se v praxi ukázalo být rychlé a intuitivní.

Každý *barcode reader interface* reprezentuje jednoho unikátního uživatele. Uživatel je vytvořen ve stavu `ANONYMOUS` při přijetí první zprávy z daného *barcode reader interface*.

Dalším krokem je přihlášení uživatele do konkrétní *PTL* sekce. Pracovník se tedy přesune do příslušné sekce a naskenuje čárový kód reprezentující její identifikátor – identifikátor sekce je fixní a je umístěn v dané sekci na viditelném místě *PTL* konstrukce, čímž se přesune do stavu



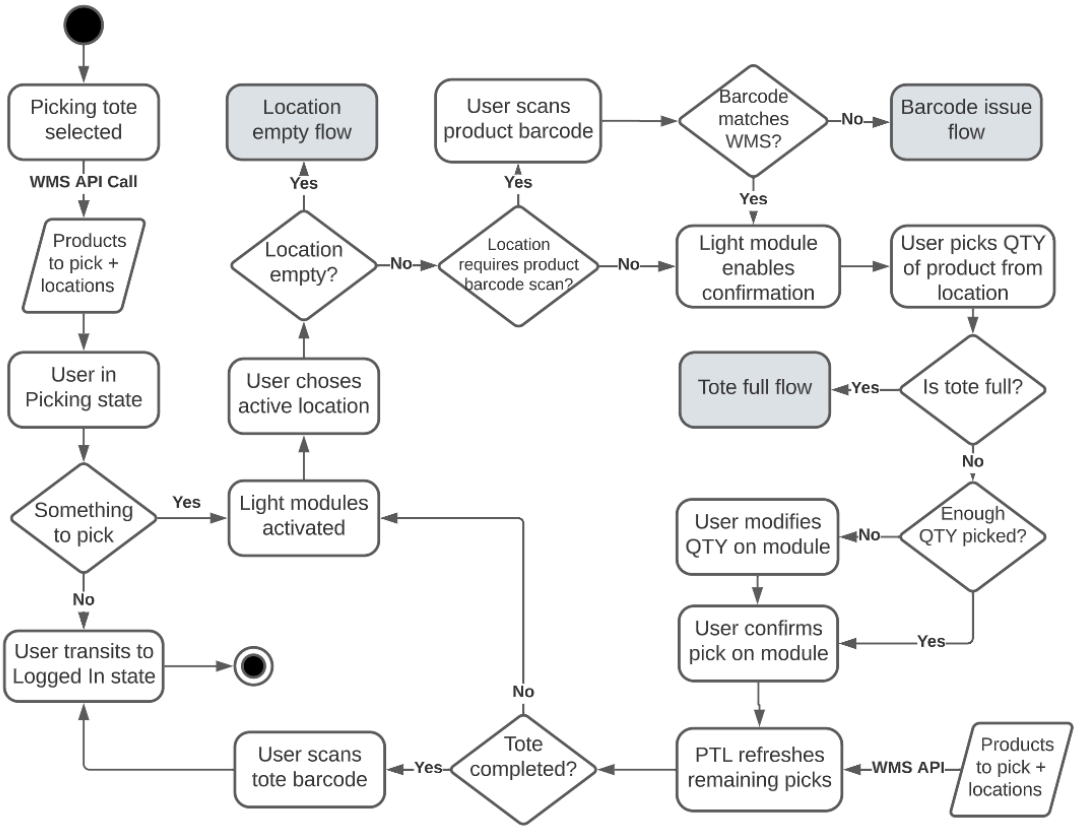
Obrázek 6.7 Stavy a přechody uživatele *PTL* systému

`PICKING_AREA_SELECTED`. Po zvolení sekce se aktivují batch displeje (viz. 4.1.3) přiřazené k dané sekci, na kterých je zobrazena nápověda pro další očekávanou akci a krátce zobrazené zprávy v reakci na poslední provedenou operaci.

Třetím krokem, již s nápovědou na batch displeji, je naskenování přihlašovací karty pracovníka. Přihlašovací karta je unikátní pro každého pracovníka a je generována a validována *WMS*. V případě *WMS* zadavatele je přihlašovací karta používána napříč všemi procesy, a nejedná se tedy o specifikum *PTL* systému, nicméně tento fakt je implementační detail – *WMS* musí pouze implementovat endpoint pro validaci identifikátoru uživatele (popsáno dále v kapitole 6.5). Po zvalidování přihlašovací karty se uživatel přesune do stavu `LOGGED_IN`.

V tomto stavu může pracovník začít vychystávat objednávky. Z čekací odbočky na dopravníku vyhrazené pro danou sekci *PTL* pracovník odebere krabici a naskenuje její čárový kód. Tím se přesune do stavu `PICKING` a *PTL* aplikace se dotáže *WMS* na potřebné informace o produktech, které je třeba vychystat. Proces vychystání je už o něco složitější a je proto rozkreslen v diagramu v 6.8.

Po přijetí odpovědi se aktivují příslušné světelné moduly a zobrazí buď kvantitu, kterou je třeba přemístit z lokace do krabice, nebo, v případě produktů s čárovým kódem, kód ‘BC‘ značící, že je třeba prvně naskenovat čárový kód produktu. Tento mezikrok validace čárového kódu



Obrázek 6.8 Diagram aktivit pro část procesu vychystávání objednávky

produktu byl přidán po prvotním testovacím běhu s cílem minimalizace chyb při vychystávání objednávek, které vzniknou již při doplňování zboží, a nejsou nutně způsobeny chybou pracovníka *PTL*. Po naskenování čárového kódu je světelný modul odblokován a na displeji se již zobrazuje kvantita produktu, kterou je třeba přemístit z lokace do krabice.

Pokud *WMS* vrátí prázdný seznam, například z důvodu, že ke krabici není přiřazena žádná objednávka, v dané sekci se nenachází potřebné produkty nebo jsou již všechny produkty v krabici vychystány, na batch displeji se na okamžik zobrazí zpráva "PC COMPLETED" a uživatel se přesune zpět do stavu `LOGGED_IN`.

V ideálním případě pracovník postupně odbavuje jednotlivé lokace s rozsvícenými moduly, přičemž *PTL* systém o každém potvrzeném modulu ihned notifikuje *WMS*. Pokud na lokaci není dostatečná kvantita produktu, může pracovník pomocí tlačítka '-' na modulu snížit vychystávanou kvantitu, přičemž tato informace je ihned odeslána *WMS* – v takovém případě *WMS* přealokuje chybějící kvantitu zboží na jinou lokaci a spustí proces kontroly lokace (viz. 1.2.5).

Na konstrukci *PTL* dané sekce jsou dále umístěny tři speciální čárové kódy. V případě, že se vychystávaný produkt do krabice nevejde, může pracovník naskenovat speciální čárový kód `TOTE_FULL`, čímž *PTL* systém předá *WMS* signál, že je krabice plná a objednávku je třeba dokončit jinými procesy.

Pokud naopak na lokaci zboží chybí úplně, je poškozené, nebo je jinak nevhodné pro vychystávání, může pracovník naskenovat tento speciální čárový kód a následně identifikátor lokace, čímž označí lokaci jako problémovou, na základě čehož *WMS* přealokuje veškeré zboží na alternativní lokaci obsahující stejné zboží a nastartuje proces kontroly lokace, viz. 1.2.5.

Posledním ošetřeným speciálním případem je situace, kdy čárový kód produktu nesedí nebo jej

nelze naskenovat – v takovém případě pracovník může naskenovat speciální čárový kód a následně identifikátor lokace produktu, čímž dojde k přeskočení lokace a odeslání informace o problémovém čárovém kódu *WMS*.

Pokud byly všechny lokace v dané sekci vychystány nebo došlo k některé ze speciálních situací uvedených výše, musí uživatel na závěr naskenovat čárový kód krabice, s cílem zjistit, zda celou dobu neprováděl operace s nesouvisejícími krabicemi. Jde o kompromis mezi snahou o minimalizaci chyb při vychystávání objednávek a komplikací procesu – bylo by bezpečnější krabici skenovat pro potvrzení každé akce, ale to by byl značný zásah do UX (*User Experience*).

Krabici, pro jejíž objednávku už nezbývá v dané *PTL* sekci práce, pracovník odkládá zpět na dopravník a krabice odjíždí na další stanoviště dle rozhodnutí *WMS*.

6.4.0.0.2 Uživatelské vstupy aplikace lze rozdělit do dvou kategorií – vstupy generované čtečkami čárových kódů a reakce *PTL* systému na události (kompletace aktivního modulu stiskem tlačítka). Zatímco zprávy z *PTL* systému mají fixní formát (viz. kapitola 6.3), čtečky přečtou libovolný čárový kód jako sekvenci alfanumerických znaků. [3] Čárové kódy pro *PTL* systém byly tedy navrženy tak, aby bylo možné jednotlivé akce doménové logiky jednoduše rozlišit regulárními výrazy a aby zároveň odpovídaly konvencím čárových kódů již zavedených ve skladu jinými procesy.

6.4.1 Warehouse Event

Při návrhu aplikace byl kladen důraz na maximální oddělení business logiky aplikace od komunikace s *PTL* systémem, a to primárně z těchto důvodů:

- zvýšení přehlednosti kódu – logika je modelována jen jako přechody stavů uživatele, bez přímého ovlivňování fyzického stavu *PTL* systému,
- zjednodušení unit testů na úrovni jednotlivých akcí,
- schopnost se vrátit do původního stavu při výpadku spojení či jiné komunikační chybě.

Základním prvkem business logiky aplikace je třída `WarehouseEvent`, která reprezentuje jednu obecnou akci uživatele / *PTL* systému už z pohledu řídící aplikace, nikoliv čisté komunikace s *PTL* systémem. Mezi podtřídy `WarehouseEvent` patří:

- `PickingAreaScanEvent`
- `LoginCardScanEvent`
- `ProductPickEvent`
- `BarcodeScanEvent`
- `ToteIsFullEvent`
- `BarcodeIssueEvent`
- `EmptyLocationEvent`

Instance podtříd třídy `WarehouseEvent` jsou vytvářeny z příchozích zpráv (příkazů) *PTL* systému třídou `WarehouseEventDecoder`. WarehouseEventDecoder nebere ohled na aktuální stav uživatele, pouze mapuje obecnou komunikaci na bussinesové události. Pro jednotlivé podtřídy

[3] Používané čtečky přečtou kód v prakticky libovolném standardním formátu (Code 39, Code 128, EAN, UPC). Ačkoliv některé standardy omezují rozsah povolených vstupů, a naopak zajišťují vyšší spolehlivost, finálním výstupem je vždy alfanumerická sekvence.

jsou pak registrovány instance podtříd `WarehouseEventHandler`, které implementují business logiku operace, již pro konkrétní instanci uživatele.

Aplikace tak může pro stejné eventy definovat různé handlery na základě aktuálního stavu aplikace – toho je využito pro rozdělení operačních módů na klasický proces vychystávání (jedna sada event handlers) a proces údržby (jiná sada).

Další důležitou komponentou modulu je třída `PickToLightStateResolver`, která se stará o transformaci aktuálního stavu uživatelů *PTL* aplikace do příkazů pro *PTL* systém. Metoda `PickToLightStateResolver.resolvePickToLightStateFromUserState` přijímá aktuální stav uživatele a provede veškerou potřebnou komunikaci s *PTL* kontroléry tak, aby se systém z libovolného stavu dostal do stavu odpovídajícímu aktuálnímu stavu uživatele. Tato metoda je volána na základě návratové hodnoty z provedené `WarehouseEventHandler` akce, jde ale pouze o optimalizaci, rezoluci lze volat kdykoliv. Hlavními výhodami tohoto přístupu je striktní oddělení business logiky od komunikace s *PTL* systémem (tudíž jednodušší testovatelnost) a možnost kdykoliv obnovit stav, například při chybě v síťové komunikaci.

Pro zobrazování uživatelských zpráv na batch displejích využívá `PickToLightStateResolver` třídu `PickToLightBatchDisplayMessageService`. Uživatelské zprávy jsou totiž v aplikaci dvojího typu – zprávy reprezentující uživatelské pokyny, které jsou pevně provázané s uživatelským stavem a krátce zobrazované zprávy, které je možné zobrazovat kdykoliv v reakci na jakoukoli událost (chybové hlášky, informace o stavu systému, provedené akce atd.).

Proces údržby se z tohoto konceptu vymyká – narozdíl od procesu vychystávání běží v módu jednoho uživatele a event handlery si tak mohou dovolit komunikovat s *PTL* systémem napřímo (skrze operace definované v `MaintenanceService`) namísto přepisování stavu `PickToLightUser` entity.

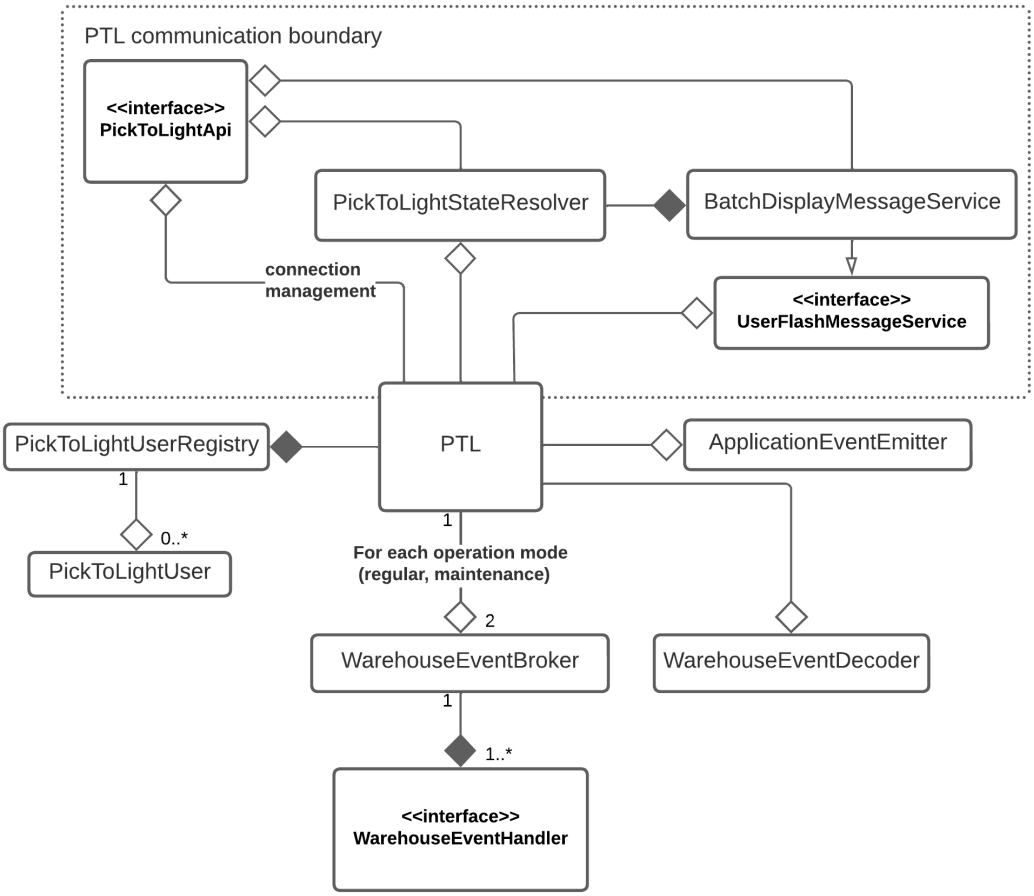
6.4.2 PTL fasáda

Hlavním vstupním a řídícím bodem aplikace je pak třída *PTL*, která spojuje jednotlivé výše zmíněné komponenty a spravuje seznam aktivních uživatelů (skrze pomocnou třídu *PickToLightUserRegistry*).

PTL třída volí sadu aktivních event handlerů na základě aktuálního operačního módu (vychystávání vs. údržba), volá rezoluci stavu *PTL* a spravuje spojení s *PTL* systémem skrze `PickToLightAPI` (6.3). Skrze `PickToLightAPI` také naslouchá na příchozí zprávy z *PTL* systému, které dále pomocí `WarehouseEventDecoder` převede na výše popsané instance *WarehouseEvent* a ty pak pomocí `WarehouseEventHandler` zpracuje. V případě chyby *PTL* systému se pokusí restartovat spojení a obnovit stav nebo aplikaci ukončí, pokud spojení nelze opravit.

Dále třída *PTL* zavádí mechanismus zajišťující pořadí vykonávaných akcí jedním uživatelem – jelikož jsou operace jako komunikace s *WMS* asynchronní, může dojít k situaci, kdy uživatel provede akci, která se ale dokončí až v momentě, kdy další akce uživatele již proběhla, čímž se uživatel dostane do nekonzistentního stavu. Tento problém je možné řešit například pomocí transakcí, které ale v případě distribuovaného systému (*PTL* aplikace + *WMS API*) nejsou nativně k dispozici. Zvoleno bylo proto jednodušší řešení – synchronizace na úrovni jednotlivých uživatelů pomocí zámků. Zamykání je tedy v aplikaci použito nezávisle na dvou místech – při komunikaci s jednotlivými *PTL* kontroléry a při volání event handlerů na úrovni uživatele, což se i při produkčním testování potvrdilo jako vhodné řešení s prakticky nulovým dopadem na výkon.

Celá architektura modulu je zobrazena na diagramu 6.9.



Obrázek 6.9 Diagram tříd business logiky řídící aplikace

6.5 Komunikace s WMS

V této sekci jsou popsány specifikace 3 různých API, které je třeba doimplementovat do *WMS* systému pro zajištění funkcionality *PTL* aplikace. Základní obecná designová rozhodnutí a technologie použité pro referenční implementaci byly popsány dříve v kapitole 6.1.0.0.3.

Tyto API jsou:

- Configuration API – pro získání konfigurace modulů a lokací systému,
- Picking API – pro manipulaci se skladovými zásobami a objednávkami,
- Maintenance API (volitelné) – pro zajištění funkcionality údržby systému.

Pro podporu procesu vychystávání objednávek je nutné implementovat Configuration API a Picking API. Maintenance API není pro základní funkčnost systému vyžadované, nicméně pro splnění všech požadavku (3) je potřeba.

PTL aplikace neudržuje žádná data o stavu skladu a informace o aktuální konfiguraci jsou dostupná jen v RAM v rámci jednoho běhu. Celá vrstva persistence je implementována v *WMS* systému. Toto řešení bylo zvoleno pragmaticky z následujících důvodů:

- WMS již obsahuje vlastní vrstvu persistence, není třeba zavádět další technologii,

- síťová komunikace v rámci skladu by měla být stabilní a rychlá (jinak by omezovala i další procesy WMS),
- vlastníkem dat o skladových zásobách je *WMS* systém. V případě udržování stavu i na straně *PTL* by bylo třeba řešit synchronizaci dat.

Pokud by některý z výše uvedených bodů neplatil, je možné doimplementovat vrstvu persistence jako separátní aplikaci, která bude implementovat specifikovaná API a teprve následně komunikovat s *WMS* systémem. I proto níže uvedená specifikace API nepokrývá všechny implementační detaily a je možné je upravit podle konkrétního řešení.

Veškeré API operace musí při úspěchu vracet HTTP status kód 200 OK a v případě nepovedené autorizace HTTP 401 Unauthorized. V případě, že operace selže, musí implementace API vracet HTTP status kód z rozsahu 4xx (např. 400 Bad Request) – ostatní detaily ohledně reprezentace chybových stavů jsou na konkrétní implementaci. Zavedení identifikátorů chybových stavů bylo zvažováno, nicméně možnost *PTL* systému informovat pracovníka o konkrétních chybách je omezená na malé batch displeje a řídící aplikace by navíc neměla uživatele do nevalidního stavu pustit. Dodatečné informace jsou tedy důležité primárně z hlediska monitorování, ladění a debugování.

Popis schématu jednotlivých API operací využívaný v této kapitole vychází ze syntaxe jazyka TypeScript, díky čemuž bylo možné popsat přenášené datové struktury včetně typové informace v kompaktním formátu. Standardně využívané formáty JSON-schema či Swagger jsou pro ukázky v textu příliš dlouhé. Všechny definované prvky jsou defaultně povinné a nenulové, pokud není uvedeno jinak (`undefined`, `null`).

6.5.1 Configuration API

Configuration API slouží pro získání konfigurace kontrolérů, modulů a lokací systému. Bez funkčního Configuration API není možné řídící aplikaci nastartovat – i v případě, že by byly IP adresy kontrolérů známé, nebylo by z *PTL* systému možné získat identifikátory lokací, ke kterým jsou světelné moduly přiřazené.

Při startu řídící aplikace není přítomný žádný uživatel systému, v při komunikaci s referenční implementací aplikace tedy při získávání konfigurace vystupuje pod servisním uživatelem. Autentizace je zajištěna pomocí HTTP Basic Authentication [28].

API definuje pouze jeden endpoint, jehož formát dotazu a odpovědi je popsán na obrázku 1. Vstupem je pouze identifikátor *PTL* systému, který je v referenčním řešení identický s identifikátorem skladu. Výstupem je pak kompletní konfigurace systému ve stromové struktuře, kdy pod jednotlivými kontroléry jsou uvedeny všechny přiřazené lokace a moduly. Řídící aplikace si z toho odvozenou strukturu uloží do paměti a má ji k dispozici po celý běh aplikace.

6.5.2 Picking API

Picking API obstarává veškerou komunikaci s WMS při procesu vychystávání objednávek a definuje proto následující operace, které vycházejí z popisu procesu vychystávání objednávek v kapitole 5.1, resp. jeho detailnějšího popisu v diagramu 6.8:

- přihlášení pracovníka,
- získání informací o produktech potřebných pro vybranou krabici,
- vychystání zboží z lokace do přepravky,
- přeskočení lokace kvůli problémům (např. chybějící zboží, problém s čárovým kódem produktu, …),

```
/**
 * GET /api/v1/pick-to-light/configuration/{ptlSystemId}
 * HTTP Basic Auth
 * ---------
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK
 * Response body
 */
interface SuccessfulConfigurationResponsePayload {
    data: {
        controller_configurations: Array<{
            device_ip: string;
            device_port: number;
            timeout: number;
            devices: Array<
            {
                device_id: number;
                device_type: 'pickToLightLightScanner';
            }
                |
            {
                device_id: number;
                device_type: 'pickToLightLightModule';
                location_bin: string;
                picking_area_name: string;
            }
                |
            {
                device_id: number;
                device_type: 'pickToLightLightBatchDisplay';
                picking_area_name: string;
            }>;
        }>;
        picking_areas: Array<string>;
    };
}
```

Výpis kódu 1 Configuration API – operace getConfiguration

- označení přepravky jako plné.

Veškeré akce Picking API jsou narozdíl od Configuration API prováděny pod uživatelským účtem pracovníka *PTL* systému.

6.5.2.0.1 Operace login zajišťuje přihlášení uživatele do systému. Pro autentizaci uživatele (a následnou autorizaci akce) je nejprve třeba vyměnit identifikátor login karty za aplikační klíč.

Toho je docíleno provoláním operace `login` popsané na obrázku 2. V dotazu vystupuje vedle identifikátoru login karty ještě `application_secret`, což je fixní klíč vydaný pro *PTL* aplikaci, který je součástí konfigurace systému a díky kterému je značně složitější zneužít login kartu bez fyzického přístupu ke skladovému hardware – pokud by útočník získal login kartu, i pokud by byl WMS otevřený do internetu, musel by akce provádět skrze konkrétní aplikaci nebo `application_secret` uhodnout.

Výstupem operace je klíč, který se přikládá ke všem dalším akcím skrze HTTP hlavičku `Api-Key`. Specifikace nedefinuje délku platnosti klíče, pro pohodlnou práci by však délka měla odpovídat alespoň standardní pracovní době pracovníka.

```
/**
 * POST /api/v1/pick-to-light/login
 * --------------------------------
 *
 * Request body
 */
interface LoginRequestPayload {
    login_key: string;
    app_secret: string;
}

/**
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK
 * Response body
 */
interface SuccessfulLoginResponsePayload {
    data: {
        user_id: number;
        api_key: string;
    };
}
```

Výpis kódu 2 Picking API – operace login

6.5.2.0.2 Operace retrievePicks získá kvantity a lokace produktů k vychystání pro vybranou krabici. Vstupem je identifikátor přepravky přečtený pomocí skeneru pracovníka a identifikátor *PTL* sekce, ve které operuje. Výstupem je pole struktur popisující lokaci s modulem, potřebnou kvantitu a detaily o produktu. Důležitou informací o produktu je seznam čárových kódů, pod kterými je produkt evidován ve WMS. Jelikož *PTL* lokace nemohou obsahovat více produktů identifikovaných různými *SKU*, je zbytek atributů je pouze informační.

```
/**
 * POST /api/v1/pick-to-light/picking/{ptlSystemId}/retrieve-picks
 * Api-key: {apiKeyFromLoginOperation}
 * ---------
 * Request body
 */
interface RetrievePicksRequestPayload {
   picking_container_bin: string;
   picking_area_name: string;
}

/**
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK
 * Response body
*/
interface SuccessfulRetrievePicksResponsePayload {
    data: Array<{
        location_bin: string;
        quantity: number;
        light_module_identifier: number;
        product_data: {
            id: number;
            sku: string;
            name: string;
            barcodes: Array<string>;
        }
    }>;
}
```

Výpis kódu 3 Picking API – operace retrievePicks

6.5.2.0.3 Operace pickProduct reprezentuje vychystání určené kvantity zboží z lokace do přepravky. Operace v systému poníží počet kusů zboží na lokaci o požadovanou kvantitu a přičte je k počtu kusů v krabici. Vstupem je identifikátor přepravky, do které je zboží umístěno, identifikátor světelného modulu přiřazeného k lokaci, ze které je zboží vychystáno a kvantita vychystaného zboží. Výstupem pak HTTP status kód 200 v případě úspěchu a 4xx v případě selhání.

Vzhledem k omezeným možnostem *PTL* systému v komunikaci s uživatelem si musí operace poradit bez selhání i s nestandardními situacemi, zejména případem, kdy je vychystána menší než požadovaná kvantita. Referenční implementace této operace se proto v případě vychystání menší než požadované kvantity pokusí namísto chyby zbytek požadované kvantity přealokovat na jinou skladovou lokaci nebo, pokud to není možné, krabici dočasně odstaví – z pohledu *PTL* operátora ale operace proběhne v pořádku.

Ve velmi zřídka se vyskytujících případech, kdy je vychystána větší než požadovaná kvantita, například z důvodu souběhu operace vychystání a systémového přesunu zboží na lokaci, je ve *WMS* systému zaznamenán problém a přepravka je poslána na kontrolu – opět ale, aby se zabránilo řešení nestandardních situací bez kontextu, proběhne vše z pohledu *PTL* operátora bez problému. Veškeré takové problémy jsou ve *WMS* systému zaznamenány standardním způsobem a mohou tak být řešeny v rámci standardních procesů.

```
/**
 * POST /api/v1/pick-to-light/picking/{ptlSystemId}/pick
 * Api-key: {apiKeyFromLoginOperation}
 * ---------
 * Request body
 */
interface PickRequestPayload {
   picking_container_bin: string;
   light_module_identifier: number;
   quantity: number;
}

/**
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK
 * (Empty body)
*/
```

Výpis kódu 4 Picking API – operace pickProduct

6.5.2.0.4 Operace skipPick umožňuje přeskočit vychystání zboží z lokace. Vstupem operace je identifikátor přepravky, do které má být zboží umístěno, identifikátor lokace, ke které je problém vztažen a kód problému.

Referenční implementace této operace definuje problémové kódy pro situaci, kdy na lokaci z nějakého důvodu není možné naskenovat správný čárový kód produktu, ať už z důvodu, že je čárový kód nečitelný, nebo z důvodu, že je na lokaci špatný produkt, a také pro situaci, kdy je přepravka plná a není možné do ní umístit další zboží.

Tato operace je volaná při akcích *Location empty flow* a *Barcode issue flow*, viz. 6.4.

6.5.2.0.5 Operace fullTote umožňuje ukončit vychystávání zboží do konkrétní přepravky, pokud se do ní již nevejde další zboží, viz. 6.4. Vstupem operace je identifikátor přepravky. Operaci je třeba volat před samotným vychystáním konkrétního produktu. Co se s objednávkou stane je závisí na *WMS* – např. může být přepravka odstavena, dokud nebude zbytek zboží vychystán do jiné přepravky, a následně bude zboží sloučeno v průběhu balení.

```
/**
 * POST /api/v1/pick-to-light/picking/{ptlSystemId}/skip-pick
 * Api-key: {apiKeyFromLoginOperation}
 * ---------
 * Request body
 */
interface SkipPickRequestPayload {
   picking_container_bin: string;
   pick_to_light_location_bin: string;
   reason: 'barcode_issue' | 'empty_location';
}

/**
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK
 * (Empty body)
*/
```

Výpis kódu 5 Picking API – operace skipPick

6.5.3 Maintenance API

Maintenance API je nepovinnou komponentou pro podporu údržby *PTL* systému. Primární funkcí je přiřazování (a tedy přidávání nebo náhrada) světelných modulů k lokacím. Za tímto účelem API definuje dvě operace: `configureLightModule` a `verifyPermission`.

PTL systém neumožňuje objevení identifikátoru neznámého světelného modulu, je proto třeba odeslat příkaz pro změny identifikátoru (6.3.3.0.7) a identifikátor přepsat.

6.5.3.0.1 Operace configureLightModule se pak stará o přepsání konfigurace, tedy přiřazení světelného modulu k lokace ve *WMS* systému. Vstupem je identifikátor modulu a identifikátor lokace.

6.5.3.0.2 Operace verifyPermission slouží k ověření oprávnění uživatele k provedení změny konfigurace. Tuto operaci bylo třeba definovat proto, že konfigurace v *PTL* systému a WMS jsou oddělené transakce a tedy v případě chyby není jednoduché vrátit předchozí stav. V implementaci *PTL* aplikace je nejprve provedena změna v *PTL* systému, aby se zabránilo přepsání dat v případě, že při zavolání příkazu konfigurace nebyl zvolen žádný modul a až poté je změna odeslána do *WMS* systému. Při volání v tomto pořadí ale hrozí, že se o změnu pokusí uživatel bez oprávnění zasahovat do konfigurace, čímž se *PTL* systém dostane do nekonzistentního stavu. Proto je kontrola oprávnění vystavena jako separátní endpoint, který je volán před provedením změny konfigurace.

Celé flow je tedy následující:

1. Operátor aktivuje mód údržby.
2. Operátor zmáčkne a drží tlačítko modulu, který chce nakonfigurovat.
3. Operátor naskenuje čárový kód lokace, ke které chce přiřadit modul.
4. *PTL* aplikace zavolá `verifyPermission`. Pokud uživatel není oprávněn, zobrazí se chybová hláška na batch displeji a akce končí.

```
/**
 * POST /api/v1/pick-to-light/picking/{ptlSystemId}/full-tote
 * Api-key: {apiKeyFromLoginOperation}
 * ---------
 * Request body
 */
interface PickRequestPayload {
   picking_container_bin: string;
}

/**
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK
 * (Empty body)
*/
```

Výpis kódu 6 Picking API – operace fullTote

5. *PTL* aplikace odešle příkaz Změny identifikátoru modulu.
6. *PTL* aplikace zavolá `configureLightModule`.

6.5.4 Klienti

Klienti pro výše popsaná API jsou v rámci *PTL* aplikace implementovány ve složce `backend-api`. Pro komunikaci je využíváno `Fetch` API [29], resp. jeho implementace `node-fetch`.

```
/**
 * PUT /api/v1/pick-to-light/maintenance/{ptlSystemId}/location/{locationBin}
 * Api-key: {apiKeyFromLoginOperation}
 * ---------
 * Request body
 */
interface ConfigureLightModuleRequestPayload {
   module_identifier: number;
   controller_ip: string;
}

/**
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK
*/
interface SuccessfulConfigureLightModuleResponsePayload {
    data: Array<{
        location_bin: string;
        picking_area_name: string;
        device_id: number;
        device_type: string;
    }>;
}
```

Výpis kódu 7 Maintenance API – operace configureLightModule

```
/**
 * POST /api/v1/pick-to-light/maintenance/{ptlSystemId}/verify-permissions
 * Api-key: {apiKeyFromLoginOperation}
 * ---------
 * Request body (empty)
 *
 * Responses
 * ---------
 * 4xx
 *
 * 200 OK (empty body)
*/
```

Výpis kódu 8 Maintenance API – operace verifyPermission

Kapitola 7
Testování

Následující kapitola popisuje průběh a výsledky testování implementované aplikace. Použité testovací metody lze rozdělit do tří kategorií: **automatizované**, **manuální** a **uživatelské**, přičemž každá z kategorií je cílena na odhalení jiných druhů problémů.

7.1 Automatizované testy

Cílem automatizovaných testů je ověření správnosti a funkčnosti aplikace pomocí testovacích nástrojů a skriptů, které konzistentně simulují užívání aplikace nebo jejích podčástí. Automatizované testy jsou v porovnání s manuálním testování řádově rychlejší a značně méně náchylné na lidskou chybu. Zásadní výhodou je jejich opakovatelnost – správně napsaný test by měl vždy skončit stejným výsledkem, pokud se nezmění testovaná část aplikace. Oproti manuálnímu testování však automatizované testy vyžadují značnou počáteční investici do jejich vytvoření, a nejsou explorativní – při manuálním testování si tester může všimnout chyby, která nebyla součástí testovacího scénáře, případně náhodný odklon od scénáře může odhalit chybu, která by jinak zůstala skryta [30].

Ačkoliv nebyly dodržovány praktiky *programování řízeného testy* (*test-drive development*, TDD), automatizované testy byly psány v průběhu vývoje aplikace a pouštěny před každým integrováním změn do hlavní vývojové větve (viz. 2). Pro psaní testů byla použita knihovna *Jest* [31].

7.1.1 Komunikace s *PTL*

Při automatizovaném testování modulu komunikace s *PTL* byl kladen důraz na pokrytí parsování vstupních a formátování výstupních zpráv, čehož bylo dosaženo **jednotkovými testy** (unit testy).

Dále bylo jednotkovými testy pokryto rozhraní stavové třída `PickToLightConnection`, při kterých je socketové spojení simulováno (technikou *mocking*). Testovací scénáře pokrývají především proces výměny zpráv z obou systémů, obzvláště na situace náchylné souběhu (*race condition*), a schopnost aplikace reagovat na chyby spojení.

7.1.2 Komunikace s WMS

Otestování komunikace s *WMS* skrze *API* proběhlo na úrovni rozhraní jednotlivých klientů. Testy neprovádí reálné volání *API*, ale pouze simulují chování `Fetch API`. Zaznamenané vý-

sledné *HTTP* volání klienta je následně porovnáváno s předem zaznamenanými daty (*snapshot testing*) [32].

7.1.3 Business logika

Testování hlavní business logiky aplikace probíhalo na úrovni **integračních testů** (integration tests) jednotlivých `Event handler` tříd (6.4.1). Důraz byl kladen na korektní přechody mezi stavy reprezentované třídou `PickToLightUser`. Jak už bylo popsáno v části zabývající se architekturou aplikace (6.2), potřebná komunikace s *PTL* je odvozována právě z aktuálního stavu uživatele a není tedy třeba se jí zabývat na úrovni `Event handler` tříd. Volání *WMS API* jsou simulována technikou *mocking*.

7.2 Manuální testy

Manuální testování probíhalo na lokálně dostupné instalaci *PTL* systému a testovací instanci *WMS*. K dispozici bylo 12 světelných *PTL* modulů emulující 12 skladových lokací, jeden *PTL kontrolér* (4.1.1), jeden *junction box* (4.3) a dva připojené skenery. Lokace byly rozděleny do dvou logických sekcí, aby bylo možné testovat případy, kdy dva uživatelé pracují současně na různých sekcích.

Testovaná aplikace komunikovala s vyhrazenou instancí *WMS* určenou pro testování, kde byla uložena i konfigurace *PTL* zařízení lokální instalace. Pro každý ze scénářů bylo třeba předem připravit vhodná data. Konkrétní pravidelně prováděné testovací scénáře jsou součástí přílohy této práce (B).

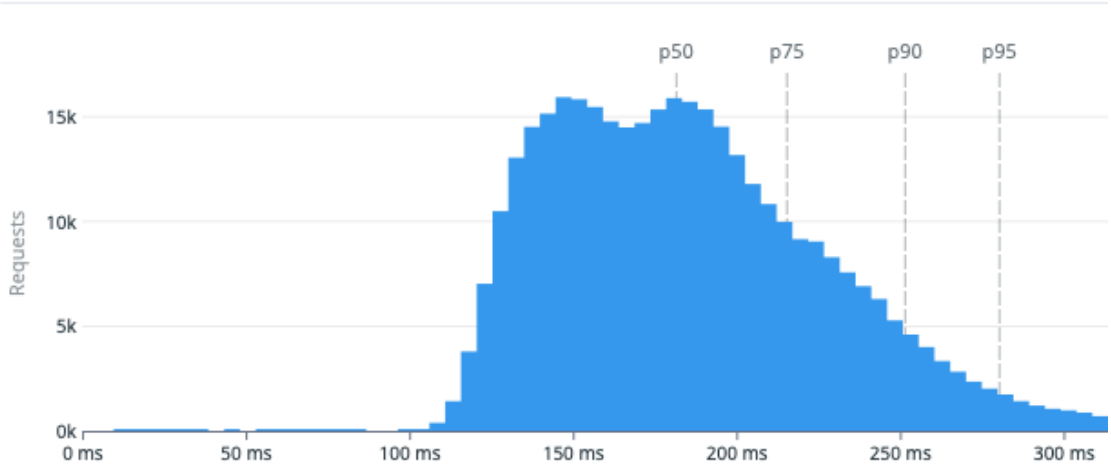
7.3 Uživatelské testy

Uživatelské testování probíhalo v pozdějších fázích vývoje v testovacích skladech a bylo zaměřeno na **identifikování problémů v navrženém procesu a uživatelském rozhraní**. Subjekty pozorování byli zaměstnanci skladu, z nichž někteří měli předchozí zkušenost s procesy z vychystávací zóny. Testování bylo prováděno prezenčně, kdy byl subjekt svým zaškoleným nařízeným požádán o vykonání několika úkolů. Autor práce vystupoval v roli pozorovatele a následně proces s vybranými uživateli konzultoval za účelem sesbírání zpětné vazby.

V rámci uživatelského testování byla například testována varianta procesu, kdy po naskenování čárového kódu produktu během vychystávání z lokace bylo nutné naskenovat i identifikátor krabice, do které je produkt vkládán. Na základě výsledků ale bylo zjištěno, že většina pracovníků krabici nosí neustále s sebou a je tedy zbytečné do procesu přidávat krok navíc.

Kromě intuitivnosti procesu byl při uživatelském testování kladen důraz na pocitovou **odezvu systému**. Ačkoliv byla architektura systému navržena tak, aby se uživatelé vzájemně blokovali co nejméně (6.2), bylo třeba tento předpoklad ověřit v praxi.

Největším potenciálním problémem se na základě sesbíraných dat (jak z testování, tak z monitoringu provozu) ukázaly být volání *WMS API*, komunikace s *PTL* systémem po lokální síti byla stabilní. Konkrétně se například ukázalo, že operace *pickProduct* (6.5.2.0.3) je v 95 % případů dokončena pod 300ms (graf 7.1)), ale v jednotkách případů trvá i několik sekund. Problém s konkrétním typem objednávek způsobujících tuto prodlevu byl identifikován a opraven optimalizací databázových dotazů.



Obrázek 7.1 Zaznamenaná latence z reálného provozu operace *pickProduct*

Kapitola 8
Sestavení a nasazení

Výsledná aplikace je distribuována formou Docker [33] obrazu obsahující celé prostředí, pro spuštění tak stačí dodefinovat proměnné prostředí a spustit kontejner. Kompilace zdrojového kódu je součástí procesu sestavení Docker obrazu, který probíhá automaticky při změně v rámci Gitlab CI [34] pipeline. Obraz každé sestavené verze je nahrán do rejstříku (konkrétně do veřejně nepřístupného AWS ECR) pod unikátním tagem.

V případě testovacích skladů byla aplikace nasazena v každém ze skladů na server dedikovaný pro aplikace komunikující se skladovým hardwarem právě jako Docker kontejner [1].

Logy z aplikačního kontejneru jsou sbírány pomocí Datadog Agent s rozšířením `journald`, který je přeposílá do cloudové platformy Datadog [35]. Chyby aplikace jsou zaznamenávány pomocí služby Sentry [36]. Nasazení aplikace probíhá změnou tagu Docker obrazu v definici konfigurace služby spouštěné pomocí nástroje `systemd` [37]. Změna používaného tagu a restart služby byl integrován do manuálně spustitelné Gitlab CI pipeline.

V rámci prvotního nasazení bylo také třeba nakonfigurovat všechny kontroléry (konfigurace sítě, času) skrze `telnet` rozhraní (4.1.1), dále zařízení typu *barcode reader interface* pomocí speciálního skriptu využívající implementovanou komunikační vrstvu a zadefinovat hardwarovou konfiguraci *PTL* systému ve *WMS* pro `Configuration API` (1). Následně bylo třeba přemapovat jednotlivé moduly na lokace pomocí operací z procesu údržby a několikrát celý systém zkontrolovat.

Aplikace v aktuální podobě nepodporuje bezvýpadkové nasazení. V případě potřeby nasazení nové verze je třeba původní aplikaci zastavit, přičemž je stav aplikace je ztracen. Jelikož *PTL* systém podporuje pouze jedno aktivní připojení a při navázání nového je předchozí přerušeno, opravdové bezvýpadkové nasazení je prakticky nemožné, ale funkcionalitu by šlo velmi dobře simulovat tím, že by původní verze aplikace předala svůj stav (například nahráním na lokální úložiště) nové verzi, která by po připojení obnovila *PTL* systém do správné konfigurace. V reálném prostředí pro to ale nebyla potřeba, jelikož se řídící aplikace po nasazení nebude příliš měnit a v případě potřeby lze v provozu najít jednotky minut, kdy lze systém pozastavit.

[1] Samotná konfigurace infrastruktury **nebyla** provedena v rámci této práce.

Kapitola 9
Závěr

Cílem této diplomové práce bylo navrhnout efektivní a intuitivní proces vychystávání zboží pomocí technologie *PTL*, který bude odpovídat potřebám skladů zaměřených na *e-commerce fulfillment*, a následně implementovat řídící aplikaci *PTL* systému, která bude tento proces realizovat pro konkrétní hardwarovou instalaci ve skladech zpřístupněných pro testování.

V práci byla nejprve popsána doména skladování a problematika vychystávání zboží u poskytovatelů *e-commerce fulfillment* služeb. Obecná problematika byla následně přiblížena z pohledu konkrétních procesů vychystávání zavedených v testovacích skladech pro získání lepšího kontextu a pochopení potřeb a požadavků na navrhovaný proces. Vypozorované požadavky byly konzultovány se zástupci produktového týmu a operativy, na základě čehož byl vytvořen finální seznam požadavků na systém. Dále byla provedena analýza konkrétní realizace technologie *PTL* instalované v testovacích skladech a rešerše existujících řešení pro komunikaci s *PTL* systémem.

Na základě výstupů těchto analýz a požadavků byl navržen proces vychystávání. Jelikož nebyla nalezena žádná implementace komunikační vrstvy pro *PTL* systém odpovídající požadavkům, bylo nutné TCP/IP komunikaci s *PTL* systémem implementovat v rámci práce. Pro realizaci aplikace byl na základě zjištěných požadavků zvolen jazyk *TypeScript* běžící na platformě *NodeJS*. Aplikace navíc umožňuje provádět základní operace týkající se údržby systému, jako je například výměna rozbitých *PTL* modulů na lokacích. Po mnoha iteracích vývojového cyklu byl vytvořen funkční prototyp aplikace, který byl následně nasazen do testovacích skladů a uživatelsky testován. Na základě zpětné vazby uživatelů byly provedeny úpravy a vylepšení aplikace, které vedly k finálnímu návrhu popsaném v této práci. Cíle práce se podařilo úspěšně realizovat – všechny požadavky na aplikaci byly splněny a **aplikace byla otestována v reálném provozu** a **nasazena do produkce**.

Příloha A

Seznam zkratek

API	Application Programming Interface
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
TELNET	Teletype Network
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
OSI	Open Systems Interconnection
PTL	Pick to Light
TCP	Transmission Control Protocol
SDK	Software Development Kit
WMS	Warehouse Management System
RPC	Remote Procedure Call
REST	Representational state transfer
UX	User Experience

Příloha B

Příloha – Testovací scénáře

Scénář 1: **Přihlášení pracovníka do sekce**
Popis: Scénář má za cíl otestovat přihlašování uživatele do systému a přepínání sekcí.

Testovací případ: **Přihlášení jednoho uživatele**

1. Uživatel připojí skener. PTL zobrazuje na batch displejích všech sekcích „SCAN PK AREA".
2. Uživatel naskenuje identifikátor sekce. Na batch displeji dané sekce se krátce zobrazí "AREA SET" a následně "SCAN LOGIN". Ostatní sekce zůstanou nezměněné
3. Uživatel naskenuje svou login kartu. Na batch displeji se krátce ukáže "LOGGED IN" a následně "SCAN TOTE".

Testovací případ: **Uživatel s nevalidní login kartou je ignorován**

1. Uživatel připojí skener. PTL zobrazuje na batch displejích všech sekcích „SCAN PK AREA".
2. Uživatel naskenuje identifikátor sekce. Na batch displeji dané sekce se krátce zobrazí "AREA SET" a následně "SCAN LOGIN". Ostatní sekce zůstanou nezměněné
3. Uživatel naskenuje nevalidní login kartu. Na batch displeji se krátce ukáže "WRONG LOGIN" a následně znovu "SCAN LOGIN".

Testovací případ 2: **Přihlášení jiného uživatele do stejné sekce**

1. Uživatel se přihlásí do sekce jako v testovacím případě **Přihlášení jednoho uživatele**
2. Další uživatel se připojí do systému se svým skenerem (připojeným k jinému barcode reader interface).
3. Nový uživatel naskenuje identifikátor stejné sekce, kde již operuje původní uživatel. Sekce je resetována a dostane se do stavu „SCAN LOGIN".
4. Původní uživatel se pokusí naskenovat svou login kartu a identifikátor přepravky. Jeho skeny jsou ignorovány.
5. Nový uživatel naskenuje svou login kartu. Je přihlášen. Sekce je ve stavu „SCAN TOTE".

Testovací případ 3: **Změna sekce již přihlášeného uživatele**

1. Uživatel se přihlásí do sekce jako v testovacím případě **Přihlášení jednoho uživatele**
2. Uživatel naskenuje identifikátor jiné sekce. Původní sekce je resetována do stavu „SCAN PK AREA". Nová sekce je ve stavu „SCAN LOGIN".
3. Naskenováním login karty dojde k přihlášení do nové sekce, původní sekce zůstane netknutá.

Testovací případ 4: **Přihlášení dvou uživatelů najednou**

1. Uživatel 1 se přihlásí do sekce jako v testovacím případě **Přihlášení jednoho uživatele**
2. Uživatel 2 se přihlásí do sekce jako v testovacím případě **Přihlášení jednoho uživatele**

Poznámky: Kroky 1 a 2 lze libovolně prokládat bez vlivu na výsledek.

Scénář 2: **Odhlášení neaktivního uživatele**
Popis: Scénář má za cíl otestovat odhlášení uživatele v případě neaktivity, a to v libovolném stavu. Sekce, na které uživatel operoval, se dostane zpět do stavu „SCAN PK AREA“.
Pro účely pohodlnějšího testování je možné snížit čas čekání na odhlášení.

Testovací případ: **Odhlášení po přihlášení**

1. Uživatel se přihlásí do sekce, sekce ukazuje „SCAN TOTE”.
2. Po X minutách neaktivity ze strany uživatele se sekce vrátí do stavu „SCAN PK AREA”. Sekce nereaguje na další skeny uživatele (Login karty, přepravky).
3. Do sekce se lze znovu přihlásit.

Scénář 3: **Vychystání objednávky**
Popis: Scénář má za cíl otestovat průběh vychystávání objednávky systémem PTL.
Práci na konkrétní objednávce by mělo jít kdykoliv přerušit a znovu navázat. Uživatelé jednotlivých sekcí by se neměli nijak ovlivňovat.

Testovací případ: **Kompletní vychystání objednávky**
Předpoklady: Ve WMS je objednávka kompletně alokovaná do lokací v jedné PTL sekci. Produkty potřebné pro objednávku nemají známé čárové kódy.

1. Uživatel se přihlásí do sekce, sekce ukazuje „SCAN TOTE”.
2. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí moduly na požadovaných lokacích. Moduly ukazují správné kvantity potřebné k vychystání
3. Uživatel vychystá v libovolném pořadí zboží ze všech aktivních lokací. Zboží je ve WMS na lokacích poníženo o vychystanou kvantitu a systémově přesunuto do přepravky. Moduly zhasnou. Systém se vrátí do stavu „SCAN TOTE“
4. Uživatel naskenuje znovu stejnou přepravku. Systém krátce ukáže „NO PICKS“ a vrátí se do stavu „SCAN TOTE“. Žádné moduly se nerozsvítí.

Testovací případ: **Vychystání objednávky s několika produkty opatřenými čárovým kódem**
Předpoklady: Ve WMS je objednávka kompletně alokovaná do lokací v jedné PTL sekci. Alespoň dva produkty mají přiřazené čárové kódy.

1. Uživatel se přihlásí do sekce, sekce ukazuje „SCAN TOTE”.
2. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí moduly na požadovaných lokacích. Moduly na lokacích obsahující produkt s čárovým kódem ukazují zprávu „BC“, značící, že na lokaci je třeba naskenovat čárový kód produktu.
3. Uživatel stiskne tlačítko na modulu se zprávou „BC“. Nic se nestane, modul stále svítí.
4. Pro každý z produktů v libovolném pořadí:
    a. Uživatel naskenuje čárový kód produktu. Modul na lokaci se aktivuje a začne ukazovat kvantitu k vychystání.
    b. Uživatel vychystá zboží z lokace. Zboží je ve WMS na lokacích poníženo o vychystanou kvantitu a systémově přesunuto do přepravky. Modul zhasne.
5. Po vychystání všech lokací systém krátce ukáže „PC COMPLETED“ a vrátí se do stavu „SCAN TOTE“
6. Uživatel naskenuje znovu stejnou přepravku. Systém krátce ukáže „NO PICKS“ a vrátí se do stavu „SCAN TOTE“. Žádné moduly se nerozsvítí.

Testovací případ: **Vychystání objednávky na dvakrát**
Předpoklady: Ve WMS je objednávka kompletně alokovaná do lokací v jedné PTL sekci. Obsahuje právě dva různé produkty bez známých čárových kódů.

1. Uživatel se přihlásí do sekce, sekce ukazuje „SCAN TOTE".
2. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí moduly na požadovaných lokacích.
3. Uživatel vychystá libovolný ze 2 produktů. Lokace zhasne. Druhá zůstává svítit.
4. Uživatel naskenuje svou login kartu. Tím systém vrátí do stavu „SCAN TOTE".
5. Uživatel znovu naskenuje přepravku. Rozsvítí se pouze zbývající lokace k vychystání.
6. Po vychystání všech lokací systém krátce ukáže „PC COMPLETED" a vrátí se do stavu „SCAN TOTE"
7. Uživatel naskenuje znovu stejnou přepravku. Systém krátce ukáže „NO PICKS" a vrátí se do stavu „SCAN TOTE". Žádné moduly se nerozsvítí.

Testovací případ: **Vychystání objednávky, která potřebuje zboží ze dvou různých sekcích**
Předpoklady: Ve WMS je objednávka kompletně alokovaná do lokací ve dvou různých sekcích.

1. Uživatel se přihlásí do jedné ze sekcí, sekce ukazuje „SCAN TOTE".
2. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí lokace s produkty potřebnými do objednávky
3. Uživatel vychystá všechny produkty z lokace. Systém ukáže „PC COMPLETED" a vrátí se dostavu „SCAN TOTE"
4. Uživatel naskenuje znovu stejnou přepravku. Systém krátce ukáže „NO PICKS" a vrátí se do stavu „SCAN TOTE". Žádné moduly se nerozsvítí.
5. Uživatel se přepne na druhou sekci.
6. Uživatel se přihlásí přes svojí login kartu.
7. Uživatel naskenuje stejnou přepravku. Lokace z druhé sekce se rozsvítí.
8. Uživatel vychystá všechno potřebné zboží z této sekce.
9. Uživatel naskenuje znovu stejnou přepravku. Systém krátce ukáže „NO PICKS" a vrátí se do stavu „SCAN TOTE". Žádné moduly se nerozsvítí.

Testovací případ: **Vychystání objednávky při nedostatečné kvantitě zboží na lokaci**
Předpoklady: Ve WMS je objednávka o dvou produktech kompletně alokovaná do lokací v jedné sekci.

1. Uživatel se přihlásí do jedné ze sekcí, sekce ukazuje „SCAN TOTE".
2. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí lokace s produkty potřebnými do objednávky
3. Uživatel vychystá první produkt kompletně. Modul se deaktivuje.
4. Uživatel vychystá druhý produkt, ale před potvrzením tlačítkem na modulu upraví kvantitu na o jedna nižší. V systému je vygenerován záznam o problému na lokaci. Objednávka zůstane ve stavu k vychystání, jeden produkt chybí.
5. Uživatel naskenuje znovu stejnou přepravku. Systém krátce ukáže „NO PICKS" a vrátí se do stavu „SCAN TOTE". Žádné moduly se nerozsvítí.

Scénář 3: **Řešení problémů při vychystávání**
Testovací případ: **Zboží se nevejde do přepravky**
Předpoklady: Ve WMS je objednávka kompletně alokovaná do lokací v jedné PTL sekci.

1. Uživatel se přihlásí do sekce, sekce ukazuje „SCAN TOTE".
2. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí moduly na požadovaných lokacích.
3. Uživatel naskenuje speciální čárový kód „FULL TOTE". Všechny moduly zhasnou, lokace jsou přeskočeny, vychystávání objednávky pro danou přepravku je ukončeno.
4. Při opětovném naskenování přepravky se ukáže „NO PICKS".

Testovací případ: **Lokace požaduje sken čárového kódu produktu, ale je prázdná**
Předpoklady: Ve WMS je objednávka kompletně alokovaná do lokací v jedné PTL sekci. Jeden z produktů má přiřazený čárový kód.

5. Uživatel se přihlásí do sekce, sekce ukazuje „SCAN TOTE".
6. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí moduly na požadovaných lokacích.
7. Uživatel namísto vychystání produktu naskenuje speciální čárový kód „LOCATION EMPTY". Na batch displeji se zobrazí zpráva „SCAN LOCATIO"
8. Uživatel naskenuje čárový kód lokace. Modul na lokaci se vypne, lokace je přeskočena. Kontrola lokace je vytvořena ve WMS.
9. Uživatel vychystá zbytek objednávky.

Testovací případ: **Problém s čárovým kódem produktu**
Předpoklady: Ve WMS je objednávka kompletně alokovaná do lokací v jedné PTL sekci. Jeden z produktů má přiřazený čárový kód.

10. Uživatel se přihlásí do sekce, sekce ukazuje „SCAN TOTE".
11. Uživatel naskenuje přepravku přiřazenou dané objednávce. PTL systém rozsvítí moduly na požadovaných lokacích. Moduly na lokacích obsahující produkt s čárovým kódem ukazují zprávu „BC", značící, že na lokaci je třeba naskenovat čárový kód produktu.
12. Uživatel si vybere lokaci vyžadující sken čárového kódu produktu (na displeji modulu je zobrazeno „BC"). Namísto naskenování produktu uživatel skenuje speciální čárový kód „BARCODE ISSUE". Na batch displeji se zobrazí zpráva „SCAN LOCATIO"
13. Uživatel naskenuje čárový kód lokace s problémovým produktem. Modul na lokaci je vypnutý, lokace je přeskočena. Kontrola lokace je vytvořena ve WMS.
14. Uživatel vychystá zbytek objednávky.

Scénář 4: **Souběžná práce dvou uživatelů**
Uživatelé nezávisle na sobě testují případy popsané ve **scénáři 2**. Výsledek akcí by neměl být nijak ovlivněn souběžnou prací vice uživatelů najednou. Latence systému by měla být ekvivalentní situaci, kdy se systémem pracuje pouze jeden uživatel.

Bibliografie

1. KOSTER, René de; LE-DUC, Tho; ROODBERGEN, Kees Jan. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*. 2007, roč. 182, č. 2, s. 481–501. ISSN 03772217. Dostupné z DOI: 10.1016/j.ejor.2006.07.009.
2. *Oracle Netsuite Order Fulfillment* [online]. [cit. 2023-04-26]. Dostupné z: https://www.netsuite.com/portal/products/erp/warehouse-fulfillment/order-fulfillment.shtml.
3. *Shipstation* [online]. [cit. 2023-04-26]. Dostupné z: https://www.shipstation.com/.
4. MOFOKENG, Thabang Excellent. The impact of online shopping attributes on customer satisfaction and loyalty: Moderating effects of e-commerce experience. *Cogent Business & Management*. 2021, roč. 8, č. 1, s. 1968206. ISSN 2331-1975. Dostupné z DOI: 10.1080/23311975.2021.1968206.
5. TARN, J. Michael; RAZI, Muhammad A.; WEN, H. Joseph; PEREZ, Angel A. E-fulfillment: the strategy and operational requirements. *Logistics Information Management*. 2003, roč. 16, č. 5, s. 350–362. ISSN 0957-6053. Dostupné z DOI: 10.1108/09576050310499345.
6. RICHARDS, Gwynne. *Warehouse management: a complete guide to improving efficiency and minimizing costs in the modern warehouse*. Second edition. London: Kogan Page Limited, 2014. ISBN 978-0-7494-6934-4.
7. NIEMCZYK, Aleksander. Warehouse processes in enterprises. 2016, s. 73–86.
8. WOZNIAKOWSKI, Tomasz; JALOWIECKI, Piotr; ZMARZLOWSKI, Krzysztof. ERP systems and warehouse management by WMS. *Information systems in management*. 2018, roč. 7.
9. BATTINI, Daria; CALZAVARA, Martina; PERSONA, Alessandro; SGARBOSSA, Fabio. A comparative analysis of different paperless picking systems. 2015, roč. 115, č. 3, s. 483–503. ISSN 0263-5577. Dostupné z DOI: 10.1108/IMDS-10-2014-0314.
10. BONILLA-ENRIQUEZ, Gladys; CABALLERO-MORALES, Santiago-Omar. Search Algorithms on Logistic and Manufacturing Problems. In: *Search Algorithm - Essence of Optimization*. Ed. G. HARKUT, Dinesh. IntechOpen, 2023. ISBN 978-1-83969-086-0. Dostupné z DOI: 10.5772/intechopen.96554.
11. MULLER, Robert. *PIECE PICKING: WHICH METHOD IS BEST?* [online]. 2007. [cit. 2023-03-06]. Dostupné z: https://www.distributiongroup.com/articles/piecepickingwhichmethod.pdf.
12. *Sure Sort* [online]. [cit. 2023-02-07]. Dostupné z: https://www.bastiansolutions.com/solutions/technology/sortation/opex-sure-sort/.

13. *Locus Robotics* [online]. [cit. 2023-03-04]. Dostupné z: https://locusrobotics.com/.

14. RASMI, Seyyed Amir Babak; WANG, Yuan; CHARKHGARD, Hadi. Wave order picking under the mixed-shelves storage strategy: A solution method and advantages. *Computers & Operations Research*. 2022, roč. 137, s. 105556. ISSN 03050548. Dostupné z DOI: 10.1016/j.cor.2021.105556.

15. DANIELS, Richard L.; RUMMEL, Jeffrey L.; SCHANTZ, Robert. A model for warehouse order picking. *European Journal of Operational Research*. 1998, roč. 105, č. 1, s. 1–17. ISSN 03772217. Dostupné z DOI: 10.1016/S0377-2217(97)00043-X.

16. *Agile model* [online]. [cit. 2023-03-02]. Dostupné z: https://www.tutorialscampus.com/sdlc/agile-model.htm.

17. FOWLER, Martin. *Continuous Integration* [online]. 2006. [cit. 2023-03-12]. Dostupné z: https://martinfowler.com/articles/continuousIntegration.html.

18. *Git* [online]. [cit. 2023-03-01]. Dostupné z: https://git-scm.com/.

19. *MWU Series Installation User Manual v1.0*. Lightning Pick | Matthews Automation Solutions, [b.r.]. Dostupné v přiloženém médiu.

20. *L-PICK Controller Command Reference v4.1*. Lightning Pick | Matthews Automation Solutions, [b.r.]. Dostupné v přiloženém médiu.

21. *ProGlove Mark 2* [online]. [cit. 2023-02-07]. Dostupné z: https://proglove.com/products/hardware/mark-2/.

22. *Lightning Pick Management Console* [online]. [cit. 2023-02-02]. Dostupné z: https://lightningpick.com/lightning-pick-version-7-1-released/.

23. LUKKARINEN, Aleksi; MALMI, Lauri; HAARANEN, Lassi. Event-driven Programming in Programming Education: A Mapping Review. *ACM Transactions on Computing Education*. 2021, roč. 21, č. 1, s. 1–31. ISSN 1946-6226. Dostupné z DOI: 10.1145/3423956.

24. *TypeScript* [online]. [cit. 2023-02-05]. Dostupné z: https://www.typescriptlang.org/.

25. LORING, Matthew C.; MARRON, Mark; LEIJEN, Daan. Semantics of asynchronous JavaScript. In: *Proceedings of the 13th ACM SIGPLAN International Symposium on on Dynamic Languages*. Vancouver BC Canada: ACM, 2017, s. 51–62. ISBN 978-1-4503-5526-1. Dostupné z DOI: 10.1145/3133841.3133846.

26. *ESlint* [online]. [cit. 2023-03-04]. Dostupné z: https://eslint.org/.

27. FIELDING, Roy Thomas; TAYLOR, Richard N. *Architectural Styles and the Design of Network-Based Software Architectures*. University of California, Irvine, 2000. ISBN 0599871180. Dis. pr.

28. (IETF), Internet Engineering Task Force. *Hypertext Transfer Protocol (HTTP/1.1): Authentication* [online]. 2014. [cit. 2023-03-25]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7235.

29. *Fetch API* [online]. [cit. 2023-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

30. GRAHAM, Dorothy; FEWSTER, Mark. *Experiences of test automation: case studies of software test automation*. Addison-Wesley Professional, 2012.

31. *Jest* [online]. [cit. 2023-04-20]. Dostupné z: https://jestjs.io/.

32. *Snapshot Testing* [online]. [cit. 2023-04-20]. Dostupné z: https://circleci.com/blog/snapshot-testing-with-jest.

33. *Docker* [online]. [cit. 2023-03-04]. Dostupné z: https://www.docker.com/.

34. *Gitlab CI* [online]. [cit. 2023-04-20]. Dostupné z: https://docs.gitlab.com/ee/ci/.

35. *Datadog* [online]. [cit. 2023-05-01]. Dostupné z: `https://www.datadoghq.com`.
36. *Sentry* [online]. [cit. 2023-05-01]. Dostupné z: `https://sentry.io`.
37. *Systemd* [online]. [cit. 2023-05-01]. Dostupné z: `https://systemd.io`.

Obsah přiloženého média

`thesis.pdf` ..... text práce ve formátu PDF
`source` ..... Zdrojové kódy implementace