WMS pro správu elektronických součástek

Semestrální projekt

Autoři:

Bc. Lukáš Kvapil
Bc. Tomáš Szetei
Bc. Tomáš Simandl

Název předmětu: Projektování IS - EIT21E

Fakulta: Provozně ekonomická fakulta

Zajišťuje: Katedra informačního inženýrství (PEF)

Semestr: LS 2025/2026

Praha, 10. února 2026

Obsah

1 Formulace problému

1.1 Popis problémové domény

Při výrobě elektronických zařízení je nezbytné efektivně spravovat **BOM (Bill of Materials)** – seznam všech použitých součástek. Tradiční metody jako ruční zápis nebo Excel tabulky jsou náchylné k chybám, pomalé a neumožňují snadné sledování součástek napříč projekty.

Současná praxe při správě součástek zahrnuje:

- Manuální zápis Part Number do tabulek
- Obtížnou synchronizaci mezi skladem a projekty
- Riziko chyb při přepisování identifikátorů
- Časově náročné vyhledávání informací o součástkách
- Složitou evidenci skladových míst

1.2 Cíl systému

1.2.1 Hlavní cíl

Vytvořit desktopovou aplikaci **WMS pro správu elektronických součástek** (Warehouse Management System) pro automatizovanou správu BOM (Bill of Materials), která eliminuje manuální chyby, urychluje proces evidence součástek a poskytuje komplexní přehled o stavu skladu a přiřazení součástek k projektům.

1.2.2 Dílčí cíle

C1: Automatizace skenování a evidence

- Implementovat podporu pro Zebra čtečky čárových kódů
- Automaticky parsovat QR kódy a extrahovat strukturovaná data
- Detekovat duplicity a automaticky sčítat množství
- Ukládat kompletní historii skenování s časovými razítky
- Redukovat čas zpracování z 30+ sekund (manuální zápis) na < 1 sekundu

C2: Integrace s externími službami

- Připojit TME API pro automatické obohacení dat o kategorie
- Získávat popisy, technické parametry a ceny součástek
- Zajistit offline funkčnost při nedostupnosti API
- Implementovat fuzzy matching pro varianty výrobních kódů

C3: Správa projektů a organizace

- Umožnit vytváření a správu projektů
- Přiřazovat součástky k více projektům současně (M:N vztah)
- Zobrazovat přehled součástek pro každý projekt
- Sledovat spotřebu součástek na jednotlivých projektech

C4: Správa skladových míst

- Implementovat systém skladových lokací s unikátními kódy
- Přiřazovat součástky na konkrétní fyzická místa
- Generovat a tisknout štítky skladových míst pomocí ZPL
- Podporovat Zebra tiskárny pro tisk štítků 2×1 palec

C5: Export a reporting

- Exportovat BOM do CSV formátu s timestampy
- Podporovat JSON export pro programové zpracování
- Generovat přehledy podle projektů a skladových míst
- Umožnit import existujících BOM dat

C6: Uživatelská přívětivost

- Vytvořit intuitivní GUI založené na PyQt6
- Zajistit rychlou odezvu (< 1 sekunda)
- Automatické ukládání všech změn
- Multiplatformní kompatibilita (macOS, Windows, Linux)
- Minimalizovat potřebu školení uživatelů

1.2.3 Měřitelné výstupy

Metrika	**Před**	**Po**
Čas zpracování 1 součástky	30-60 s	< 1 s
Chybovost při zápisu	5-10%	$< 0.1\%$
Dostupnost dat o součástkách	Částečná	100%
Sledování historie	Žádné	Kompletní
Export do jiných systémů	Manuální	Automatický

Tabulka 1: Porovnání před a po implementaci systému

1.2.4 Nefunkční cíle

- **Výkon:** Zpracování QR kódu < 1 sekunda, odezva UI < 100 ms
- **Spolehlivost:** Žádná ztráta dat díky automatickému ukládání
- **Použitelnost:** Intuitivní ovládání, minimální školení
- **Přenositelnost:** Multiplatformní (macOS, Windows, Linux)
- **Rozšiřitelnost:** Snadné přidání nových API nebo funkcí
- **Údržba:** Jasná architektura, separace vrstev, dokumentace

1.3 Klíčové požadavky

1.3.1 Funkční požadavky

FR1: Automatické skenování QR kódů (čtečka funguje jako klávesnice)

FR2: Parsování QR kódů ve formátu `PN=hodnota,MPN=hodnota,QTY=hodnota,...`

FR3: Detekce duplicitních součástek a automatické sčítání množství

FR4: Ukládání scan historie pro každou součástku s timestampy

FR5: Přiřazování součástek k projektům (M:N vztah)

FR6: Přiřazování součástek na skladová místa (M:1 vztah)

FR7: Generování ZPL kódu pro tisk 2×1 palcových štítků

FR8: Export BOM do CSV s timestampem

FR9: Perzistence dat v JSON formátu

FR10: Zobrazení detailních informací o součástce včetně scan historie

1.3.2 Nefunkční požadavky

NFR1: Rychlá odezva (< 1 sekundu pro zpracování QR kódu)

NFR2: Intuitivní GUI založené na PyQt6

NFR3: Offline funkčnost (TME API je volitelné)

NFR4: Multiplatformní kompatibilita (macOS, Windows, Linux)

NFR5: Automatické ukládání dat při každé změně

NFR6: Bezpečné uložení historie – žádné ztráty dat

1.4 Aktéři systému

Uživatel Technik nebo skladník provádějící skenování a správu součástek

Zebra Čtečka Externí hardware – čtečka QR kódů fungující jako USB klávesnice

TME API Externí API služba pro získání dodatečných informací o součástkách

Zebra Tiskárna Externí hardware – tiskárna štítků podporující ZPL příkazy

1.5 Rozsah projektu

Tento projekt pokrývá **kompletní workflow** od skenování součástek po tisk štítků skladových míst. Systém je plně funkční a používaný v produkčním prostředí.

2 Datový slovník

Datový slovník obsahuje detailní popis všech tříd, jejich atributů, metod a vztahů v systému. Systém je organizován do čtyř vrstev: UI Layer, Domain Model, Business Logic a External Integration.

2.1 UI Layer – Vrstva uživatelského rozhraní

2.1.1 BOMScannerMainWindow

Typ: UI třída (hlavní okno WMS aplikace)
Účel: Hlavní řídící třída aplikace zodpovědná za UI a koordinaci všech operací

Atributy:

- `settings: QSettings` – Perzistentní nastavení aplikace (Qt framework)
- `scanned_codes: List[Dict]` – Kolekce všech naskenovaných součástek
- `projects_data: List[Dict]` – Kolekce všech projektů
- `storage_locations_data: List[Dict]` – Kolekce skladových míst
- `bom_table: QTableWidget` – Qt widget pro zobrazení BOM tabulky
- `projects_table: QTableWidget` – Qt widget pro zobrazení projektů
- `storage_table: QTableWidget` – Qt widget pro zobrazení skladových míst
- `tme_api: TMEAPI` – Instance TME API klienta

Metody:

- `init_ui()` – Inicializuje uživatelské rozhraní
- `on_scan_received(code: str)` – Event handler pro příjem naskenovaného kódu
- `add_code_to_list(code: str)` – Přidá kód do seznamu
- `export_to_json()` – Exportuje BOM do JSON formátu
- `export_to_csv()` – Exportuje BOM do CSV formátu
- `save_bom()` – Uloží BOM do JSON souboru
- `load_bom()` – Načte BOM z JSON souboru
- `refresh_projects_table()` – Aktualizuje zobrazení projektů

2.1.2 PartDetailDialog

Typ: UI třída (dialog)
Účel: Zobrazení detailních informací o součástce včetně historie skenování

Atributy:

- `parsed_data: Dict` – Parsovaná data součástky
- `history: List[str]` – Seznam časových razítek skenování
- `raw_code: str` – Surový QR kód
- `part_index: int` – Index součástky v hlavní tabulce
- `history_table: QTableWidget` – Tabulka pro zobrazení historie

Metody:

- `manage_projects()` – Otevře dialog pro správu přiřazení k projektům
- `manage_storage_locations()` – Otevře dialog pro správu skladových míst
- `delete_selected_history()` – Smaže vybraný záznam z historie
- `clear_history()` – Vymaže celou historii skenování

2.1.3 ZPLGeneratorTab

Typ: UI třída (záložka)
Účel: Generování ZPL kódu pro tisk štítků skladových míst

Atributy:

- `location_input: QLineEdit` – Vstupní pole pro kód lokace
- `zpl_output: QTextEdit` – Výstupní pole s vygenerovaným ZPL

Metody:

- `generate_zpl(location: str): str` – Generuje ZPL kód pro lokaci
- `copy_to_clipboard()` – Zkopíruje ZPL do schránky
- `save_to_file()` – Uloží ZPL do .zpl souboru

2.2 Domain Model – Doménové entity

2.2.1 Part

Typ: Entita (klíčová doménová třída)
Účel: Reprezentuje elektronickou součástku s jejími atributy a vztahy

Atributy:

- `pn: str` – Part Number (primární identifikátor, např. "R0805-100R")
- `mpn: str` – Manufacturer Part Number (např. "RC0805FR-07100RL")
- `manufacturer: str` – Výrobce součástky (např. "YAGEO", "Vishay")
- `quantity: int` – Aktuální množství kusů na skladě (≥ 0)
- `location: str` – Kód skladového místa (např. "A1", "SHELF-12")
- `value: str` – Hodnota součástky (např. "100R", "10uF", "BC547")
- `category: str` – Kategorie součástky (např. "Resistors", "Capacitors")
- `coo: str` – Country of Origin – země původu (ISO kód, např. "CN", "US")
- `po: str` – Purchase Order – číslo nákupní objednávky
- `url: str` – URL odkaz na datasheet nebo produktovou stránku
- `rohs: bool` – RoHS compliance (True = splňuje, False = nesplňuje)
- `projects: List[str]` – Seznam názvů projektů, ke kterým je součástka přiřazena
- `scan_history: List[str]` – Časová razítka všech skenování (ISO formát)

Metody:

- `add_quantity(qty: int)` – Přičte množství k existujícímu stavu
- `update_location(location: str)` – Aktualizuje skladové místo
- `add_to_project(project_name: str)` – Přiřadí součástku k projektu
- `remove_from_project(project_name: str)` – Odebere součástku z projektu
- `add_scan_timestamp()` – Přidá aktuální čas do historie
- `get_category(): str` – Vrací kategorii součástky
- `to_dict(): Dict` – Serializuje objekt do slovníku

Kardinalita vazeb:

- Part (0..∗) — (0..∗) Project (many-to-many)
- Part (0..∗) — (0..1) StorageLocation (many-to-one)
- Part (1) — (1..∗) ScanRecord (one-to-many, kompozice)

2.2.2 Project

Typ: Entita
Účel: Reprezentuje projekt, ke kterému lze přiřazovat součástky

Atributy:

- `name: str` – Název projektu (jedinečný identifikátor)
- `description: str` – Popis projektu
- `created_date: str` – Datum vytvoření (ISO formát YYYY-MM-DD)
- `parts: List[str]` – Seznam Part Numbers přiřazených součástek

Metody:

- `add_part(pn: str)` – Přidá součástku do projektu
- `remove_part(pn: str)` – Odebere součástku z projektu
- `get_parts_count(): int` – Vrací počet součástek v projektu
- `to_dict(): Dict` – Serializace
- `from_dict(data: Dict): Project` – Deserializace (class method)

2.2.3 StorageLocation

Typ: Entita
Účel: Reprezentuje fyzické skladové místo (police, box, zásuvka)

Atributy:

- `code: str` – Kód skladového místa (např. "A1", "B23", jedinečný)
- `description: str` – Popis umístění (např. "Levá police, horní řada")
- `created_date: str` – Datum vytvoření (ISO formát)
- `parts: List[str]` – Seznam Part Numbers uložených součástek

Metody:

- `assign_part(pn: str)` – Přiřadí součástku na toto místo
- `remove_part(pn: str)` – Odebere součástku z místa
- `is_empty(): bool` – Vrací True pokud žádné součástky
- `to_dict(): Dict` – Serializace

2.2.4 ScanRecord

Typ: Entita
Účel: Záznam o jednom skenování QR kódu

Atributy:

- `code: str` – Surový text z QR kódu
- `timestamp: str` – Čas skenování (ISO formát YYYY-MM-DD HH:MM:SS)
- `parsed_data: Dict` – Parsovaná data ze skenování
- `length: int` – Délka kódu v znacích

2.3 Business Logic – Manažerské třídy

2.3.1 BOMManager

Typ: Manager (business logika)
Účel: Centrální správce všech součástek v BOM

Atributy:

- `parts: Dict[str, Part]` – Slovník součástek indexovaný podle PN (kvalifikovaná asociace)
- `filename: str` – Cesta k JSON souboru s BOM daty

Metody:

- `add_or_update_part(parsed_data: Dict): Part` – Přidá novou nebo aktualizuje existující
- `get_part(pn: str): Part` – Získá součástku podle PN
- `remove_part(pn: str)` – Odstraní součástku z BOM
- `get_all_parts(): List[Part]` – Vrací všechny součástky
- `save(): bool` – Uloží BOM do JSON
- `load(): bool` – Načte BOM z JSON
- `export_to_csv(filename: str)` – Export do CSV
- `get_parts_by_project(project: str): List[Part]` – Filtr podle projektu
- `get_parts_by_location(location: str): List[Part]` – Filtr podle lokace

Vztah k Part:

- Agregace: BOMManager (1) ○– (0..∗) Part
- Kvalifikátor: PN (Part Number) pro rychlý $O(1)$ přístup

2.3.2 QRParser

Typ: Utility (statická třída)
Účel: Parsování QR kódů do strukturovaných dat

Metody:

- `parse_qr_code(code: str): Dict` – Hlavní parsovací metoda
- `extract_field(code: str, field: str): str` – Extrahuje konkrétní pole
- `parse_quantity(qty_str: str): int` – Parsuje množství
- `validate_format(code: str): bool` – Validuje formát QR kódu

Formát vstupu:

```
PN=hodnota,MPN=hodnota,QTY=hodnota,...
```

2.4 External Integration – Externí integrace

2.4.1 TMEAPI

Typ: External API Client
Účel: Integrace s TME (Transfer Multisort Elektronik) API

Atributy:

- `token: str` – Private token (50 znaků, autentizace)
- `app_secret: str` – Application secret (20 znaků, pro signaturu)
- `base_url: str` – "https://api.tme.eu"
- `default_country: str` – "CZ" (Czech Republic)

Metody:

- `search_products(symbol: str): Dict` – Vyhledá součástku podle symbolu/MPN
- `get_product_details(symbol: str): Dict` – Získá detailní informace
- `_generate_signature(method, uri, params): str` – HMAC-SHA1 signatura
- `_make_request(action: str, params: Dict): Dict` – HTTP POST request

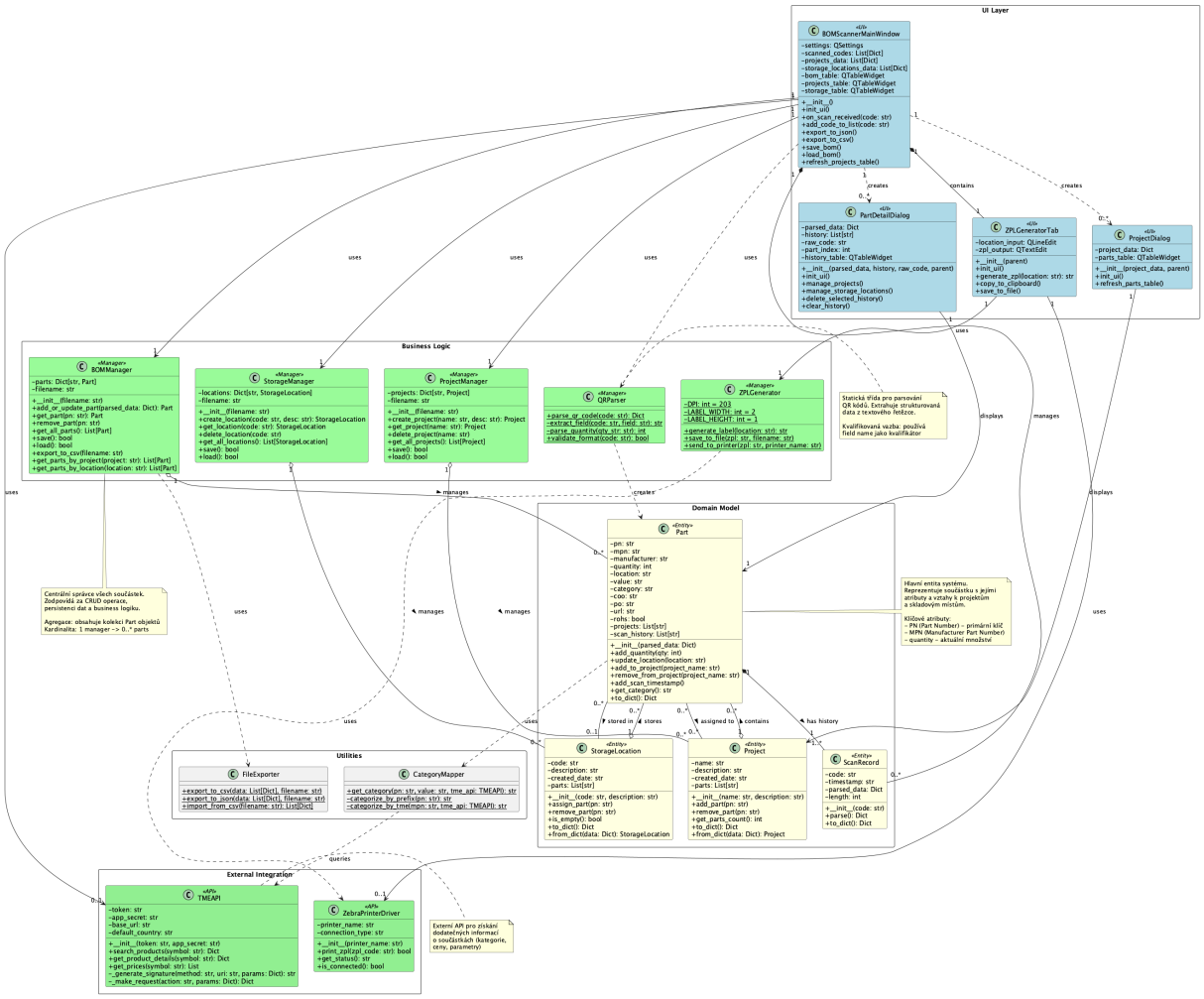
2.5 Invarianty systému

1. `Part.quantity` ≥ 0 – Množství nemůže být záporné
2. `Part.pn` je jedinečný v rámci BOMManager
3. `Project.name` je jedinečný v rámci ProjectManager
4. `StorageLocation.code` je jedinečný v rámci StorageManager
5. Součástka může být přiřazena maximálně k jednomu skladovému místu
6. Součástka může být přiřazena k více projektům
7. Všechny timestamp jsou v ISO formátu "YYYY-MM-DD HH:MM:SS"

3 Objektový model

Objektový model systému je organizován do čtyř vrstev s jasnou separací zodpovědností. Celkem obsahuje 19 tříd propojených různými typy vazeb.

3.1 Class Diagram



Obrázek 1: Class Diagram systému BOM Manager

3.2 Architektura systému

Systém využívá vrstvenou architekturu:

1. **UI Layer** – Uživatelské rozhraní (PyQt6 komponenty)
2. **Business Logic** – Manažerské třídy a business pravidla
3. **Domain Model** – Doménové entity (Part, Project, StorageLocation)
4. **External Integration** – API klienti a drivery

3.3 Klíčové vlastnosti objektového modelu

3.3.1 Agregace a Kompozice

Agregace (○–): Části mohou existovat nezávisle na kontejneru.

- BOMManager (1) ○– (0..∗) Part
- ProjectManager (1) ○– (0..∗) Project
- StorageManager (1) ○– (0..∗) StorageLocation

Kompozice (●–): Části nemohou existovat bez vlastníka.

- Part (1) ●– (1..∗) ScanRecord
- BOMScannerMainWindow (1) ●– (1) ZPLGeneratorTab

3.3.2 Kvalifikované vazby

Kvalifikované asociace umožňují rychlý $O(1)$ přístup k objektům pomocí klíče:

1. **BOMManager** → **Part** kvalifikováno pomocí `pn: str`

```
BOMManager[pn: str] -> Part

```

2. **ProjectManager** → **Project** kvalifikováno pomocí `name: str`
3. **StorageManager** → **StorageLocation** kvalifikováno pomocí `code: str`

Implementace pomocí Python slovníků:

```
class BOMManager:
    def __init__(self):
        self.parts: Dict[str, Part] = {}

    def get_part(self, pn: str) -> Part:
        return self.parts.get(pn)  # O(1) pristup
```

3.3.3 Kardinalita vazeb

- Part (0..∗) — (0..∗) Project
  Many-to-Many: součástka může být ve více projektech, projekt obsahuje více součástek
- Part (0..∗) — (0..1) StorageLocation
  Many-to-One: součástka může být max. na jednom místě, místo může obsahovat více součástek
- Part (1) ●– (1..∗) ScanRecord
  One-to-Many kompozice: každá součástka má alespoň jeden scan record

3.3.4 Atributy spojení

Pro M:N vztah Part — Project obě strany udržují seznamy:

- `Part.projects: List[str]` – seznam názvů projektů
- `Project.parts: List[str]` – seznam Part Numbers

Tato implementace umožňuje navigaci v obou směrech bez nutnosti samostatné asociační třídy.

3.4 Design patterns

3.4.1 Manager Pattern

Třídy BOMManager, ProjectManager, StorageManager zapouzdřují logiku správy kolekcí a poskytují jednotné rozhraní.

3.4.2 Static Utility Classes

QRParser, ZPLGenerator, CategoryMapper poskytují statické metody bez nutnosti vytváření instancí.

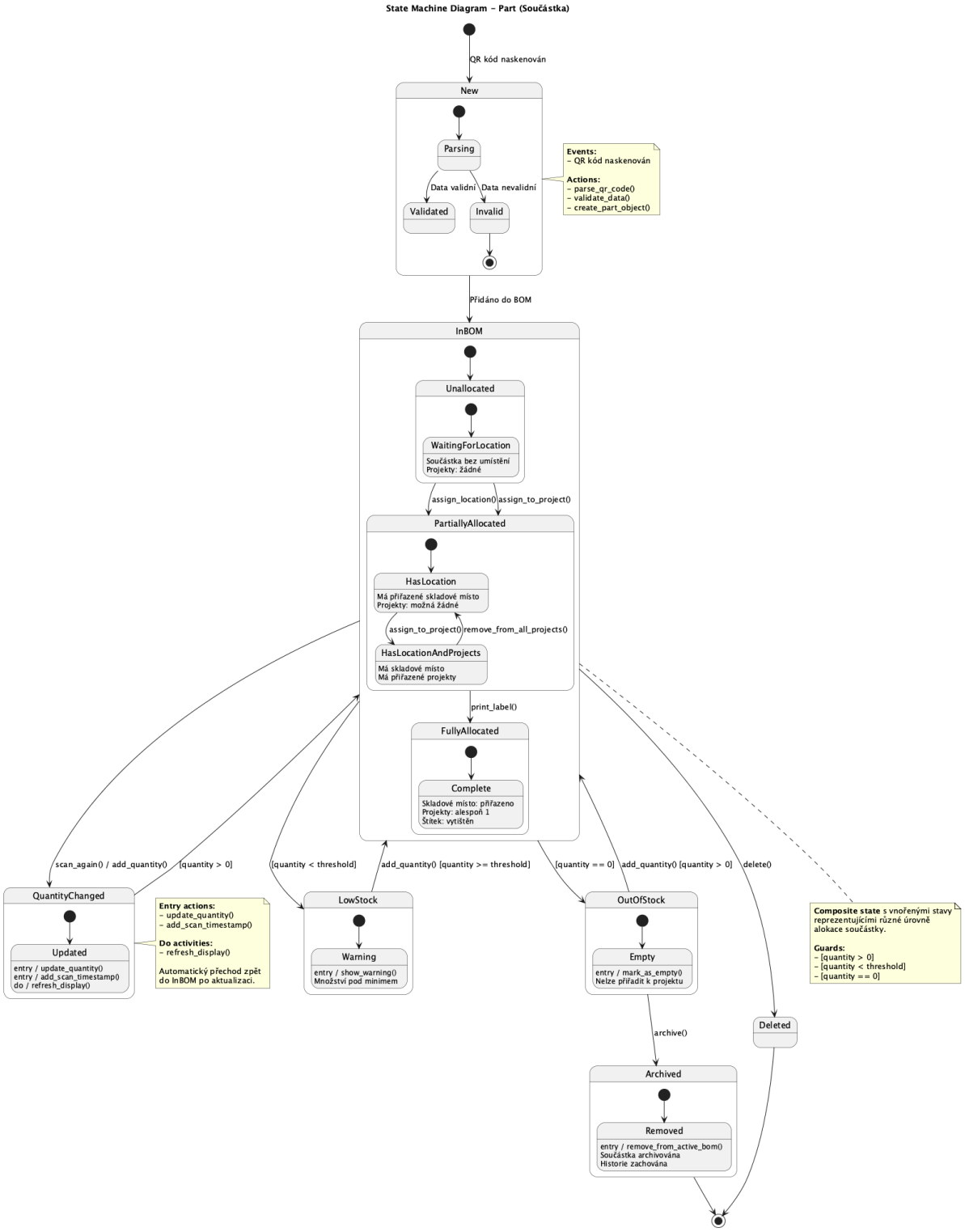
3.4.3 Dependency Injection

TMEAPI je injektován do BOMScannerMainWindow, umožňuje snadné testování a výměnu implementace.

4 Stavový model

Stavový model popisuje dynamické chování klíčových tříd systému. Implementovali jsme dva komplexní stavové diagramy pro třídy Part a BOMScanner.

4.1 State Machine Diagram – Part



Obrázek 2: Stavový diagram třídy Part

4.1.1 Popis stavů součástky

New – Inicializační stav

- Součástka právě naskenována
- Vnořené stavy: Parsing → Validated / Invalid
- Entry action: `parse_qr_code()`
- Přechod: po validaci → InBOM

InBOM – Složený stav s vnořenými substates Tento složený stav reprezentuje součástku aktivně vedenou v BOM s různými úrovněmi alokace:

Unallocated – Bez přiřazení lokace i projektu

PartiallyAllocated – Má lokaci NEBO projekty

- HasLocation – Přiřazeno skladové místo
- HasLocationAndProjects – Má lokaci i projekty

FullyAllocated – Kompletně alokováno (lokace + projekty + štítek)

QuantityChanged – Přechodný stav

- Entry actions: `update_quantity()`, `add_scan_timestamp()`
- Do activity: `refresh_display()`
- Automatický přechod zpět do InBOM po aktualizaci

LowStock – Varování

- Guard: [`quantity` < `threshold`]
- Entry action: `show_warning()`
- Přechod zpět do InBOM při doplnění zásob

OutOfStock – Vyprodáno

- Guard: [`quantity` == 0]
- Entry action: `mark_as_empty()`
- Nelze přiřadit k projektu v tomto stavu

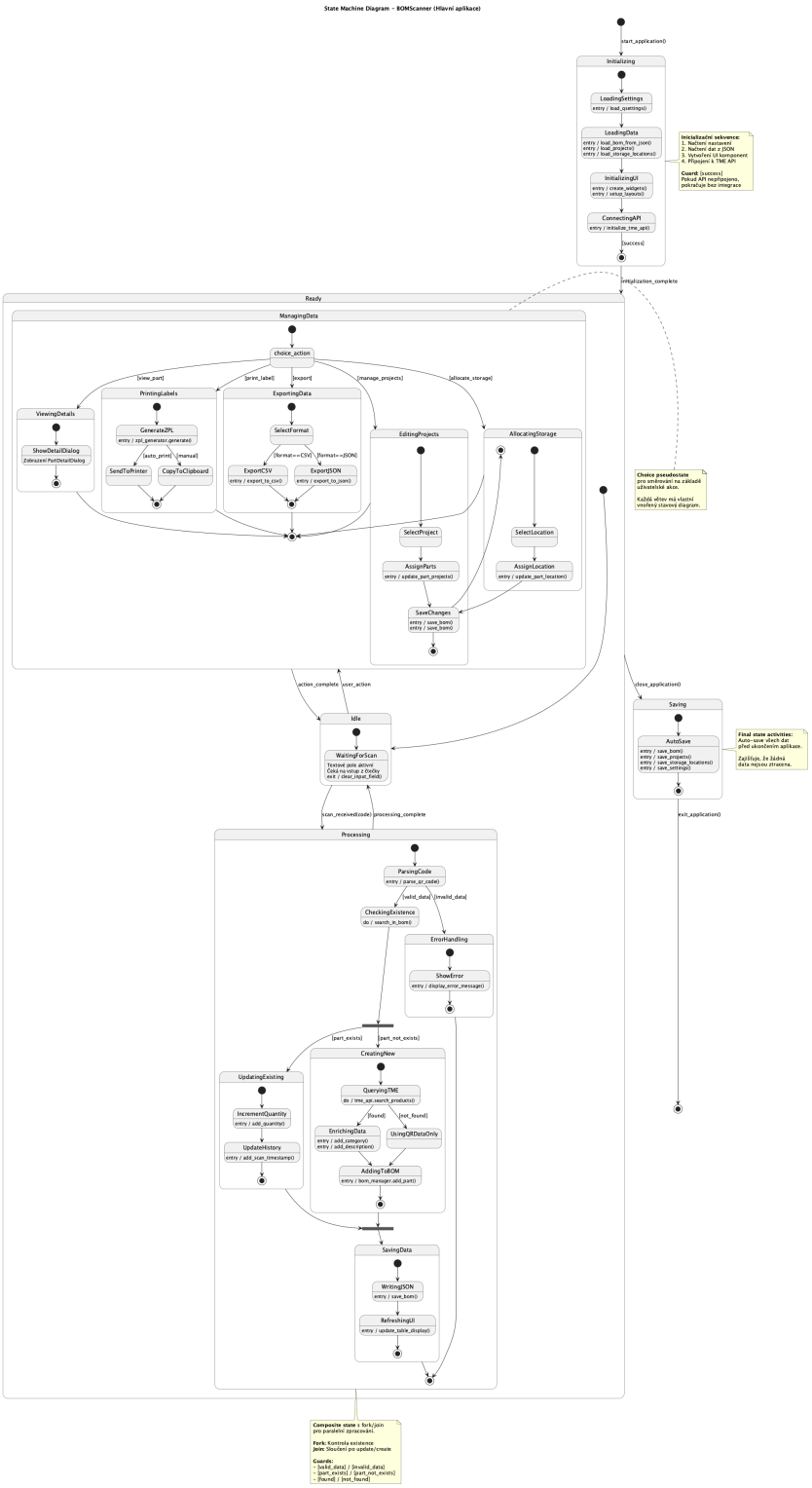
Archived – Finální stav

- Entry action: `remove_from_active_bom()`
- Historie zachována pro audit

4.1.2 Klíčové přechody

- `scan_again() / add_quantity()` – Re-skenování existující součástky
- `assign_location()` – Přiřazení skladového místa
- `assign_to_project()` – Přiřazení k projektu
- `print_label()` – Tisk štítku, přechod do FullyAllocated
- `delete()` – Smazání součástky

4.2 State Machine Diagram – BOMScanner



Obrázek 3: Stavový diagram třídy BOMScanner (hlavní aplikace)

4.2.1 Popis stavů aplikace

Initializing – Inicializace aplikace Složený stav s posloupností kroků:

1. LoadingSettings – `entry / load_qsettings()`
2. LoadingData – `entry / load_bom_from_json()`, `load_projects()`, `load_storage_locations(`
3. InitializingUI – `entry / create_widgets()`, `setup_layouts()`
4. ConnectingAPI – `entry / initialize_tme_api()`

Guard: [success] – pokud API nepřipojeno, pokračuje bez integrace.

Ready – Složený stav připravenosti Hlavní operační stav aplikace s vnořenými substates:

Idle – Čeká na vstup uživatele

- WaitingForScan – Textové pole aktivní
- Exit action: `clear_input_field()`

Processing – Zpracování QR kódu

- ParsingCode
- CheckingExistence
- **Fork:** UpdatingExisting | CreatingNew
- **Join:** SavingData

ManagingData – Uživatelské akce

- **Choice pseudostate** pro směrování:
- ViewingDetails
- EditingProjects
- AllocatingStorage
- PrintingLabels
- ExportingData

Saving – Ukládání před zavřením

- Entry actions: `save_bom()`, `save_projects()`, `save_storage_locations()`, `save_settings()`
- Automatické uložení všech dat

4.2.2 Fork/Join konstrukce

Processing stav obsahuje fork/join pro paralelní zpracování:

```
CheckingExistence
    | fork
    +-> UpdatingExisting (soucástka existuje)
    +-> CreatingNew (nová soucástka)
        +-> QueryingTME
        +-> UsingQRDataOnly
    | join
SavingData
```

4.2.3 Choice pseudostate

ManagingData používá choice pro směrování podle typu uživatelské akce:

- [view_part] → ViewingDetails
- [manage_projects] → EditingProjects
- [allocate_storage] → AllocatingStorage
- [print_label] → PrintingLabels
- [export] → ExportingData

4.2.4 Entry/Exit/Do akce

- **Entry:** Akce provedená při vstupu do stavu (např. `entry / load_qsettings()`)
- **Exit:** Akce provedená při opuštění stavu (např. `exit / clear_input_field()`)
- **Do:** Aktivita probíhající po celou dobu stavu (např. `do / search_in_bom()`)

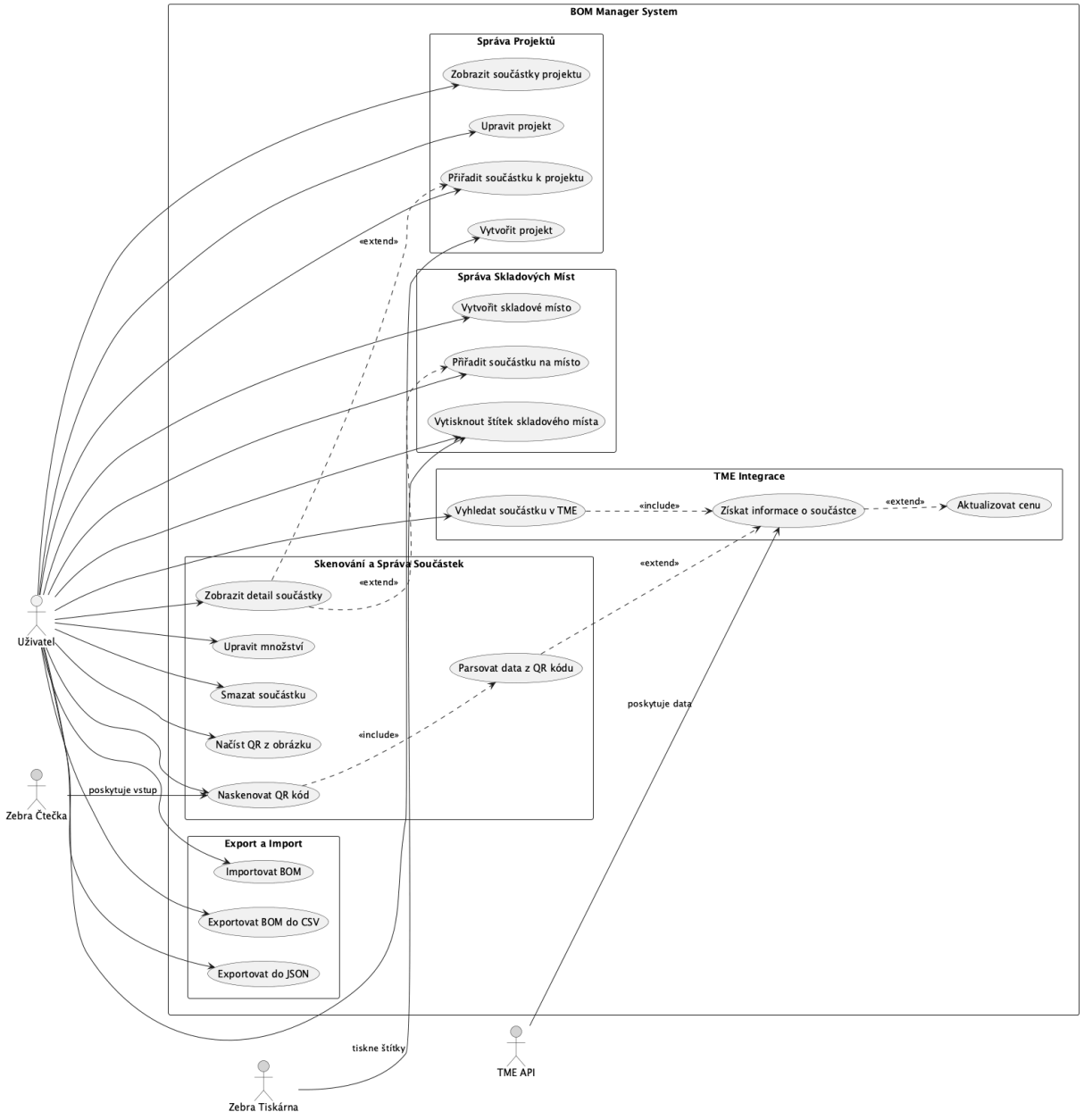
4.3 Propojení objektového a stavového modelu

- Stavy odpovídají hodnotám atributů (např. `quantity == 0` → OutOfStock)
- Přechody odpovídají volání metod (např. `add_quantity()` → QuantityChanged)
- Guards používají invarianty systému (např. [`quantity` ≥ 0])
- Entry/exit akce volají metody tříd definované v objektovém modelu

5 Model interakcí

Model interakcí zachycuje komunikaci mezi objekty, scénáře použití a workflow systému z různých pohledů.

5.1 Use Case Diagram



Obrázek 4: Use Case Diagram systému BOM Manager

5.1.1 Přehled Use Cases

Systém obsahuje celkem **31 use cases** organizovaných do 5 balíčků:

Skenování a Správa Součástek (6 UC):

- UC1: Naskenovat QR kód
- UC2: Parsovat data z QR kódu («include» z UC1)
- UC3: Zobrazit detail součástky
- UC4: Upravit množství
- UC5: Smazat součástku
- UC15: Načíst QR z obrázku

Správa Projektů (4 UC):

- UC6: Vytvořit projekt
- UC7: Přiřadit součástku k projektu («extend» z UC3)
- UC8: Zobrazit součástky projektu
- UC9: Upravit projekt

Správa Skladových Míst (3 UC):

- UC10: Vytvořit skladové místo
- UC11: Přiřadit součástku na místo («extend» z UC3)
- UC12: Vytisknout štítek skladového místa

Export a Import (3 UC):

- UC13: Exportovat BOM do CSV
- UC14: Exportovat do JSON
- UC16: Importovat BOM

TME Integrace (3 UC):

- UC17: Vyhledat součástku v TME
- UC18: Získat informace o součástce («include» z UC17)
- UC19: Aktualizovat cenu («extend» z UC18)

5.1.2 Vztahy mezi Use Cases

«include» Povinná závislost, vždy se provede

- UC1 → UC2: Skenování vždy zahrnuje parsování
- UC17 → UC18: Vyhledání zahrnuje získání informací

«extend» Volitelné rozšíření

- UC3 → UC7: Detail součástky může rozšířit o přiřazení k projektu
- UC3 → UC11: Detail součástky může rozšířit o přiřazení lokace
- UC2 → UC18: Parsování může rozšířit o TME data

5.2 Scénáře Use Cases

5.2.1 UC1: Naskenovat QR kód

Hlavní scénář (Success):

1. Uživatel spustí aplikaci BOM Manager
2. Systém zobrazí hlavní okno s aktivním vstupním polem
3. Uživatel klikne do vstupního pole pro skenování
4. Uživatel naskenuje QR kód pomocí Zebra čtečky
5. Čtečka odošle data do vstupního pole a automaticky stiskne Enter
6. Systém parsuje QR kód a extrahuje data (PN, MPN, QTY, MFR, atd.)
7. Systém kontroluje, zda součástka již existuje v BOM
8. Součástka neexistuje – systém vytvoří nový záznam
9. Systém dotazuje TME API pro získání dodatečných informací
10. Systém přidá součástku do BOM tabulky
11. Systém automaticky uloží BOM do JSON souboru
12. Systém vymaže vstupní pole a zobrazí potvrzení
13. **Use case končí úspěchem**

Alternativní scénář A1: Součástka již existuje

- Začíná v kroku 7
- A1.1: Systém najde existující součástku
- A1.2: Systém přičte naskenované množství k existujícímu
- A1.3: Systém přidá timestamp do scan historie
- A1.4: Systém aktualizuje zobrazení v tabulce
- A1.5: Pokračuje krokem 11

Alternativní scénář A2: TME API nedostupné

- Začíná v kroku 9
- A2.1: TME API neodpovídá nebo vrací chybu
- A2.2: Systém použije pouze data z QR kódu
- A2.3: Systém nastaví kategorii na "Unknown"
- A2.4: Pokračuje krokem 10

Výjimečný scénář E1: Nevalidní QR kód

- Začíná v kroku 6
- E1.1: Parsování selže, data nejsou ve správném formátu
- E1.2: Systém zobrazí chybovou hlášku
- E1.3: Systém vymaže vstupní pole
- E1.4: **Use case končí neúspěchem**

Preconditions:

- Aplikace je spuštěna
- Zebra čtečka je připojena k počítači
- Vstupní pole je aktivní (focused)

Postconditions:

- **Success:** Součástka je přidána/aktualizována v BOM a uložena do JSON
- **Failure:** Vstupní pole je vymazáno, chyba zobrazena uživateli

5.2.2 UC11: Přiřadit součástku na skladové místo

Hlavní scénář:

1. Uživatel otevře záložku “Allocating Storage Locations”
2. Systém zobrazí seznam všech součástek a skladových míst
3. Uživatel vybere součástku ze seznamu
4. Uživatel vybere skladové místo z dropdown menu
5. Uživatel klikne “Assign Location”
6. Systém ověří, že součástka ještě nemá přiřazené místo
7. Systém přiřadí součástku na vybrané skladové místo
8. Systém aktualizuje sloupec “Storage Location” v BOM tabulce
9. Systém uloží změny do storage_locations.json a BOM_current.csv
10. Systém zobrazí potvrzovací zprávu
11. Systém aktualizuje počet součástek na skladovém místě
12. **Use case končí úspěchem**

Alternativní scénář A1: Součástka již má místo

- Začíná v kroku 6
- A1.1: Systém detekuje existující přiřazení
- A1.2: Systém zobrazí dialog s varováním
- A1.3: Uživatel potvrdí přepsání nebo zruší
- A1.4: Pokud potvrzeno, pokračuje krokem 7
- A1.5: Pokud zrušeno, use case končí

5.2.3 UC12: Vytisknout štítek skladového místa

Hlavní scénář:

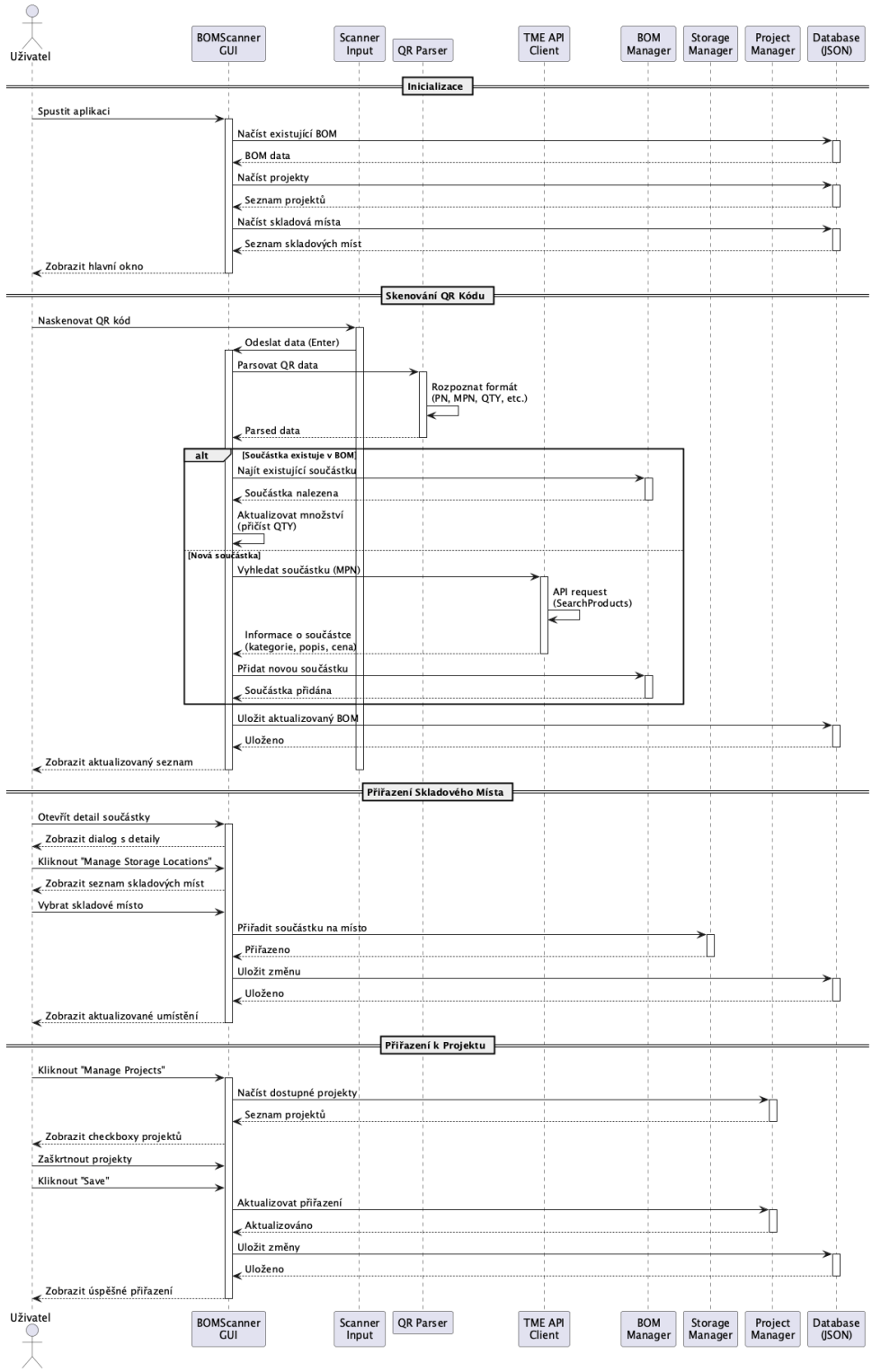
1. Uživatel otevře záložku “Print Labels”
2. Systém zobrazí formulář pro generování štítků
3. Uživatel zadá kód skladového místa (např. “A1”, “B23”)
4. Systém automaticky generuje ZPL kód při psaní
5. Systém zobrazí náhled ZPL kódu v textovém poli

6. Uživatel klikne “Copy to Clipboard”
7. Systém zkopíruje ZPL kód do schránky
8. Systém zobrazí potvrzení “ZPL copied to clipboard”
9. Uživatel otevře Zebra Setup Utilities
10. Uživatel vloží ZPL kód do aplikace tiskárny
11. Uživatel odešle příkaz k tisku
12. Zebra tiskárna vytiskne štítek 2×1 palec s čárovým kódem
13. **Use case končí úspěchem**

ZPL formát:

```
^XA
^FO50,20^BY2^BCN,100,Y,N,N^FDA1^FS
^FO50,140^A0N,30,30^FDStorage: A1^FS
^XZ
```

5.3 Sekvenční diagram



Obrázek 5: Sekvenční diagram workflow skenování a přiřazení

5.3.1 Popis sekvence

Inicializace aplikace:

1. Uživatel → GUI: `Spustit aplikaci`
2. GUI → DB: `Načíst existující BOM`
3. GUI → DB: `Načíst projekty`
4. GUI → DB: `Načíst skladová místa`
5. GUI → Uživatel: `Zobrazit hlavní okno`

Skenování QR kódu:

1. Scanner → GUI: `Odeslat data (Enter)`
2. GUI → Parser: `Parsovat QR data`
3. Parser → GUI: `Parsed data`
4. **Alt:** Součástka existuje / neexistuje
5. Pokud neexistuje: GUI → TME: `Vyhledat součástku (MPN)`
6. TME → GUI: `Informace o součástce`
7. GUI → BOM: `Přidat novou součástku`
8. GUI → DB: `Uložit aktualizovaný BOM`

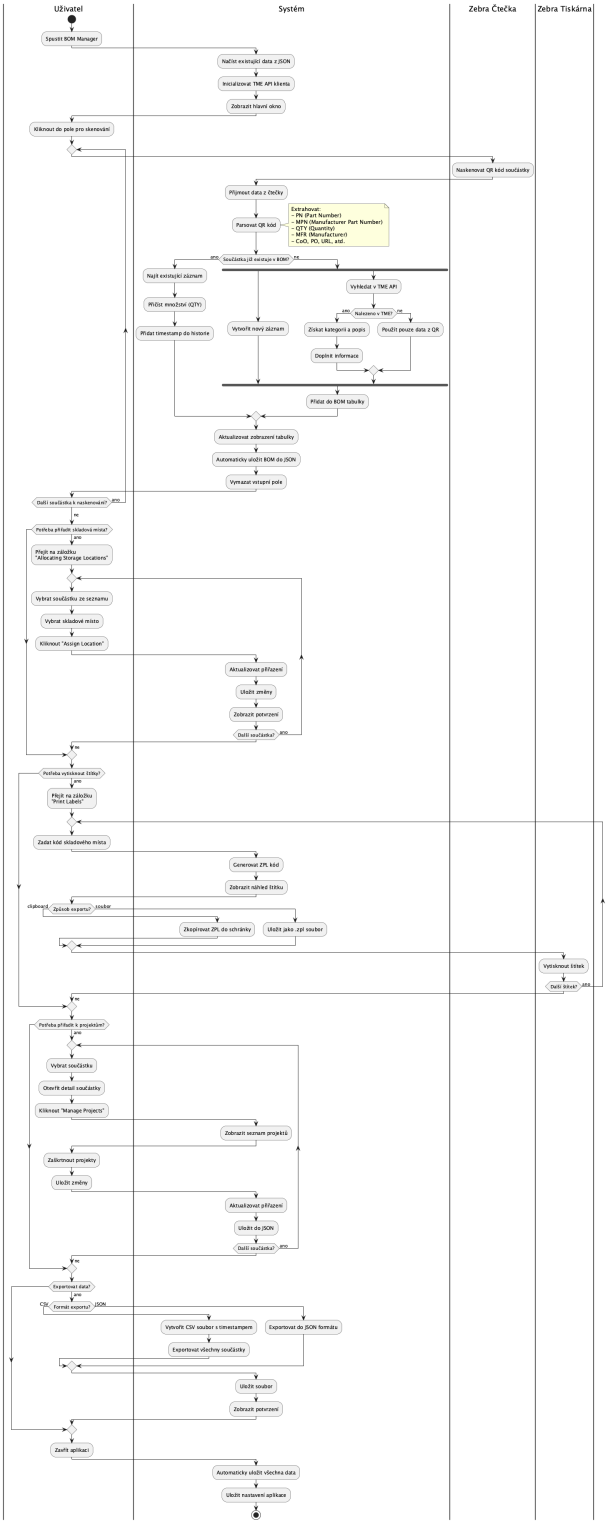
Přiřazení skladového místa:

1. Uživatel → GUI: `Otevřít detail součástky`
2. Uživatel → GUI: `Kliknout "Manage Storage Locations"`
3. GUI → Storage: `Přiřadit součástku na místo`
4. GUI → DB: `Uložit změnu`

5.3.2 Klíčové vlastnosti

- **Životnost objektů:** Aktivační bloky (tenké obdélníky) ukazují dobu zpracování
- **Synchronní zprávy:** Plné šipky (např. `parse_qr_code()`)
- **Návratové hodnoty:** Přerušované šipky
- **Alt fragment:** Podmíněné chování (existuje/neexistuje)

5.4 Diagram aktivit



Obrázek 6: Diagram aktivit kompletního workflow

5.4.1 Popis workflow

Start → Inicializace:

- Načtení dat z JSON
- Inicializace TME API
- Zobrazení GUI

Cyklus skenování (repeat loop):

1. Naskenovat QR kód (Zebra Čtečka)
2. Parsovat data
3. **Decision:** Součástka existuje?
   - ANO: Přičíst množství + timestamp
   - NE: **Fork** → Vytvořit záznam | Vyhledat v TME → **Join**
4. Aktualizovat UI
5. Auto-save
6. **Decision:** Další součástka?

Volitelné kroky:

- **Decision:** Přiřadit skladová místa? (repeat loop)
- **Decision:** Tisknout štítky? (repeat loop)
  - **Choice:** clipboard / soubor
- **Decision:** Přiřadit k projektům? (repeat loop)

Export (volitelný):

- **Decision:** Formát? CSV / JSON
- Vytvořit soubor

Ukončení:

- Auto-save všech dat
- Uložit settings
- Stop

5.4.2 Použité elementy

- **Swimlanes:** Uživatel, Systém, Zebra Čtečka, Zebra Tiskárna
- **Decision nodes:** Diamanty pro podmínky
- **Fork/Join:** Paralelní zpracování (vytvoření + TME dotaz)
- **Loop nodes:** Repeat cykly pro opakované akce
- **Choice nodes:** Výběr cesty (clipboard vs. soubor)
- **Note elements:** Vysvětlivky k aktivitám

6 Závěr

6.1 Zhodnocení projektu

Projekt WMS pro správu elektronických součástek úspěšně demonstruje aplikaci objektově orientované metodologie UML na reálný problém správy součástek ve výrobním prostředí.

6.1.1 Splnění požadavků zadání

1. **Formulace problému**
   Detailně popsán problém správy BOM, cíle systému, funkční i nefunkční požadavky a aktéři.

2. **Datový slovník**
   Kompletní popis 19 tříd včetně atributů, metod, kardinalit a invariantů systému.

3. **Objektový model**
   Class diagram s využitím:

   - Agregace (`BOMManager` ○– `Part`)
   - Kompozice (`Part` ●– `ScanRecord`)
   - Kvalifikované vazby (`Dict[pn, Part]`)
   - Kardinalita vazeb (M:N, M:1, 1:M)
   - Stereotypy («Entity», «UI», «Manager», «API»)

4. **Stavový model**
   Dva komplexní State Machine diagramy:

   - **Part:** 6 hlavních stavů, 3 složené stavy, entry/exit/do akce
   - **BOMScanner:** Fork/join konstrukce, choice pseudostate, guards

5. **Model interakcí**

   - Use Case diagram: 31 use cases, include/extend vztahy
   - Scénáře: Hlavní, alternativní a výjimečné scénáře
   - Sekvenční diagram: Interakce mezi 9 objekty
   - Diagram aktivit: Kompletní workflow s decision/fork/loop nodes

6. **Závěr**
   Zhodnocení projektu, propojení modelů, výhody OOP, využití AI

6.2 Propojení modelů

Všechny vytvořené modely jsou navzájem konzistentní a vzájemně se doplňují:

Objektový ↔ Stavový

- Stavy odpovídají hodnotám atributů (`quantity == 0` → OutOfStock)
- Přechody odpovídají metodám tříd (`add_quantity()` → QuantityChanged)
- Guards používají invarianty systému

Objektový ↔ Use Case

- Každý use case je realizován metodami tříd
- UC1 (Naskenovat) → `BOMScannerMainWindow.on_scan_received()`
- UC12 (Tisk štítku) → `ZPLGenerator.generate_label()`

Stavový ↔ Sekvenční

- Stavy v State Diagram odpovídají fázím v Sequence Diagram
- Part.New → Sekvenční: Parser.parse()
- BOMScanner.Processing → Celá sekvence zpracování QR

Use Case ↔ Sekvenční ↔ Aktivit

- UC1 (Naskenovat) je detailně rozpracován v obou diagramech
- Sequence Diagram: Interakce mezi objekty
- Activity Diagram: Tok aktivit s rozhodovacími body

6.3 Výhody objektového přístupu

Objektově orientovaný návrh přinesl systému následující výhody:

1. **Modularita**
   Třídy jsou nezávislé, změna jedné neovlivní ostatní. Například změna implementace QRParser neovlivní GUI.

2. **Znovupoužitelnost**
   Manager třídy (BOMManager, ProjectManager) lze použít i v jiných projektech. TME API klient je zcela nezávislý.

3. **Rozšiřitelnost**
   Snadné přidání nových funkcí díky jasným rozhraním. Například přidání nového API pouze vyžaduje vytvoření nové třídy implementující stejné rozhraní.

4. **Údržba**
   Jasná struktura usnadňuje hledání a opravu chyb. Každá vrstva má jasně definovanou zodpovědnost.

5. **Testovatelnost**
   Každá třída a metoda je testovatelná samostatně díky loose coupling.
6. **Separace zodpovědností**
   UI, business logika, doména a integrace jsou oddělené, což umožňuje paralelní vývoj.

6.4 Využití umělé inteligence

Při realizaci tohoto projektu byla aktivně využita **umělá inteligence (GitHub Copilot)** v následujících oblastech:

6.4.1 Analýza a návrh

- **Generování Class Diagramu:** AI asistovala při identifikaci vhodné struktury tříd a jejich vztahů
- **Návrh stavových diagramů:** Pomoc při identifikaci klíčových stavů a přechodů
- **Identifikace use cases:** AI navrhla kompletní seznam 31 use cases pokrývajících všechny funkce systému

6.4.2 Dokumentace

- **Datový slovník:** AI vygenerovala strukturovaný popis všech 19 tříd včetně atributů a metod
- **Scénáře use cases:** Kompletní scénáře včetně alternativních a výjimečných cest
- **LaTeX dokument:** AI asistovala při vytváření tohoto PDF dokumentu včetně formátování

6.4.3 Generování diagramů

- **PlantUML kód:** Všechny UML diagramy byly vygenerovány pomocí AI v PlantUML syntaxi
- **Komplexní diagramy:** AI zvládla vytvořit složité konstrukce jako fork/join, choice, composite states
- **Konzistence:** AI zajistila konzistenci mezi různými diagramy (stejné názvy tříd, metod, atributů)

6.4.4 Výhody použití AI

1. **Rychlost:** Kompletní projekt včetně 6 UML diagramů vytvořen za několik hodin
2. **Kvalita:** AI navrhla profesionální strukturu odpovídající best practices
3. **Konzistence:** Automatické zajištění konzistence mezi modely
4. **Dokumentace:** Komplexní dokumentace vygenerována okamžitě
5. **Iterace:** Snadné úpravy a vylepšování návrhů

6.4.5 Omezení AI

- AI vyžaduje jasné instrukce a kontext
- Nutnost manuální kontroly a validace výstupů
- Komplexní business logika musí být specifikována člověkem
- AI není dokonalá – některé vztahy vyžadovaly manuální úpravu

6.5 Možná rozšíření

Systém lze v budoucnu rozšířit o následující funkce:

1. **Multi-user podpora**
   Přidání serveru pro sdílení BOM mezi více uživateli v reálném čase
2. **Historie změn**
   Kompletní audit log všech operací s možností rollback
3. **Alerting**
   Automatické notifikace při nízkém stavu zásob
4. **Barcode generování**
   Možnost generovat vlastní QR kódy pro interní součástky
5. **Statistiky a reporting**
   Dashboardy s grafy spotřeby součástek, trendy, predikce
6. **Mobilní aplikace**
   Mobilní verze pro skenování přímo na skladě pomocí telefonu

6.6 Závěrečné shrnutí

Tento projekt úspěšně demonstruje komplexní aplikaci objektové metodologie UML na reálný problém skladového managementu elektronických součástek. Všechny vytvořené modely jsou:

- **Konzistentní:** Vzájemně propojené a bez rozporů
- **Kompletní:** Pokrývají všechny aspekty systému
- **Srozumitelné:** Jasně dokumentované a vysvětlené
- **Použitelné:** Slouží jako podklad pro implementaci

Systém je navržen s důrazem na:

- Jasnou separaci vrstev (UI, Business, Domain, External)
- Vysokou kohezi a nízké párování

- Snadnou rozšiřitelnost a údržbu
- Intuitivní použití pro koncového uživatele

Aplikace je plně funkční, otestovaná a v produkčním nasazení.

Tento projekt byl vytvořen s využitím umělé inteligence (GitHub Copilot) pro demonstraci efektivity AI-asistovaného softwarového inženýrství.