FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Systém pro vytváření digitálních dvojčat skladů ve frameworku NVIDIA Omniverse

Jakub Křižanovský

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Systém pro vytváření digitálních dvojčat skladů ve frameworku NVIDIA Omniverse

Jakub Křižanovský

Vedoucí práce
Ing. Jiří Dobrý

PLZEŇ 2024

Citace v seznamu literatury:
KŘIŽANOVSKÝ, Jakub. *Systém pro vytváření digitálních dvojčat skladů ve frameworku NVIDIA Omniverse*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Jiří Dobrý.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub KŘIŽANOVSKÝ**
Osobní číslo:	**A21B0192P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Systém pro vytváření digitálních dvojčat skladů ve frameworku NVIDIA Omniverse**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s možnostmi a funkcemi softwarového frameworku NVIDIA Omniverse se zaměřením na implementaci digitálních dvojčat skladů a továren.
2. Navrhněte systém pro intuitivní tvorbu digitálních dvojčat skladů a jejich následnou vizualizaci s možností zobrazení statistických veličin z dat načítaných z externího systému v 3D modelu.
3. Implementujte systém dle návrhu z předchozího bodu využitím frameworku NVIDIA Omniverse.
4. Na jednoduchém projektu digitálního dvojčete skladu reálného/fiktivního zákazníka ověřte funkčnost navrženého řešení. Zdokumentujte systém z uživatelského i programátorského hlediska.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Jiří Dobrý**
Aimtec a. s. , Plzeň

Konzultant bakalářské práce: **Ing. Martin Červenka**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 2. května 2024

. . . . . .
Jakub Křižanovský

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Tato práce se zaměřuje na vytvoření systému pro tvorbu a prohlížení digitálních dvojčat skladů využitím frameworku **NVIDIA Omniverse**. Hlavním cílem práce je ověřit vhodnost použití frameworku pro tento účel. V rámci práce jsou prozkoumány možnosti frameworku **NVIDIA Omniverse** včetně technologie **Universal Scene Description**, na které je framework postavený. Dále je navržen a implementován systém skládající se ze dvou aplikací. Funkčnost systému je ověřena na jednoduchém projektu digitálního dvojčete.

Abstract

This thesis, titled *System for warehouse digital twin creation in the NVIDIA Omniverse framework*, focuses on creating a system for creating and viewing digital twins of warehouses using the **NVIDIA Omniverse** framework. The main goal of the thesis is to verify the suitability of using the framework for this purpose. The thesis explores the possibilities provided by the **NVIDIA Omniverse** framework, including the **Universal Scene Description** technology on which the framework is built. Furthermore, a system consisting of two applications is designed and implemented. The functionality of the system is verified through a simple digital twin project.

Klíčová slova

logistika • digitální dvojče • 3D grafika • Universal Scene Description • NVIDIA Omniverse

Poděkování

Na tomto místě bych chtěl poděkovat všem, kteří přispěli k úspěšnému dokončení mé bakalářské práce.

Především bych chtěl poděkovat vedoucímu práce, panu Ing. Jiřímu Dobrému, za profesionální vedení, odborné rady, konstruktivní kritiku a vstřícný přístup, které mi pomohly práci dotáhnout do konce.

Dále bych rád poděkoval panu Ing. Martinu Červenkovi za ochotu a cenné připomínky k textu práce, které mi pomohly zkvalitnit jeho finální podobu.

V neposlední radě bych chtěl poděkovat mé rodině za jejich podporu a motivaci během celého studia.

Obsah

Úvod 1

Digitální dvojčata skladů můžou být efektivním nástrojem který může přinést řadu výhod ve skladovacích operacích.

Společnost **NVIDIA** v nedávné době vytvořila novou platformu **NVIDIA Omniverse**, která využívá technologii **Universal Scene Description** pro tvorbu, popis a vizualizace 3D scén.

Hlavním cílem této práce, je ověřit, zda by tato platforma byla vhodná pro tvorbu digitálních dvojčat. Důkazem bude implementace systému určeného pro jejich tvorbu a prohlížení.

Společnost **AIMTEC a. s.** již nabízí řešení pro digitální dvojčata skladů - systém **Generic Warehouse Digital Twin**. Avšak problémem tohoto systému je, že proces tvorby digitálních dvojčat je velmi časově náročný a může vývojáři trvat i více jak týden práce. Dalším cílem této práce je tento proces urychlit tak, aby v ideálním případě nezabral vývojáři seznámenému se systémem více jak den.

Kapitola 2 poskytuje letmý pohled do světa logistiky a systémů pro správu skladů a pojednává o společnosti **AIMTEC a. s.** a jejích produktech. Kapitola 3 vystvětluje pojem digitální dvojče a představuje výhody digitálních dvojčat skladů. Kapitola také ukazuje a porovnává různé možnosti a frameworky, které by se daly pro implementaci digitální dvojčat skladů použít.

Následující kapitoly se zaměřují na framework **Universal Scene Description** (**USD**) a platformu **NVIDIA Omniverse**. Kapitola 4 detailně rozebírá možnosti **USD**, zatímco kapitola 5 se věnuje platformě **NVIDIA Omniverse**.

Dále následují kapitoly 6 - 8 zabývající se systémem pro tvorbu digitálních dvojčat, který v rámci práce vznikne. Kapitola 6 definuje požadavky na funkcionalitu, které by měl výsledný systém splňovat. Kapitola 7 popisuje návrh a architekturu systému na vysoké úrovni, zatímco kapitola 8 se věnuje detailnímu členění a implementaci jednotlivých částí.

Nakonec je v kapitole 9 představen výsledný systém a zhodnocen framework **NVIDIA Omniverse**.

Logistika 2

Definice pojmu logistika není úplně jednotná. Pro představu lze použít například následující:

Logistika je proces plánování, skladování a provádění efektivní přepravy zboží od místa původu do místa spotřeby. Cílem logistiky je splnit požadavky zákazníků včas a nákladově efektivně. [1]

Cílem této kapitoly určitě není vysvětlení celé logistiky, všech jejích pojmů a odvětví, ale pouze poskytnutí určitého kontextu a stručného přehledu nezbytných logistických principů relevantních pro tuto práci.

Hlavní dvě funkce logistického průmyslu jsou přeprava a skladování. Obě jsou složitými úkoly a společnosti pro ně typicky používají komplexní softwarové systémy. Pro přepravu se používá systém správy přepravy (*transportation management system - TMS*) a pro sklady systém správy skladu (*warehouse management system - WMS*). V rámci této práce bude potřeba hlavně znalost *WMS* a proto na něj text dále zaměří. Na obrázku 2.1 se nachází ukázka skladu. [1]

Obrázek 2.1: Ukázka skladu [2]

2.1 WMS

WMS se skládá z softwaru a procesů, které umožňují organizacím řídit a spravovat skladové operace od okamžiku, kdy zboží nebo materiály vstoupí do skladu, až do chvíle, kdy ho opouštějí.

Sklady jsou v centru výrobních a logistických operací, protože obsahují veškerý materiál používaný nebo vyrobený v těchto procesech, od surovin po hotové výrobky. Účelem *WMS* je zajistit, že zboží a materiály procházejí sklady co nejúčinnějším a nákladově efektivním způsobem. *WMS* zpracovává mnoho funkcí umožňujících tyto pohyby, včetně sledování skladových zásob, kompletace objednávek, příjmu a umístění zboží.

WMS také poskytuje přehled o zásobách organizace v jakémkoliv místě a čase, ať už se nacházejí ve skladu nebo jsou v pohybu. [3]

2.2 AIMTEC a. s.

AIMTEC a. s. je plzeňská společnost, která se zabývá především digitalizací a automatizací výroby a logistiky. Na trhu je více jak 25 let a během té doby se z ní stala globální společnost se zákazníky po celém světě. [4]

Společnost je v rámci práce zmíněna z důvodu její role jako zadavatele projektu. Systém vyvíjený v rámci této práce navíc využívá jako své datové zdroje externí systémy, které jsou produkty této společnosti.

2.2.1 Aimtec DCIx

Aimtec DCIx je jedním z hlavních produktů společnosti **AIMTEC a. s**. Jedná se o digitalizační platformu, která propojuje logistiku, výrobu, kvalitu, expedici, automatizační technologie, stroje a lidi a dovoluje tyto věci řídit z jednoho místa.

Dále má velké možnosti integrací na jiné externí systémy (např. **SAP**), což platformě dodává velkou flexibilitu.

Součástí funkcí platformy je také *WMS*, ze kterého bude možné načítat data pro tvorbu digitálního dvojčete. [5]

2.2.2 Generic Warehouse Digital Twin

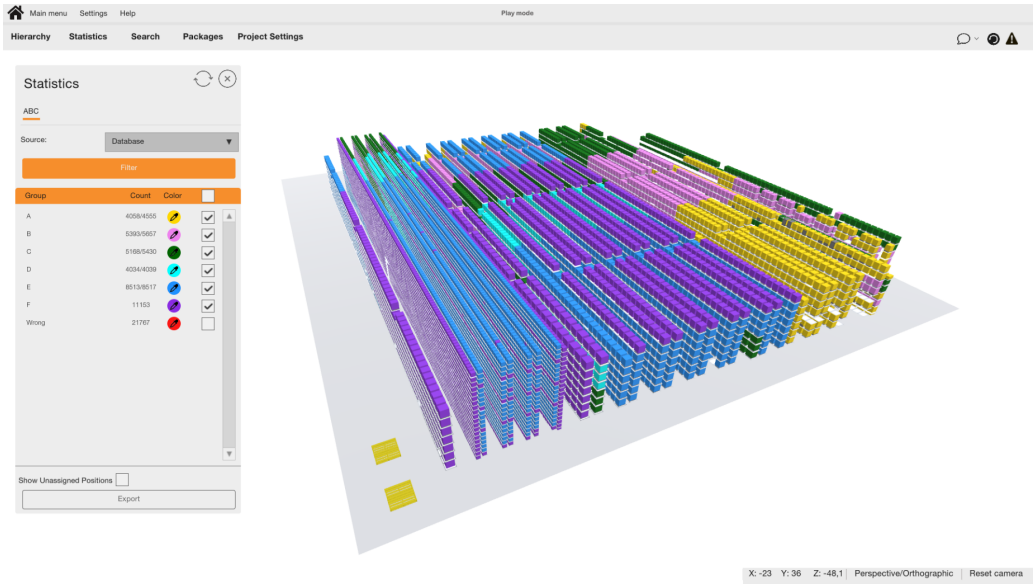
Generic Warehouse Digital Twin (dále jen **GDTW**) je modulem produktu **Aimtec DCIx** a je současným řešením firmy pro tvorbu a zobrazování digitálních dvojčat skladů. Obecný pojem digitální dvojče bude vysvětlen v kapitole 3.

Jedná se o aplikaci složenou ze dvou částí - back-end a front-end. Back-end aplikace je napsaný v programovacím jazyce *Java* za použití frameworku *Spring*. Je

napojený na databázi systému **Aimtec DCIx** a poskytuje *REST API*, které front-end využívá jako datový zdroj.

Front-end aplikace je napsaný v herním enginu **Unity** a může běžet buď jako tlustý klient, nebo jako webový klient použitím technologie *WebGL*. Může běžet ve dvou režimech - *Edit* a *View*. Režim *Edit* slouží k tvorbě a úpravám digitálního dvojčete a klient při něm musí mít přístup přímo k databázi **Aimtec DCIx** (je možné pouze jako tlustý klient). V režimu *View* je pak možné vytvořené digitální dvojče interaktvně prohlížet, vyhledávat v něm a zobrazovat různé statistiky. Ukázka systému **GDTW** v režimu *View* se nachází na obrázku 2.2.

Předmětem této práce bude vytvořit nový systém ve frameworku **NVIDIA Omniverse**, který jako zdroj dat využije stávající back-end. Front-end nového systému by měl plně nahradit ten původní, přičemž zachová plnou funkcionalitu režimů *View* i *Edit*.



Obrázek 2.2: Ukázka systému **Generic Warehouse Digital Twin**

Digitální dvojče 3

Digitální dvojče je dynamická virtuální kopie fyzického předmětu, procesu, systému nebo prostředí, která vypadá a chová se identicky jako její reálný protějšek ve skutečném světě. Digitální dvojče přijímá data a replikuje procesy tak, aby bylo možné předpovídat možné výsledky a problémy, kterým by skutečný produkt ve skutečném světě mohl čelit.

Hlavní síla digitálních dvojčat spočívá v propojení fyzických objektů s daty z reálného světa, což usnadňuje jejich vizualizaci. Digitální dvojče, poháněné technologií počítačové grafiky, která dokáže vykreslovat obsah v reálném čase, má schopnosti kombinovat, organizovat a prezentovat různé zdroje dat formou realistických a interaktivních vizualizací. Tyto vizualizace umožňují uživatelům snadněji a efektivněji získávat klíčové informace než například při práci s rozsáhlými tabulkami. [6]

3.1 Digitální dvojče skladu

Tvorba a využití digitálních dvojčat může přinést řadu výhod ve skladovacích operacích. Využitím dat z *WMS* a případnou kombinací s využitím sítě internetu věcí (*IoT*) může digitální dvojče poskytnout real-time model skladu k interakci a monitorování. Příklady využití digitálních dvojčat skladů jsou popsány v následujících podkapitolách.

3.1.1 Monitorování skladových zásob

Digitální dvojče skladu dovoluje efektivně monitorovat stav skladových zásob tím, že v reálném čase poskytuje detailní informace o poloze, stavu a pohybu každé položky. Tato úroveň přehledu může minimalizovat riziko vyčerpání skladových zásob, zrychlovat vyřizování objednávek a snižovat nadbytečné zásoby. [7]

3.1.2 Optimalizace logistických procesů

Digitální dvojče skladu může být také nástrojem pro optimalizaci logistických procesů ve skladech. Jeho využitím lze simulovat a monitorovat skladové operace v reálném čase, což může vést k efektivnějšímu využívání prostoru, zvýšení efektivity toků zboží a snižování chyb při správě inventáře. Tímto způsobem dodává významný přínos pro modernizaci a zefektivnění logistických operací ve skladu. [8]

3.1.3 Testování nových pracovní postupů

Digitální dvojčata umožňují testovat nové pracovní postupy. Manažeři často museli vytvářet efektivní pracovní postupy až po průchodu procesem pokusů a omylů. S využitím digitálních dvojčat lze však nové pracovní postupy testovat virtuálně. Digitální dvojčata mohou zohlednit nové proměnné a parametry pro posouzení dopadu takových změn ve skladu předtím, než se změny skutečně provedou. Mohou tak být vnímána jako skvělý způsob návrhu nových automatizačních systémů, přizpůsobení se měnícím se požadavkům nebo řešení jedinečných situací. [8]

3.1.4 Školení personálu

Digitální dvojče je také možné využít pro školení zaměstnanců například spojením s technologií virtuální nebo rozšířené reality. To může například snížit riziko úrazů těchto zaměstnanců. [8]

3.2 Možnosti implementace

3.2.1 Existující produkt

První možností pro nasazení digitálního dvojčete by bylo použití nějakého již hotového produktu přímo pro tvorbu digitálních dvojčat. Příkladem takového produktu je například platforma **Twinzo**. [9]

Hlavní výhodou tohoto řešení je jeho jednoduchost. Kromě integrace na datový zdroj není potřeba prakticky nic implementovat a jde tedy čistě o tvorbu digitálního dvojčete samotného.

Nevýhodou by pak byla malá flexibilita řešení. Můžeme využít pouze funkce a vizualizace, které platforma nabízí.

3.2.2 Vytvoření vlastní aplikace

Jinou možností je tvorba celého systému od základu za použití knihovny pro vykreslení 3D grafiky.

Velkou výhodou tohoto řešení je jeho velká flexibilita. Pro systém lze tímto způsobem zvolit prakticky libovolné technologie a přizpůsobit ho přesně požadavkům.

Na druhou stranu nevýhodou této možnosti je přílišná složitost a časová náročnost vývoje a implementace.

3.2.3 Implementace v herním enginu

Další možností pro tvorbu systému by bylo použití nějakého herního enginu např. **Unity**. [10]

Tato možnost stále zachovává vysokou flexibilitu. Velkou výhodou oproti tvorbě systému od základu je ale navíc velká škála běžných funkcionalit, které už jsou implementované a engine je tak řeší za vás.

Jak již bylo zmíněno v sekci 2.2.2, společnost **AIMTEC a. s.** se jednou již touto cestou úspěšně vydala při tvorbě systému **GDTW** vytvořeném právě v herním enginu **Unity**.

3.2.4 Simulační nástroj

Další možnost představují simulační nástroje, například **FlexSim**. [11]

Výhodou tohoto řešení je možnost simulovat chování systému v reálném čase. Tato možnost může být především výhodná v prostředích, kde je klíčové porozumět interakcím mezi různými částmi systému a optimalizovat je.

Nevýhodou může být specifičnost těchto nástrojů pro oblast simulací a jejich omezená schopnost přizpůsobit se ostatním potřebám systému. Problematická může být například integrace s externími systémy pro načítání dat.

3.2.5 NVIDIA Omniverse

Možností, která byla vybrána pro tvorbu systému, je použití nové platformy **NVIDIA Omniverse** založené na technologii **Universal Scene Description**.

Tato možnost zachovává většinu výhod jako možnost využití herního enginu a stále má vysokou flexibilitu. Výhodami dále mohou být například velice modulární tvorba aplikací, velmi fotorealistický 3D model, sada předpřipravených assetů pro tvorbu digitálních dvojčat nebo samotná technologie **Universal Scene Description**, díky které je možné výsledné modely exportovat a použít v jiných aplikacích. Z těchto důvodů má platforma velký potenciál právě pro vytvoření systému pro implementaci digitálních dvojčat.

Platforma **NVIDIA Omniverse** má bezesporu obrovský potenciál. Zadavatel práce - společnost **AIMTEC a. s.** měl zájem tuto platformu prozkoumat a zjistit její využitelnost pro tvorbu digitálních dvojčat skladů. To je důvod, proč byla pro implementaci zvolena tato platforma.

Universal Scene Description

4

Vzhledem k tomu, že jsme se rozhodli pro implementaci systému pomocí frameworku **NVIDIA Omniverse**, který využívá technologii **Universal Scene Description**, tato kapitola je věnována podrobnému zkoumání právě této technologie.

Universal Scene Description (zkráceně **USD**) je veřejně dostupný software, který umožňuje robustní a škálovatelný přenos a úpravy libovolných 3D scén, které mohou být složené z různých prvků.

USD poskytuje prostředky pro výměnu elementárních prvků nebo animací. Oproti jiným nástrojům pro výměnu dat však také nabízí možnosti skládat a organizovat libovolný počet prvků do virtuálních sestav, scén a světů. Tyto prvky lze přenášet mezi různými aplikacemi a nedestruktivně je editovat pomocí jednotného a konzistentního *API*. **USD** přináší bohatý soubor nástrojů pro čtení, zápis, úpravu a rychlé zobrazení 3D geometrie, stínování, osvětlení, fyziky a rostoucí počet dalších oblastí spojených s počítačovou grafikou. [12]

4.1 Stručná historie

USD bylo původně vyvinuto společností **Pixar Animation Studios** za účelem zdokonalení technologie pro tvorbu animovaných filmů. [12]

V roce 2016 bylo společností převedeno na open source projekt a byl tak umožněn volný přístup k jeho zdrojovému kódu a rozvoj ze strany komunity. [13]

V srpnu roku 2023 oznámily splečnosti **Pixar**, **Adobe**, **Apple**, **Autodesk** a **NVIDIA**, společně s **Joint Development Foundation**, částí **Linux Foundation**, alianci pro **OpenUSD** (**AOUSD**), za účelem podpořit standardizaci a vývoj technologie **USD**. [14]

V prosinci stejného roku se k této alianci přidalo 12 dalších členů včetně například společností **Meta**, **Unity** nebo **Epic Games**. [15]

4.2 Základní pojmy a funkce

USD přináší mnoho pojmů a konceptů, které jsou hlavní podstatou jeho síly a funkcionality. V této sekci vysvětlíme ty nejdůležitější.

4.2.1 Primitivum (prim)

Primitiva jsou jedním ze základních typů objektu v *USD*. Primitiva jsou kontejnery, které mohou obsahovat další primitiva a vlastnosti (properties) a tím tvořit hierarchii scény. Jak primitiva, tak i vlastnosti mají určené pořadí.[16]

4.2.2 Vlastnost (property)

Vlastnosti jsou druhým základním typem v *USD*. Zatímco primitiva slouží k organizaci scény, vlastnosti obsahují samotná data. Existují dva typy - atributy (attributes) a vztahy (relationships). [16]

4.2.2.1 Atribut (attribute)

Atributy jsou běžnějším z typů vlastností. Jednoduše řečeno, atribut je proměnná, která v daném okamžiku má konkrétní hodnotu nebo je prázdná. Navíc musí být předem definovaný typ hodnoty, který musí být jeden z výčtu typů, které *USD* poskytuje. [16]

4.2.2.2 Vztah (relationship)

Vztah je jednoduše řečeno ukazatel na jiný objekt (objekty), kterým může být primitivum, atribut nebo vztah. Jeden vztah může mít jako cíl více objektů. Na rozdíl od atributů, nemůže být jejich hodnota proměnná v čase. [17]

4.2.3 Vrstva (layer)

Vrstvy jsou dalšími z kontejnerů v *USD*. Vrsvty obsahují tzv. *PrimSpecs*, které popisují vlastnosti primitiv. Každá vrstva má identifikátor, který dovoluje vytvářet reference z jiných vrstev. Vrstvy mohou být také vnořené a lze je skládat a tvořit tzv. *LayerStack*, který obsahuje všechny podvrstvy určité vrstvy seřazené plus vrstvu samotnou jako první a nejsilnější. [16]

4.2.4 Scéna (stage)

Scény jsou vnějším kontejnerem v *USD*. Scéna může obsahovat více vrstev a vzniká jejich kompozicí. Každá scéna má navíc definovanou kořenovou vrstvu (root layer), která popisuje postup, jak tuto kompozici realizovat.

Scény je dále možno procházet (traversal) a zpracovávat jednotlivá primitiva po jednom nebo paralelně, čehož se využívá například při renderování. [16, 18]

4.2.5 Názor (opinion)

Názory představují základní prvky, které se zapojují do procesu vyhodnocování hodnot vlastností. Každá změna vlastnosti primitiva znamená pouze vyjádření názoru na tuto vlastnost v určité vrstvě. V rámci scény vytvořené kompozicí pak může být výsledná hodnota ovlivněna více názory z různých vrstev. [16]

4.2.6 Cesta (path)

Cesty slouží k popisu umístění objektu ve scéně. Můžou být absolutní či relativní. Relativní cesty jsou relativní od primitiva, které je obsahuje. Absolutní cesta začíná / a popisuje cestu od kořenu scény. Cesty používají dva primární typy separátorů. Separátor / slouží k určení potomka. Pomocí separátoru . ("tečka") lze pak popisovat vlastnosti. Na začátku relativních cest lze ještě použít .. ("dvě tečky") k určení předka primitiva. [19]

4.2.7 Třída (class)

USD dovoluje definovat třídy primitiv. Třídy jsou abstraktní, což znamená, že budou standardně přeskočeny při procházení scén. Třídy lze pak dále využít pomocí referencí a dědění viz sekce 4.2.8.

4.2.8 Kompoziční oblouk (composition arc)

Kompoziční oblouky jsou operátory, které dovolují tvořit komplexní kompozice z mnoha vrstev, které kombinují základní popis scény a přepisy. Každý z nich má cíl/cíle, kterými můžou být primitiva, vztahy, nebo kombinace obojího. [16]

4.2.8.1 Podvrstva (sublayer)

Podvrstvy jsou kompozičním obloukem, který slouží k skládání vrstev a tvorbě tzv. LayerStacků zmíněných již v sekci 4.2.3.

4.2.8.2 Reference

Reference jsou mechanismem, který umožňuje vkládat obsah z jedné vrstvy či scény do jiné. Vložený obsah je navíc možno ve scéně dále upravovat (přidávat názory), aniž by tím byla změněna původní verze.

Výhodou je, že pokud scéna obsahuje více referencí na stejný objekt, je objekt v paměti uchováván pouze jednou. Reference umožňují lepší organizaci projektů a snižují redundanci dat. [20]

4.2.8.3 Payload

Payloady jsou speciálním typem referencí. Chovají se víceméně stejně jako reference, jediným rozdílem je, že jejich načtení je volitelné. [16]

4.2.8.4 Sady variant (variant sets)

Sady variant dovolují tvůrci vytvořit množinu alternativ, mezi kterými pak lze snadno přepínat a dále upravovat. Jednoduše by se sady variant daly popsat jako přepínatelné reference. [16]

4.2.8.5 Dědičnost (inherit)

Dědičnost adresuje problém nedestruktivní editace celé třídy objektů. U primitiv můžeme definovat, kterou třídu dědí. Díky tomu pak zdědíme všechny názory definované ve třídě, podobně jako je tomu u referencí.

Rozdíl od referencí získáme za situace, kdy definujeme v nižší vrstvě primitivum s referencí na třídu/odděděné od třídy a pak ve vyšší vrstvě třídu změníme (přidáme nový názor). Pokud se jedná o referenci, změny se ve výsledku neprojeví. Pokud se ale jedná o dědičnost, nové změny se projeví. [16]

4.2.8.6 Specializace (specialize)

Specializace jsou kompozičním obloukem, který se chová velice podobně jako dědičnost. Klíčovým rozdílem mezi dědičností a specializací je síla názorů. Názory definované dědičností jsou vždy silnější, než názory definované ve zděděné třídě. U specializací je tomu naopak. Toto má za následek odlišné chování ve specifických situacích. [16]

4.2.9 Flattening

V některých případech může být užitečné odstranit ze scény její složitou kompozici. Přesně k tomuto účelu slouží flattening, který ze složité scény umožňuje vytvořit jedinou vrstvu bez kompozičních oblouků. [16]

4.2.10 Shrnutí

Celkově tedy při práci s *USD* pracujeme se scénami do kterých přidáváme primitiva. Primitivům lze dále přidávat další primitiva jako potomky, či vlastnosti (atributy a

vztahy). K adresování objektů v rámci scény slouží cesty. Scény můžeme dále členit do vrstev, což umožňuje přepisovat vlastnosti primitiv pomocí názorů. Pro složitější kompozice scén pak slouží kompoziční oblouky.

4.3 USD soubory

V rámci práce s *USD* se data ukládají do *USD* souborů. *USD* v základu podporuje několik formátů souborů, z nichž každý má své vlastnosti a využití. Formáty souborů jsou následující:

4.3.1 .usda (USD ASCII)

Soubory tohoto formátu jsou textové soubory v *ASCII* kódování. Výhodou tohoto formátu je, že jsou soubory lidsky čitelné. Nevýhodou pak je, že při načítání dat ze souboru je potřeba nejprve celý soubor přečíst, parsovat a načíst do paměti a pak až je s ním možno dále pracovat. [16]

4.3.2 .usdc (USD Crate)

Formát *USD Crate* je vlastním binárním formátem *USD*. Soubor tohoto formátu je binární soubor, který je optimalizovaný. Hlavní výhodou je, že při načítání dat není nutné soubor číst celý, ale je možné přečíst jen malý index obsahu souboru při otevření a odložit přístup k velkým datům až do chvíle, kdy jsou explicitně zapotřebí.

Kromě velmi malých souborů (do několika set kB), jsou soubory tohoto typu mnohem kompaktnější, a tak je efektivnější ve srovnání s *ASCII* formátem. [16]

4.3.3 .usd

Jedná se o speciální formát, kde jeho soubory můžou být buď textové v *ASCII* kódování, nebo binární *Crate* soubory. [16]

4.3.4 .usdz (USD ZIP)

Soubor formátu *USD ZIP* je bezkompresní, nešifrovaný zip archiv. Tento archiv kromě souborů typu *USD*, *USDA* a *USDC* může obsahovat také obrázky/textury ve formátech *PNG*, *JPEG*, *EXR* a zvuk ve formátech *M4A*, *MP3* nebo *WAV*. Formát slouží hlavně pro balení a distribuci scén v jediném archivu. [21]

4.4 Hydra

Hydra je framework pro renderování scén, který je součástí **USD**. Umožňuje různým renderovacím enginům (např. **Pixar Storm** nebo **RenderMan**) interagovat s daty uloženými ve formátu *USD* a zobrazovat scény v reálném čase.

Framework byl původně vytvořen pouze jako renderovací engine, který sloužil k zobrazování *USD* scén pomocí *OpenGL*. Postupem času, byly ale obě tyto složky abstrahovány a nyní je to open source framework, který slouží k transportu dat scén rendererům. [22]

NVIDIA Omniverse — 5

NVIDIA Omniverse je digitální platforma, která slouží pro vývoj 3D postupů a aplikací založených na technologii **Universal Scene Description**. Dále může také sloužit pro virtuální spolupráci a simulace. [23]

5.1 Základní komponenty

NVIDIA Omniverse je designovaný pro maximální flexibilitu a škálovatelnost a skládá se z pěti základních komponent. [24]

5.1.1 Omniverse Nucleus

Omniverse Nucleus je kolaborativní a databázový engine **Omniverse**. Jeho významné funkce zahrnují technologie jako *OmniLive*, která umožňuje živé sdílení a spolupráci v reálném čase, *Atomic Checkpoints* pro správu verzí a možnost nastavení práv pro zabezpečení přístupu k datům.

Existují dvě možnosti distribuce tohoto enginu. První je **Enterprise Nucleus Server**, který lze nasadit buď on-premise nebo v cloudu. **Nucleus Workstation** umožňuje nasazení **Nucleus** serveru lokálně a primárně slouží jako prostředek pro testování a zkoušení funkcí v menším měřítku. [25]

5.1.2 Omniverse Connect

Omniverse Connect slouží k napojení aplikací na platformu **Omniverse**. Ve spojení s **Omniverse Nucleus** umožňuje týmovou spolupráci v reálném čase, kdy může jeden či více uživatelů za použití jedné či více aplikací spolupracovat na stejném projektu.

Pro komunikaci s platformou **Omniverse** potřebuje aplikace konektor. V současné době již existují konektory pro širokou škálu aplikací, včetně například **Unity**, **Unreal Engine**, **Autodesk Maya**, **Blender**.

Díky **Omniverse Connect** pak může například na jedné scéně jeden uživatel vytvářet kompletní virtuální prostředí v **Unreal Engine**, zatímco druhý upravuje použité modely v **Blenderu**. Tyto změny jsou následně viditelné u obou účastníků spolupráce

téměř okamžitě. Ukázka spolupráce více uživatelů využitím technologie **Omniverse Connect** se nachází na obrázku 5.1. [26]

Obrázek 5.1: Spolupráce více uživatelů využitím **Omniverse Connect** [27]

5.1.3 Omniverse Kit

Omniverse Kit je sada pro vývoj softwaru (SDK), která slouží k podpoře vývojářů a tvůrců obsahu v rámci platformy **NVIDIA Omniverse**. Sada poskytuje uživatelům prostředky pro tvorbu vlastních aplikací, rozšíření stávajících funkcí a integraci s existujícími nástroji.

Omniverse Kit spojuje několik důležitých komponent **Omniverse** a ke každé dává k dispozici *API*, včetně modulů pro správu renderování, manipulaci s *USD*, reagování na vstupy uživatele a mnohých dalších. Tímto způsobem poskytuje tvůrcům a vývojářům flexibilitu a možnost plně přizpůsobit **Omniverse** jejich specifickým potřebám a projektovým požadavkům. [28]

5.1.3.1 Skriptování

Jedna z komponent **Kitu** je také interpret jazyka *Python* a systém pro skriptování. Tyto funkce se používají hlavně pro tvorbu extensionů (viz sekce 5.2), mohou být ale také využity pro psaní jednoduchých skriptů, které pak lze vykonat pomocí příkazové řádky nebo *API*. [29]

5.1.4 Omniverse RTX Renderer

Omniverse RTX Renderer je škálovatelný fotorealistický renderer, který je založen na frameworku Hydra (viz sekce 4.4). Je navržen tak, aby efektivně využíval výpočetní sílu grafických karet **NVIDIA RTX**.

Může pracovat ve dvou režimech *RTX - Real-Time* a *RTX - Interactive (Path Tracing)*. V režimu *RTX - Real-Time* je zaměřen na okamžitou vizualizaci, která je vhodná pro tvorbu a úpravy scén v reálném čase. Naopak režim *RTX - Interactive (Path Tracing)* se soustředí na detailní a fotorealistické renderování pro dosažení maximální vizuální kvality. [30, 31]

5.1.5 Omniverse Simulation

Součástí **Omniverse** je i sada technologií pro simulace, například open-source fyzikální simulátor **NVIDIA PhysX**, který se často využívá pro počítačové hry. [32]

5.2 Omniverse Extensions

Omniverse Extensions jsou základní stavební bloky, které se používají pro tvorbu aplikací. Jedná se o rozšíření, která slouží k obohacení a rozšíření funkcí platformy **NVIDIA Omniverse**. Dovolují přidat nové prvky a nástroje do základního prostředí **Omniverse** tak, aby lépe vyhovovalo specifickým potřebám uživatelů a projektů.

5.2.1 Struktura extensionu

Extension je v nejjednodušší formě pouze adresářovou složkou s konfiguračním souborem `extension.toml`. Systém pro správu extensionů při nalezení extensionu určí, zda je extension zapnutý. Pokud ano, přečte tento soubor a vykoná, co soubor určuje - načte python moduly, jiné extensiony, nastavení a další.

Také jsou podporovány extensiony, které nejsou tvořeny složkou, ale pouze souborem `extension.toml`. [33]

5.2.2 Závislosti extensionů

Extensiony můžou dále také definovat závislosti na jiných extensionech, což jim umožňuje používat jejich aplikační rozhraní. V praxi to pak dovoluje v kódu importovat a používat třídy z jiných extensionů.

Tyto závislosti se definují v souboru `extension.toml` v sekci `[[dependencies]]`. Závislosti pak tvoří orientovaný graf, který nesmí být cyklický, jinak extensiony nenastartují. [33]

5.2.3 Třída omni.ext.IExt

Třída `omni.ext.IExt` je speciální v tom, že při startu extensionu Omniverse automaticky vyhledá všechny třídy odděděné od této třídy, instancuje je a zavolá jejich metodu `on_startup()`. Podobně pak při vypnutí extensionu volá metodu `on_shutdown()`. Pokud tedy od této třídy oddědíme, získáme možnost reagovat v kódu na tyto události. [33]

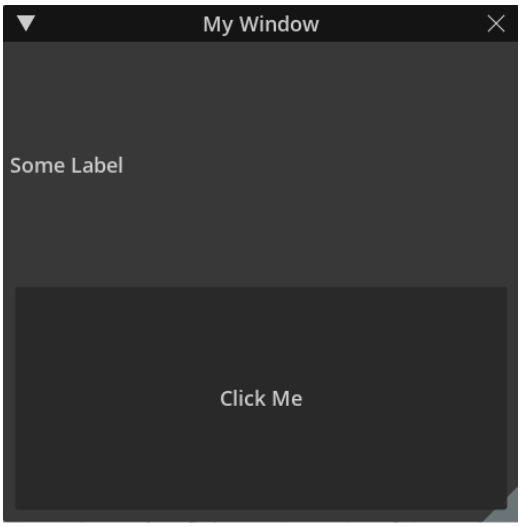
5.2.4 Tvorba UI

K tvorbě uživatelského rozhraní pro extensiony je zapotřebí extension `omni.ui`, který je nutné definovat jako závislost. Uživatelské rozhraní se pak definuje pomocí kódu.

Základem je okno extensionu reprezentované třídou `omni.ui.Window`. Vytvořením instance této třídy vznikne okno v aplikaci, se kterým uživatel může libovolně v rámci aplikace pohybovat nebo ho někam ukotvit.

Další důležitou částí UI jsou frames reprezentované třídou `omni.ui.Frame`. Frames jsou kontejnery, které můžou jako potomka obsahovat vždy pouze jeden UI prvek. Pro layout prvků lze dále použít další typ kontejnerů - vertikální a horizontální stack (`omni.ui.VStack` a `omni.ui.HStack`), které umožňují skládání prvků vedle sebe a pod sebe.

Dále jsou k dispozici běžně známé UI prvky jako tlačítka (`omni.ui.Button`), popisky (`omni.ui.Label`), obrázky (`omni.ui.Image`) a mnoho dalších. Ukázka jednoduchého uživatelského rozhraní se nachází na obrázku 5.2. [34, 35]



Obrázek 5.2: Ukázka uživatelského rozhraní v **NVIDIA Omniverse**

5.3 Aplikace

V rámci frameworku **NVIDIA Omniverse** lze také vytvářet samostatné aplikace. Jak již bylo zmíněno v sekci 5.1.3, aplikace pro **Omniverse** se vytváří pomocí **Omniverse Kit**u. Aplikace je tvořena předem definovanou množinou extensionů, které dohromady poskytují požadovanou funkcionalitu. Pro tvorbu lze použít jakýkoliv počet extensionů vytvořený buď námi, či **Omniverse** komunitou nebo přímo společností **NVIDIA**.

Každá *Kit* aplikace musí mít *Kit* soubor, který ji definuje. Soubor má koncovku `.kit` (což je pouze přejmenovaný `.toml`) a má velice podobnou strukturu jako soubor `extension.toml`, který definuje extension. V tomto souboru se specifikují použité extensiony (v sekci `[dependencies]`) a případná další nastavení, která se mají použít (v sekci `[settings]`).

Výsledné uživatelské rozhraní aplikace je tvořeno okny uživatelských rozhraní použitých extensionů. Navíc lze ještě definovat vlastní layout pro výchozí rozmístění oken jednotlivých extensionů. [36, 37]

5.3.1 Šablona pro tvorbu Kit aplikací

Společnost Nvidia dává také volně k dispozici šablonu *Omniverse Kit App Template*, která je vhodným výchozím bodem pro tvorbu *Kit* aplikací. Tato šablona ukazuje na příkladu několika verzí jednoduché *Kit* aplikace tvorbu těchto aplikací, nastavení extensionů, které používají, jejich build a balení. [38]

5.3.2 Významné Omniverse aplikace

V rámci **Omniverse** platformy jsou také k dispozici již vytvořené aplikace od společnosti **NVIDIA** (tzv. *foundation apps*). Tyto aplikace lze využít pro specifické účely, případně jejich funkcionalitu rozšířit pomocí vlastních extensionů. Dále jsou v této sekci stručně popsány ty nejdůležitější z nich.

5.3.2.1 USD Presenter

Omniverse USD Presenter (dříve **USDView**) je jednoduchá aplikace na prohlížení 3D projektů založených na *USD*. Kromě interaktivního prohlížení umožňuje scény například i anotovat, měřit v nich vzdálenosti nebo pořizovat snímky a videa. [39]

5.3.2.2 USD Composer

Omniverse USD Composer (dříve **USDCreate**) je aplikace, která umožňuje uživatelům stavět, simulovat, a renderovat rozsáhlé *USD* scény. [40]

5.3.2.3 Code

Omniverse Code je aplikace, která slouží pro vývojáře k vývoji extensionů. Funkcemi je hodně podobná **USD Composer**u, ale navíc přináší funkce jako například editor skriptů, nebo propojení s editorem kódu **Visual Studio Code** pro debugování. [41]

5.4 Omniverse Launcher

Omniverse Launcher představuje vstupní bod platformy **NVIDIA Omniverse**. Jde o aplikaci, která slouží k stahování a aktualizaci **Omniverse** aplikací a konektorů.

Launcher dále poskytuje například funkce pro procházení obsahu uloženého na **Nucleus** serveru nebo přehledný katalog učebních materiálů a dokumentace pro seznámení se s funkcemi celé platformy. [42]

5.5 Předpřipravené assety

Součástí **NVIDIA Omniverse** jsou také předpřipravené assety, které je možné použít ve vlastních projektech. Tyto assety zahrnují 3D modely, animace, materiály, světla a mnohé další.

Jednou ze sad předpřipravených assetů je navíc sada pro tvorbu digitálních dvojčat skladů, která obsahuje různé modely regálů, krabic, technologií a mnoho dalších věcí. Ukázka scény využívající těchto assetů se nachází na obrázku 5.3. [43]

Obrázek 5.3: Scéna využívající assety pro tvorbu digitálních dvojčat skladů [44]

5.6 Možnosti výstupu

Výstupy projektu vytvořeného v **NVIDIA Omniverse** můžou být různé od renderování statických obrázků, videí přes uložení scény ve formátu *USD* až po samostatně spustitelnou aplikaci.

5.6.1 Renderování obrázků, videa

Platforma **Omniverse** poskytuje nástroje pro tvorbu poutavých obrázků a videí. Tato funkcionalita je dostupná v aplikaci **Omniverse USD Presenter**, a to konkrétně v nástroji *Movie Capture* určeném pro snímání a nahrávání videa. [45]

5.6.2 Export USD

Dalším možným výstupem z aplikace je model ve formátu *USD*, který může být použit v jiných aplikacích.

5.6.3 Export do jiných formátů

Další možností je pak export scény nebo jejích částí do různých jiných formátů. Exporty umožňuje extension *Asset Converter*, který podporuje formáty *OBJ*, *FBX* a *glTF*. [46]

5.6.4 Aplikace

Hlavním výstupem jsou pak samotné aplikace, které mohou být distribuovány několika způsoby.

5.6.4.1 Fat package

První možností distribuce je tzv. *Fat package*. Součástí balíku aplikace je pak vše co aplikace potřebuje a pro instalaci již není nic dalšího potřeba stahovat. Hodí se například pro instalaci na místa s omezeným přístupem k internetu. [47]

5.6.4.2 Thin package

V opačné situaci se hodí možnost distribuce pomocí tzv. *Thin package*. Balík aplikace při využití tohoto způsobu obsahuje pouze nutné minimum aby byla instalace možná. Zbylé závislosti si pak stahuje při instalaci. [47]

5.6.4.3 Launcher package

Poslední možností je tzv. *Launcher package*. Balík vytvořený při využití této možnosti může být *Fat package* i *Thin package* a výsledná aplikace se pak instaluje a spouští pomocí **Omniverse Launcher**u. [47]

Požadavky na funkcionalitu systému

6

Tato kapitola pojednává o požadovaných funkcionalitách, které by měl vytvořený systém poskytovat. Systémem je zde i dále v textu myšlen systém pro tvorbu a prohlížení digitálních dvojčat skladů vytvořený ve frameworku **NVIDIA Omniverse**, který by měl vzniknout jako výsledek této práce.

Požadavky na nový systém jsme specifikovali s konzultantem ze společnosti **AIMTEC a. s.** Vycházeli jsme ze stávající funkcionality aktuálně používaného systému **GDTW**, kterou požadoval zachovat. Souhrn všech těchto požadavků dostatečně ověří vhodnost platformy **NVIDIA Omniverse** pro vytváření digitálních dvojčat skladů.

Hlavní funkcionality, které musí systém podporovat jsou dvě - tvorba digitálního dvojčete a jeho následné prohlížení. Jednotlivá digitální dvojčata vytvořená za pomocí systému budou v textu dále označována jako projekty.

6.1 Tvorba digitálního dvojčete skladu

První z hlavních funkcionalit systému je tvorba digitálního dvojčete. Systém musí mít možnost založení nového projektu a uložení právě editovaného projektu. Dále musí být možnost otevřít a editovat již existující projekt.

Předpokládá se, že projekty vytváří vývojář, který je se systémem pro tvorbu a zobrazování digitálních dvojčat, jeho možnostmi a funkcionalitou seznámený. Nicméně, i přes to, by neměla tvorba projektu být příliš složitá a měla by být relativně intuitivní.

6.1.1 Editace 3D modelu

V režimu editace musí být možnost vidět 3D scénu s modelem skladu. Dále musí být možnost vidět a procházet hierarchii *USD* scény a jednotlivých objektů v této scéně.

USD objekty musí jít přidávat, odstraňovat, posouvat, otáčet, škálovat a nesmí zde chybět možnost upravovat jejich *USD* atributy (např. kód skladové pozice).

6.1.2 Generování skladové hierarchie

Kromě editace a ručního vytváření *USD* objektů ve scéně, musí mít systém možnost generování skladové hierarchie podle dat vyčítaných z externího systému. Do modelu by měly jít snadno pomocí uživatelského rozhraní generovat řady regálů, které se dále skládají z sloupců a skladových pozic. Konečná struktura vygenerované hierarchie by měla odpovídat struktuře dat vyčítaných z externího systému.

Tato struktura je konkrétně pro systém **Aimtec DCIx** následující:

- Sklad - Nejvyšší úroveň hierarchie, která zahrnuje celé skladové zařízení nebo budovu
- Řada - Úroveň pod skladem, která rozděluje skladovací prostor na menší sekce oddělené uličkami nebo jinými strukturami
- Sloupec - vertikální část řady obsahující několik skladových pozic nad sebou
- Patro - horizontální část řady obsahující několik skladových pozic vedle sebe
- Skladová pozice - konkrétní místo pro uložení zásob jednoznačně určené sloupcem a patrem

6.1.3 Nastavení projektu

Dalé musí systém umožnit v režimu editace projektu nastavovat a měnit nastavení projektu. Mezi tyto nastavení bude patřit například nastavení komunikace s externím systémem pro načítání dat, nebo také nastavení konfigurace statistických veličin (viz sekce 6.2.2).

6.1.4 Design

Do modelu skladu bude také možné přidávat 3D objekty, které zde budou za účelem realističtějšího vzhledu scény. Tyto objekty mohou tvořit například stěny, či technologie nebo jiné dekorace. Pro tyto účely bude vhodné využití předpřipravených assetů pro tvorbu digitálních dvojčat součástí **Omniverse** (viz sekce 5.5). Systém by měl mít v režimu editace možnost nabízené assety procházet a do modelu je snadno přidávat.

6.2 Prohlížení digitálního dvojčete skladu

Druhou hlavní funkcionalitou systému je prohlížení již vytvořených projektů digitálních dvojčat skladů. Systém musí mít možnost načíst hotový projekt digitálního dvojčete a prohlížet si ho jako 3D model.

Zde se předpokládá uživatel, který se systémem nemusí být seznámený a tak musí být kladen ještě větší důraz na intuitivnost práce se systémem. Navíc v režimu prohlížení nesmí systém umožnit v projektu dělat žádné změny. Nezkušený uživatel tak nebude mít možnost projekt nevratně poškodit.

6.2.1 Zobrazování balení

Systém by měl v režimu prohlížení mít možnost načítat zboží z externího systému a zobrazovat je ve 3D modelu digitálního dvojčete. Zboží je ve skladu ukládáno ve formě různých balení (krabice, boxy, palety atd.) a takto bude reprezentováno i v rámci digitálního dvojčete. V rámci textu bude dále pro označení těchto skladovacích jednotek používán pojem balení.

Balení by se měla zobrazovat jako 3D objekty na skladových pozicích, na kterých se momentálně v externím systému nachází. Dále by neměla chybět možnost zjistit detailnější informace o konkrétním balení. Tyto informace by měly zahrnovat minimálně skladovou pozici, na které se balení nachází, číslo etikety balení a informace o zboží, které balení obsahuje.

6.2.2 Statistické veličiny

Dále by do modelu měly jít různými zobrazovacími metodami zobrazovat různé statistické veličiny. Data pro tyto statistické veličiny (dále jen statistiky) se budou načítat z externího systému a uživatel by měl mít k dispozici uživatelské rozhraní, kterým si bude zobrazování jednotlivých statistik ovládat. V určitou chvíli by měla být v modelu zobrazena vždy pouze jedna statistika, jinak by vizualizace statistik nebyla přehledná.

Tyto statistiky by měly být modulární a přidání nové statistiky by ideálně nemělo vyžadovat změny v kódu. Samozřejmě s předpokladem, že statistika použije již implementovanou zobrazovací metodu.

6.2.3 Procházení skladové hierarchie

Systém v režimu prohlížení by měl podporovat procházení skladové hierarchie vygenerované podle sekce 6.1.2. Tato hierarchie by měla jít zobrazit a interaktivně procházet za pomocí uživatelského rozhraní s určitými vizuálními změnami ve scéně tak, aby uživatel poznal, který objekt zrovna prohlíží.

6.2.4 Export

Systém by měl umožňovat exportovat model digitálního dvojčete do formátu *USD* tak, aby šel použít i v jiných aplikacích, které tento formát podporují.

Návrh systému 7

Tato kapitola popisuje způsob, jak jsou v rámci systému řešeny jednotlivé požadavky popsané v kapitole 6. V kapitole je popsána vysokoúrovňová architektura systému, struktura projektu a chování jednotlivých částí aplikací.

7.1 Architektura

Architektura systému je tvořena z aplikací a extensionů a využívá možností definování závislostí mezi těmito entitami.

7.1.1 Aplikace

Systém se skládá ze dvou aplikací - *Editor* a *Viewer*, které fungují nezávisle na sobě. Obě aplikace jsou vytvořeny pomocí frameworku **Omniverse Kit**.

7.1.1.1 Editor

Editor (`dtw.editor.app`) je první aplikací ze dvou částí systému. Tato aplikace slouží k vytváření a editaci digitálních dvojčat. Slouží tedy výhradně pro vývojáře, který její pomocí digitální dvojče navrhne, vytvoří a nastaví vše potřebné.

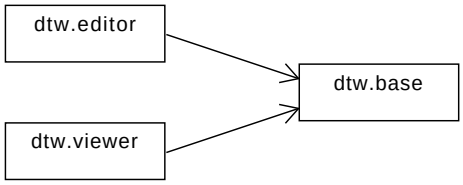
7.1.1.2 Viewer

Druhou částí systému je aplikace *Viewer* (`dtw.editor.app`), která slouží k prohlížení vytvořených digitálních dvojčat. Kromě prohlížení 3D modelu, umožňuje také do modelu promítat statistické veličiny. S výjimkou úpravy konfigurace barev nedovoluje v digitálním dvojčeti dělat žádné persistentní změny.

7.1.2 Extensiony

Systém se skládá z tří vlastních extensionů, kterými jsou `dtw.base`, `dtw.editor` a `dtw.viewer`.

Extensiony `dtw.editor` a `dtw.viewer` jsou obě závilsé na extensionu `dtw.base`, ale nejsou závislé na sobě navzájem. V praxi to ale pak znamená, že moduly těchto extensionů nesmí mít mezi sebou žádné závislosti a jakákoliv společná funkcionalita musí být implementována v extensionu `dtw.base`. Schéma závislostí extensionů je uvedeno na obrázku 7.1.



Obrázek 7.1: Závislosti extensionů

7.1.2.1 Base

Extension `dtw.base` slouží k funkcím, které jsou společné pro obě aplikace *Editor* a *Viewer*. Mezi jeho funkcionalitu patří načítání dat z externího systému, práce s prvky skladové hierarchie, nebo například práce s nastavením projektu.

Kromě toho také poskytuje užitečné utility, které zjednodušují práci s některými prvky frameworku, jako například práci s primitivy, které zbylé extensiony využívají.

7.1.2.2 Editor

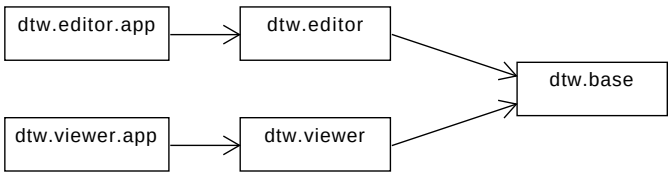
Extension `dtw.editor` slouží k funkcionalitám, které jsou potřebné pouze pro aplikaci *Editor*. Mezi funkcionality tohoto extesionu patří inicializace nového projektu, nastavení konfigurace komunikace s externím systémem, konfigurace statistických veličin a generování skladové hierarchie.

7.1.2.3 Viewer

Podobně pak extension `dtw.viewer` slouží k funkcionalitám, které jsou potřebné pouze pro aplikaci *Viewer*. Funkcionality tohoto extensionu zahrnují dynamické načítání a zobrazování balení, statistik, interaktivní procházení skladové hierarchie a export.

7.1.3 Závislosti aplikací na extensionech

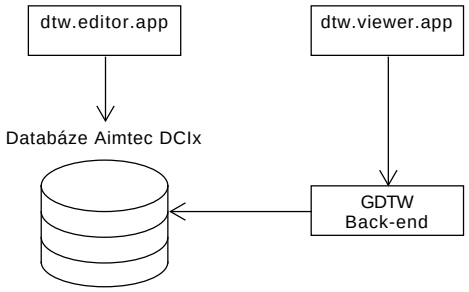
Schéma závislostí **Kit** aplikací je uvedeny na obrázku 7.2. Toto schéma zobrazuje jak závislostí aplikací na vlastních extensionech, tak i závislosti extensionů mezi sebou.



Obrázek 7.2: Závislosti aplikací na extensionech

7.2 Datové zdroje

Systém využívá pro svoji funkčnost data načtená z systému **Aimtec DCIx**, který byl popsán v sekci 2.2.1. Ke komunikaci s tímto systémem využívá dvou různých typů spojení - přímý přístup do databáze systému a komunikaci s *REST API* back-end části aplikace **GDTW**, která je na systém napojená. Schéma této komunikace je uvedeno na obrázku 7.3.



Obrázek 7.3: Schéma komunikace se systémem **Aimtec DCIx**

7.2.1 Databáze

První možností komunikace, kterou systém využívá pro získávání dat je přímý přístup do databáze systému **Aimtec DCIx**. Tato možnost je využita jen pro *Editor* a to konkrétně pro načtení seznamu dostupných skladových pozic.

Důvodem použití přímého přístupu do databáze pro načítání těchto dat je absence této funkcionality ze strany **GDTW** back-endu, který není součástí této práce. Architektura systému by však umožňovala snadné nahrazení přímého přístupu do databáze za přístup přes back-end.

Parametry pro přístup do databáze (databázový server, port, konektor, název databáze, uživatelské jméno a šifrované heslo) lze pro každý projekt nastavit a ukládají se společně s projektem v *USD* souborech.

7.2.2 GDTW back-end

Druhým způsobem, který systém využívá jako zdroj dat je komunikace s *REST API* systému **Generic Warehouse Digital Twin** popsaného v sekci 2.2.2. Z tohoto systému pak načítá všechna data potřebná pro funkce aplikace *Viewer*. Ta zahrnují data aktuálních balení a pak data jednotlivých statistik. Parametry tohoto spojení (URL serveru, uživatelské jméno a šifrované heslo) se stejně jako u komunikace s databází dají pro každý projekt nastavit a jsou opět uloženy s projektem v rámci *USD* souborů.

7.3 Adresářová struktura systému

Systém jako celek tvoří jedna složka (`OmniverseWarehouseDigitalTwin`), ve které se nachází všechny jeho části. Kostra adresářové struktury této systému se nachází na obrázku 7.4. Tato struktura z velké části vychází ze struktury volně přístupné šablony *Omniverse Kit App Template*, která již byla zmíněna v sekci 5.3.1.

```
OmniverseWarehouseDigitalTwin
├── extensions
│   ├── dtw.base
│   ├── dtw.editor
│   └── dtw.viewer
├── apps
│   ├── dtw.editor.app
│   └── dtw.viewer.app
├── assets
├── projects
└── tools
```

Obrázek 7.4: Adresářová struktura systému

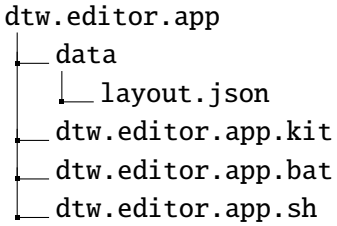
7.3.1 Složka extensions

Složka `extensions` obsahuje jednotlivé vytvořené extensiony, které byly popsány v sekci 7.1.2. Součástí složky každého extensionu je jeho zdrojový kód v jazyce python a metadata, která obsahují mimo jiné také konfigurační soubor `extension.toml`, který byl popsán v sekci 5.2.1.

7.3.2 Složka apps

Ve složce `apps` se nachází data popisující konfiguraci jednotlivých aplikací popsaných v sekci 7.1.1. Složky obou aplikací mají dále stejnou strukturu, která se nachází na obrázku 7.5 (pro příkla uvedena struktura aplikace `dtw.editor.app`). Pro aplikaci je hlavní její konfigurační **Kit** soubor popsaný v sekci 5.3. Dále pak obsahuje

složku `data` obsahující soubor `layout.json`, který popisuje výchozí rozložení oken. Kromě toho obsahuje také skripty v jazycích *batch* (pro *Windows*) a *bash* (pro *Linux*) pro spuštění aplikace.



Obrázek 7.5: Adresářová struktura aplikace

7.3.3 Složka assets

Složka `assets` obsahuje různé assety, které systém používá. Tyto assety budou dále popsány v sekci 7.5.

7.3.4 Složka projects

Složka `projects` slouží pro ukládání jednotlivých projektů. Každý projekt pak má v této složce vlastní podsložku, ve které jsou uloženy všechny jeho soubory.

7.3.5 Složka tools a skripty

Složka `tools` slouží pro build aplikací. Tato složka je převzata z šablony *Omniverse Kit App Template* a obsahuje různé softwarové nástroje a skripty, které se při buildu aplikací využívají.

Kromě kostry uvedené na obrázku 7.4 kořenová složka systému také obsahuje skripty v jazycích *batch* a *bash*, které slouží pro nastavení vývojářského prostředí a build aplikací. Využití skriptů pro build aplikací je dále popsáno v sekci 7.13.

7.4 Struktura projektu

Projekt jako takový tvoří *USD* scéna (projektem je stále myšlena jedna konkrétní instance digitálního dvojčete). Tato scéna má předem definovanou hierarchii a skládá se z několika *USD* vrstev (viz sekce 4.2.3). Tyto vrstvy pak obsahují kromě dat načítaných z externího systému **Aimtec DCIx** všechna data projektu.

7.4.1 Vrstvy

Scéna projektu je rozdělená celkem do čtyř *USD* vrstev. V projektu jsou vždy následující vrstvy: základní, vizuální, konfigurace zobrazení a procházení hierarchie.

7.4.1.1 Základní vrstva

Základní vrstva (`BaseLayer.usd`) je hlavní *USD* vrstvou celého projektu. Tato vrstva obsahuje nastavení projektu (nastavení komunikace s databází a *API* **GDTW** backendu), samotný model skladu a pak zbylé věci, jako například světla.

Součástí modelu skladu je podkladová deska, která může mít nastavenou texturu. Dále je součástí pak samotná hierarchická struktura skladu, která obsahuje modely regálů a informace o skladových pozicích a ostatních prvcích skladové hierarchie. Kromě toho mohou být součástí modelu také objekty, které zde jsou za účelem realističtějšího vzhledu scény.

Dělat změny v této vrstvě je možné pouze pomocí aplikace *Editor*. Při použití v aplikaci *Viewer* se vrstva uzamkne, aby v ní nebyly možné dělat žádné změny a digitální dvojče tak při prohlížení změnit.

7.4.1.2 Vizuální vrstva

Vizuální vrstva (`VisualLayer.usd`) je vrstvou, která se používá pouze v aplikaci *Viewer*. Její účel je jednoduchý - mít možnost zobrazovat do modelu vizualizace, které se ale nebudou společně s modelem ukládat. Vždy při spuštění či ukončení aplikace je tedy obsah této vrstvy vyčištěn. Vrstva je systémem využita pro zobrazování balení a vizualizaci statistik pomocí highlightů (viz sekce 7.9).

7.4.1.3 Vrstva konfigurace zobrazení

Vrstva konfigurace zobrazení (`ViewConfigurationLayer.usd`) je vrstva, která slouží pro ukládání konfigurace, kterou používá aplikace vieweru. Jelikož je základní vrstva při použití vieweru zamčená, není možné do ní tedy zapisovat žádná data a proto není pro toto použití vhodná. Vizuální vstva se neukládá, proto také pro toto využití není vhodná. Pro aplikaci vieweru je tedy zapotřebí ještě třetí vrstva, kam viewer ukládá data své konfigurace.

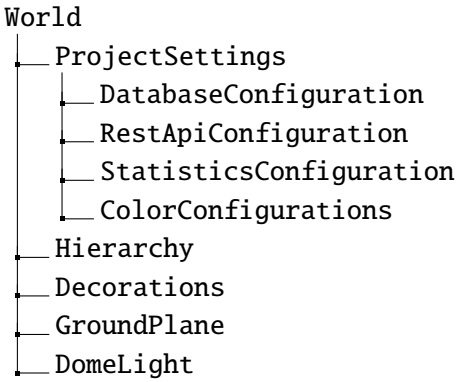
Mezi tato ukládaná data patří hlavně materiály, které jsou použité pro highlighty. Barvu těchto materiálů může uživatel nastavit a je žádoucí, aby tato nastavení byla persistentní - zůstala uložená i po restartu aplikace.

7.4.1.4 Vrstva procházení hierarchie

Vrstva procházení hierarchie (`HierarchyBrowsingLayer.usd`) slouží, jak její název vypovídá, k funkcionalitě procházení skladové hierarchie. Při procházení struktury dochází k častým posunům objektů a tyto změny pozic objektů nechceme do základní, ani žádné jiné vrstvy ukládat. Zároveň pak vyloučení změn do samostatné vrstvy umožní změny snadno zahodit vyčištěním této vrstvy při ukončení procházení.

7.4.2 Hierarchie scény

Projekty mají pevně danou hierarchii *USD* primitiv, které jsou součástí scény. Kostra této hierarchie je uvedena na obrázku 7.6. Veškerá primitiva v rámci scény jsou potomky primitiva `World`, které je kořenovým primitivem scény.



Obrázek 7.6: Hierarchie scény

7.4.2.1 Primitivum ProjectSettings

V rámci potomků primitiva `ProjectSettings` jsou uložena veškerá nastavení projektu. Tato nastavení zahrnují nastavení komunikace se systémem **Aimtec DCIx** - nastavení připojení do databáze pro aplikaci *Editor* a nastavení komunikace s *REST API* **GDTW** back-endu. Nastavení jsou uložena jako *USD* atributy (viz sekce 4.2.2.1) primitiv `DatabaseConfiguration` a `RestApiConfiguration` v rámci základní vrstvy projektu.

Nastavení projektu obsahují také konfiguraci statistik, která může být pro každý projekt specifická. Tato konfigurace je uložena jako *USD* atributy primitiva `StatisticsConfiguration` opět v rámci základní vrstvy.

Dále pak nastavení zahrnují barevné konfigurace statistik. Tato nastavení jsou uložena jako *USD* materiály, které jsou potomky primitiva `ColorConfigurations` a které mají barvy nastavené jako *USD* atributy. Materiály jsou pak dále použity pro vizualizace statistik a jsou uloženy ve vrstvě konfigurace zobrazení z důvodů vysvětlovaných v sekci 7.4.1.

7.4.2.2 Primitivum Hierarchy

V rámci potomků tohoto primitiva je uložena celá hierarchická struktura skladu. Toto primitivum má tedy funkci jistého symbolického kořenu celé této hierarchie. Entity této struktury jsou uloženy každé jako vlastní primitivum s názvem odpovídajícím jejich kódu v systému **Aimtec DCIx**. Vzhledem k tomu, že *USD* ale nepodporuje

některé speciální symboly (například `.` a `:`) v identifikátorech (názvech primitiv či atributů), je nutné tyto symboly něčím nahradit. V tomto případě jsou nahrazeny tečky v názvech vzniklých z kódů podtržítky.

7.4.2.3 Primitivum Decorations

Účelem primitiva `Decorations` je být rodičem všech dekorací či jiných primitiv, která jsou v modelu pouze za účelem designu. Toto primitivum tyto objekty sdružuje tak, aby hierarchie scény zůstávala i při větším počtu těchto designových prvků stále přehledná.

7.4.2.4 Primitivum GroundPlane

Primitivum `GroundPlane` tvoří podkladovou desku, na které se nachází celý model. Deska má dále nastavený materiál s texturou, která je využita pro podkladový plán skladu. Tato textura je uložena v projektových assetech (viz sekce 7.5.2).

7.4.2.5 Primitivum DomeLight

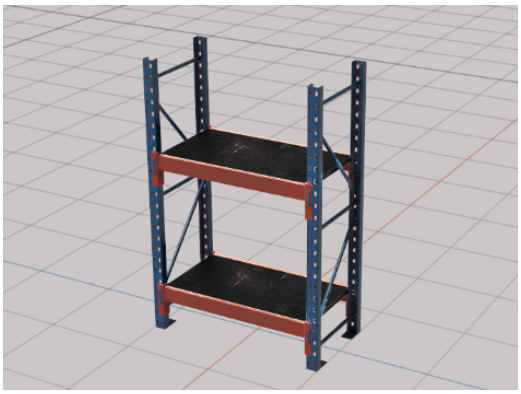
Toto primitivum je pouze světlo, které je systémem automaticky vytvořeno při inicializaci projektu. Poté již nemá žádný speciální význam a je možné ho vyměnit za jakýkoliv jiný typ světla, či více světel. Jedná se o světlo, které osvětluje objekty rovnoměrně ze všech stran.

7.5 Assety

Součástí systému jsou i některé assety, které projekty využívají. Tyto assety se dělí na dva typy a to globální a projektové. Assety mohou tvořit libovolné soubory, které se pak využívají v projektech. Nejčastěji jsou to pak *USD* soubory s objekty, které jsou součástí 3D modelů, či textury.

7.5.1 Globální assety

Globální assety jsou assety společné pro všechny projekty. Jsou uložené ve složce `assets`, která se nachází v kořenové složce `OmniverseWarehouseDigitalTwin`. Patří mezi ně například části regálů, které obsahují objekty ze sady předpřipravených assetů pro tvorbu digitálních dvojčat. Tyto objekty jsou obsaženy jako *USD* reference, které jsou dále upravené, aby vyhovovaly projektům. Ukázka assetu části regálu se nachází na obrázku 7.7.



Obrázek 7.7: Asset části regálu

7.5.2 Projektové assety

Projektové assety jsou assety, které jsou součástí pouze jediného projektu, pro který jsou specifické. Jsou uloženy ve složce `assets`, která se nachází ve složce každého projektu. Mezi projektové assety patří například textura podkladové desky modelu, která je pro každý projekt jiná, ale všechny projekty by ji měly mít.

7.6 Ukládání projektu

Projekt digitálního dvojčete vytvořený pomocí aplikace *Editor* je samozřejmě potřeba ukládat. Pro ukládání je ideální využít možnosti frameworku **Omniverse**, který má intuitivní ukládání *USD* scény již implementované. Jak už bylo zmíněno v sekci 7.4, všechna data spojená s projektem kromě dat načítaných ze systému **Aimtec DCIx** se ukládají v rámci *USD* souborů.

Při ukládání *USD* scény je každá *USD* vrstva uložena zvlášť jako samostatný *USD* soubor. Kořenová vrstva scény (root layer) pak referencuje zbylé vrstvy scény, což zajišťuje jejich načtení při otevření kořenové vrstvy. Jak bylo zmíněno již v sekci 7.4.1, ukládají se pouze základní vrstva a vrstva konfigurace zobrazení. Ostatní vrstvy musí existovat jako prázdné *USD* soubory, protože framework **Omniverse** nepodporuje práci s vrstvami, které nejsou uloženy na disku.

7.7 Generování skladové hierarchie

Aplikace *Editor* umožňuje generování skladové hierarchie podle dat načtených ze systému **Aimtec DCIx**. Tato data se načítají pomocí přímého připojení do databáze systému **Aimtec DCIx** a zahrnují pouze kompletní seznam kódů skladových pozic. Aplikace umožňuje tyto pozice dále filtrovat pomocí regulárních výrazů a pak generovat regály či skladové plochy, které tyto pozice obsahují.

Dále je pak ještě možné zobrazit si informace o tom, kolik skladových pozic bylo celkově načteno z externího systému a kolik jich bylo vyfiltrováno. Pro každou z entit skladové hierarchie informace dále zobrazují, kolik instancí této entity bylo vyfiltrováno a kolik jich již bylo vygenerováno a jsou tak součástí modelu.

Pro funkcionalitu generování skladové hierarchie je vyhrazeno v aplikaci vlastní okno uživatelského rozhraním s názvem *Hierarchy Generator*.

7.7.1 Filtrování pomocí regulárního výrazu

Skladových pozic načtených z externího systému může být relativně velké množství a tak je systém při načtení dovoluje filtrovat pomocí regulárních výrazů. Tento popis využívá tzv. pojmenované skupiny zachycení (named capture group) k oddělení jednotlivých částí kódů, které postupně popisují entity skladové hierarchie. Tyto skupiny jsou `Warehouse`, `Rack`, `Column` a `Floor`. V regulárním výrazu musí být vždy obsaženy všechny tyto skupiny a žádná z nich se nesmí opakovat. Pro pojmenované skupiny je u regulárního výrazu nutné použít syntaxi, kterou podporuje programovací jazyk *Python*.

Využití regulárních výrazů a pojmenovaných skupin navíc dává systému jistou flexibilitu popisu skladových pozic, kterou by v budoucnu bylo možné využít při použití jiného externího systému než **Aimtec DCIx** jakožto datového zdroje.

7.7.1.1 Příklad

Je zvolen následující regulární výraz:

```
^(?P<Warehouse>DTW)\.(?P<Rack>A01)\.(?P<Column>\d{2})\.(?P<Floor>\d{2})$
```

Tento regulání výraz vyfiltruje následující kódy skladových pozic:

- `DTW.A01.01.02`
- `DTW.A01.21.03`
- `DTW.A01.99.99`

Regulární výraz naopak nevyfiltruje kódy:

- `DTW.B01.01.01`
- `DTW.A01.123.123`
- `ABC.A01.01.01`

Například kód skladové pozice `DTW.A01.01.02` je rozparsován následovně:

- Kód skladu: `DTW`

- Kód řady: `A01`
- Kód sloupce: `01`
- Kód patra: `02`

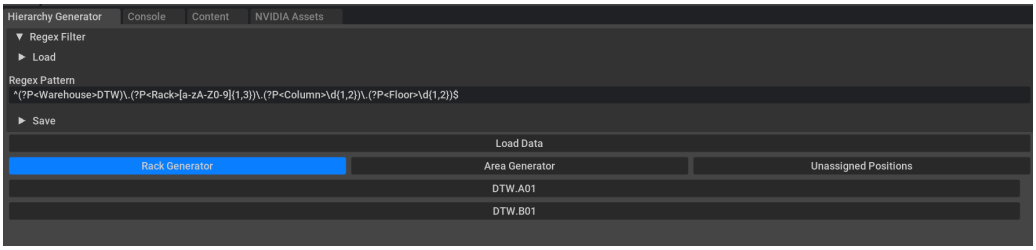
Jedná se tedy o skladovou pozici, která se nachází v prvním sloupci a druhém patře v řadě s kódem `A01`, ve skladu s kódem `DTW`.

7.7.1.2 Ukládání regulárních výrazů

Aplikace také dovoluje uživateli si tyto regulární výrazy ukládat pod zvoleným názvem a pak je zpět načítat pro jejich opětovné použití. Tyto regulární výrazy se ukládají do souboru `rack_generator_regexes.json` ve složce `assets`. Soubor už navíc obsahuje jeden výchozí uložený výraz s názvem `DCIx`, který popisuje obecný kód skladové pozice v systému **Aimtec DCIx**. Tento regulární výraz je zvolen jako výchozí při spuštění aplikace.

7.7.2 Generování regálů

Po vyfiltrování skladových pozic je dále možné generovat regály. Vyfiltrované skladové pozice se seskupují podle kódu skladu a regálu. Tyto skupiny se pak zobrazí uživateli, který zvolením některé z těchto skupin vygeneruje regál, který tyto pozice obsahuje. Tento regál se vygeneruje uprostřed modelu skladu a uživatel ho může posunout do požadované pozice v modelu. Uživatelské rozhraní pro generování regálů se nachází na obrázku 7.8.



Obrázek 7.8: Uživatelské rozhraní generování regálů

Akcí vygenerování se vygenerují veškeré dosud neexistující entity skladové hierarchie jako *USD* primitiva, která se umístí do hierarchie scény pod primitivum `Hierarchy`. Pokud tedy například generujeme regál, ve skladu `ABC`, zatímco již je sklad `ABC` vygenerovaný a obsahuje jiné regály, tento regál se pouze přidá do tohoto skladu a primitivum skladu již se znovu negeneruje.

U takto vygenerovaných primitiv se také nastaví atributy, které dovolují systému entity skladové hierarchie z *USD* scény zpět načítat a při spuštění aplikace editoru už tak není nutné skladové pozice znovu načítat z externího systému.

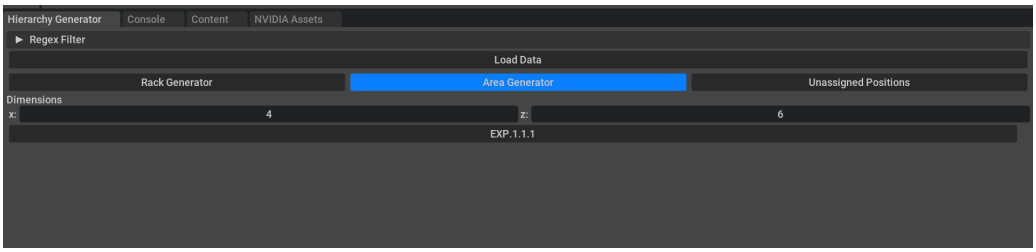
Polohu, rozměry či atributy vygenerovaných primitiv si může uživatel navíc ručně upravit a dosáhnout tak velké flexibility například při generování netradičních konfigurací regálů. Tyto úpravy umožňují extensiony `omni.kit.window.stage` a `omni.kit.property.usd`, které jsou součástí aplikace *Editor*. Ukázka vygenerovaného regálu se nachází na obrázku 7.9.

Obrázek 7.9: Regál

7.7.3 Generování skladových ploch

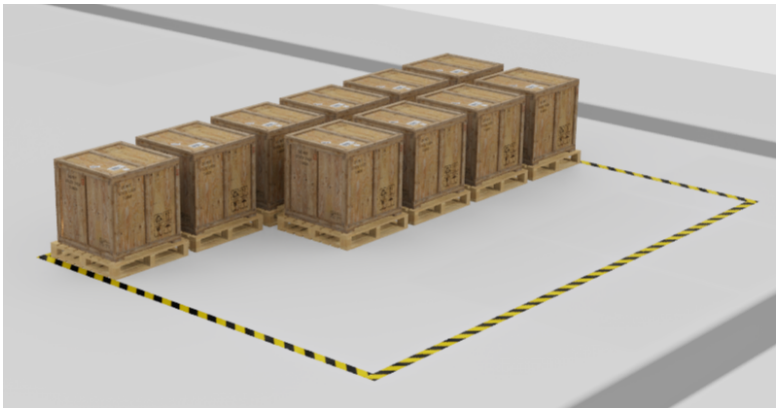
Kromě regálů lze generovat také skladové plochy. Každá skladová plocha je definována pouze pomocí jedné skladové pozice. V režimu generování skladových ploch se tedy v uživatelském rozhraní místo agregovaných skupin skladových pozic kompletní seznam dosud nevygenerovaných skladových pozic, které byly vyfiltrovány. Kromě toho si uživatel může nastavit rozměry této zóny. Uživatelské rozhraní pro generování skladových ploch se nachází na obrázku 7.10.

Po zvolení jedné z vyfiltrovaných možností se pak podobně jako při generování regálů vygeneruje uprostřed modelu skladová plocha. V hierarchii scény se tato plocha vygeneruje jako soustava primitiv, která zahrnuje regál, sloupec i skladovou pozici, aby s ní bylo možné pracovat stejně jako s regály. Pro plochu se ale nevyge-



Obrázek 7.10: Uživatelské rozhraní generování skladových ploch

neruje police, ale místo toho pouze čáry, které ji ohraničují. Ukázka vygenerované skladové plochy se nachází na obrázku 7.11.



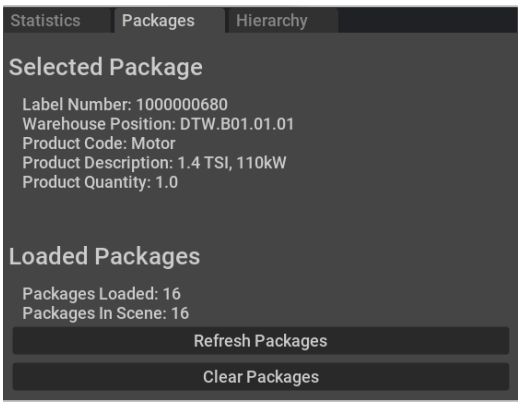
Obrázek 7.11: Skladová plocha

7.8 Zobrazování balení

V rámci aplikace *Viewer* se na skladových pozicích zobrazují balení na základě dat načítaných z *REST API* back-endu systému **GDTW**. Tato balení se zobrazují jako krabice na paletě, které jsou do scény přidány jako *USD* payloady. Tyto payloady pak odkazují na asset uložený v globálních assetech.

Balení jsou načtena a zobrazena již při startu aplikace a uživateli je v rámci uživatelského rozhraní s názvem *Packages* umožněno načtená balení z modelu odstranit nebo provést refresh a balení znovu načíst.

Po zvolení některého z objektů balení v modelu se v tomto uživatelském rozhraní navíc zobrazí data tohoto balení, která zahrnují skladovou pozici, na které se balení nachází, číslo etikety balení a informace o produktu, který balení obsahuje. Uživatel má navíc možnost zvolit tažením myši více balení zároveň a uživatelské rozhraní mu pak pomocí combo-boxu umožňuje z těchto zvolených balení vybrat to, jehož informace chce zobrazit. Uživatelské rozhraní se nachází na obrázku 7.12.



Obrázek 7.12: Uživatelské rozhraní zobrazování balení

7.9 Zvýraznění skladových pozic

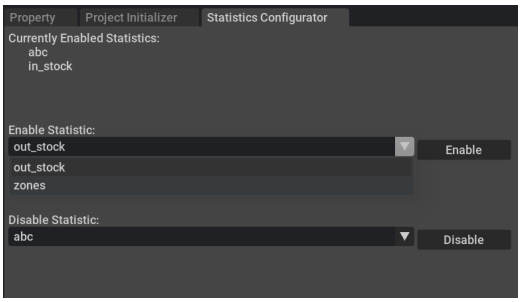
Takzvané highlighty slouží pro označení nebo zvýraznění konkrétních skladových pozic, či jejich skupin. Jedná se o termín, který byl zaveden v rámci systému **GDTW** a připadá mi velice výstižný. Proto jsem se rozhodl toto označení v rámci textu zachovat. V modelu má highlight podobu průhledné barevné kostky, která je umístěná přes skladovou pozici. Díky průhlednosti highlightu je na skladové pozici stále vidět i případné balení.

Highlighty jsou v rámci aplikace *Viewer* využity ke dvěma účelům a to pro statistické veličiny a pro označení zvolené pozice v rámci procházení skladové hierarchie.

7.10 Statistické veličiny

Aplikace *Viewer* umožňuje do modelu zobrazovat některé statistické veličiny, zkráceně statistiky. Statistiky jsou navrženy modulárně tak, aby přidání statistiky bylo co nejjednodušší a bylo možné přidávat nové statistiky i bez změn v kódu. Chování jednotlivých statistik se definuje v `JSON` souborech - definicích statistik. Tyto soubory se nachází ve složce `assets/statistic_definitions` a jejich název je vždy zkratka názvu statistiky.

V aplikaci *Editor* si uživatel pro každý projekt vybírá dostupné statistiky. Aplikace mu v rámci uživatelského rozhraní s názvem *Statistics Configurator* zobrazí seznam statistik definovaných ve složce `assets/statistic_definitions`. Uživatel si pak zvolí ty, které chce pro projekt zpřístupnit.



Obrázek 7.13: Uživatelské rozhraní konfigurace statistik

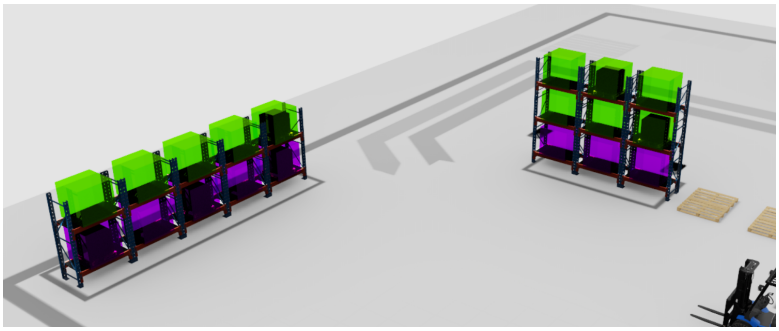
V aplikaci *Viewer* si pak uživatel může v rámci uživatelského rozhraní zobrazovat dostupné statistiky do modelu. Pro zobrazování dat do modelu statistiky využívají highlightů. Při přepnutí na jinou statistiku se pak highlighty původní statistiky odstraní tak, aby se data jednotlivých statistik nemíchala a model tak zůstával přehledný.

7.10.1 Typy vizualizačních modulů

Pro vizualizaci statistik jsou implementovány různé vizualizační moduly. Každý modul jinak zpracovává načítaná data a zobrazuje je ve 3D modelu. Zároveň má každý vizualizační modul své specifické uživatelské rozhraní.

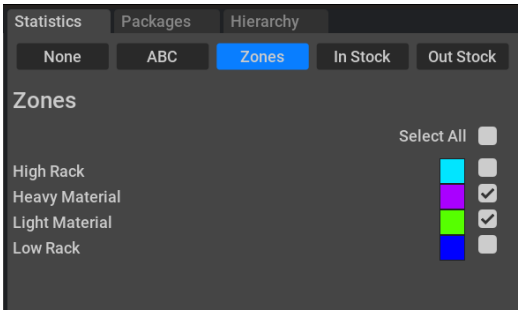
7.10.1.1 Skupiny pozic

Prvním typem vizualizačního modulu je modul, který popisuje skupiny skladových pozic. Data v tomto modulu jsou, jak název vypovídá, skupiny skladových pozic s určitou vlastností. Všechny skladové pozice konkrétní skupiny jsou pak v modelu vyznačeny highlighty se stejnou barvou, kterou si může uživatel pomocí color pickeru v rámci uživatelského rozhraní zvolit. Ukázka vizualizace skupin pozic se nachází na obrázku 7.14.



Obrázek 7.14: Vizualizace skupin pozic

V uživatelském rozhraní statistiky se při načtení dat uživateli zobrazí seznam skupin, kde každá položka (řádek) obsahuje název skupiny, již zmíněný color picker a checkbox, kterým si uživatel může nastavit, zda budou highlighty skupiny v modelu zobrazené či nikoliv. Kromě seznamu skupin uživatelské rozhraní obsahuje také checkbox, který umožňuje uživateli hromadně zobrazit či skrýt všechny skupiny najednou. Ve výchozím stavu po načtení dat jsou všechny skupiny zobrazené. Uživatelské rozhraní tohoto vizualizačního modulu se nachází na obrázku 7.15.

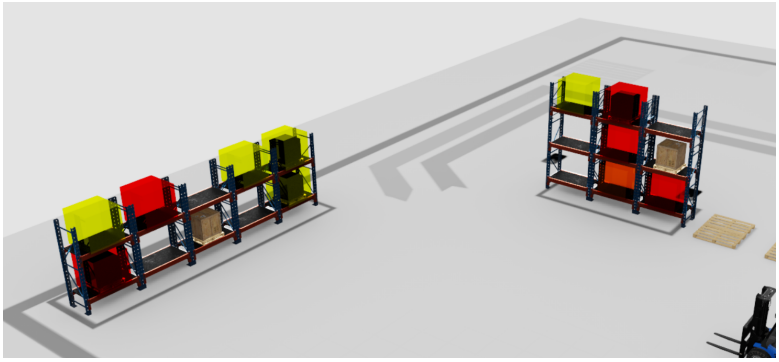


Obrázek 7.15: Uživatelské rozhraní vizualizačního modulu skupiny pozic

Příkladem statistiky, která využívá tohoto typu modulu je statistika *Zone*. Tato statistika rozděluje sklad na libovolný počet skladových zón, kde každá zóna zahrnuje seznam konkrétních skladových pozic.

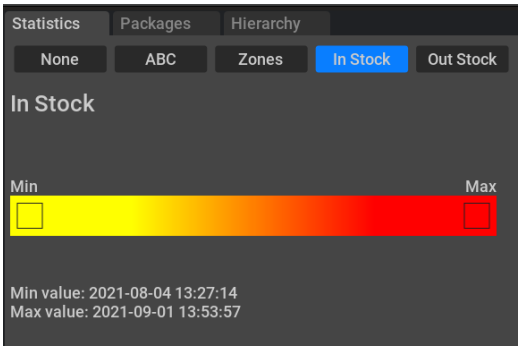
7.10.1.2 Heatmapy

Druhým typem vizualizačních modulů jsou pak heatmapy. Data u tohoto typu modulu jsou uspořádané dvojice skladová pozice - hodnota. Pro hodnoty je pak nutné, aby je bylo možné seřadit, a může se jednat o čísla (celá i reálná) nebo o datumy. Jednotlivé skladové pozice jsou pak v modelu vyznačeny pomocí highlightů s barvou určenou na základě lineární iterpolace mezi minimální a maximální hodnotou dat. Ukázska vizualizace heatmap se nachází na obrázku 7.16.



Obrázek 7.16: Vizualizace heatmap

Uživatelské rozhraní statistiky využívající tento typ modulu pak zobrazuje barevný gradient, kde si uživatel může zvolit barvy reprezentující minimální a maximální hodnotu heatmapy. Dále jsou zde také uvedeny minimální a maximální hodnoty dat. Uživatelské rozhraní tohoto vizualizačního modulu se nachází na obrázku 7.17.



Obrázek 7.17: Uživatelské rozhraní vizualizačního modulu heatmapy

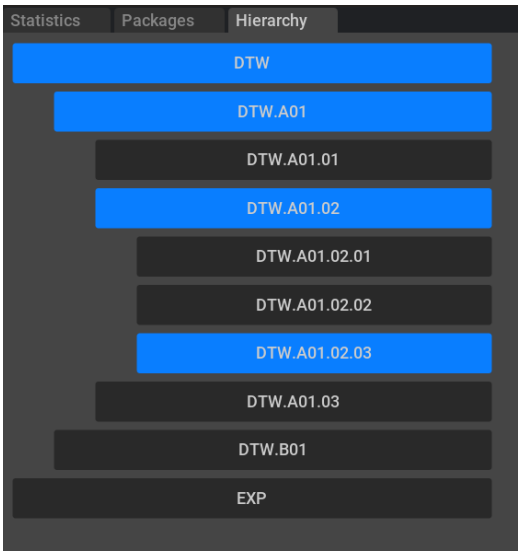
Příkladem statistiky, která využívá tohoto modulu je statistika *In Stock*, která pro každou skladovou pozici určuje, kdy bylo na tuto pozici naposledy zaskladněno zboží. Na stejném principu funguje pak také statistika *Out Stock*, která určuje, kdy bylo zboží ze skladové pozice naposledy vyskladněno.

7.11 Procházení skladové hierarchie

Aplikace *Viewer* umožňuje procházet skladovou hierarchii projektu. Tato funkcionalita je uživateli dostupná v okně uživatelského rozhraní s názvem *Hierarchy*. V tomto uživatelském rozhraní je uživateli zobrazena hierarchie digitálního dvojčete jako stromová struktura. Z počátku jsou zobrazeny pouze sklady, které jsou (kromě pomyslného kořenu) nejvyšší vrstvou této struktury. Při zvolení některého

z těchto skladů se dále zobrazí jeho potomci - řady. Při zvolení jiného skladu je pak výběr původního skladu zrušen a jeho potomci skryti. To samé platí pro všechny entity až na úroveň skladových pozic. Uživatelské rozhraní se nachází na obrázku 7.18.

Zvolení libovolného prvku skladové hierarchie v uživatelském rozhraní je doprovázeno vizuálními změnami ve scéně, které jsou řešeny animacemi. V případě zvolení skladové pozice jsou animace navíc doplněny přidáním highlightu.



Obrázek 7.18: Uživatelské rozhraní procházení skladové hierarchie

7.12 Export

Aplikace *Viewer* umožňuje export modelu digitálního dvojčete ve formátu *USD*, pro jeho následné využití v jiných aplikacích. Model se exportuje v jeho současném stavu se všemi částmi včetně případných balení a highlightů. Před exportem nejprve ale musí proběhnout dvě důležité věci.

Zaprvé je to nahrazení referencí a payloadů, které odkazují přes relativní cesty do globálních assetů. Tyto assety by totiž nebyly v exportovaném modelu k dispozici a *USD* objekty, na které odkazují, by tak v modelu chyběly. Tohoto nahrazení lze snadno dosáhnout pomocí flatteningu (viz sekce 4.2.9).

Zadruhé pak odstranění konfigurací pro komunikaci s databází a *REST API*. Jedná se o citlivé informace a nechceme tak, aby byly v exportovaném modelu k dispozici. Tohoto jednoduše dosáhneme smazáním *USD* primitiv `DatabaseConfiguration` a `RestApiConfiguration` popsaných v sekci 7.4.2, které tyto informace uchovávají.

Export je pro uživatele k dispozici v menu s cestou *File/Export*. Po zvolení této možnosti je uživateli umožněno určit, kam se výsledný soubor uloží. Pak proběhnou již zmíněné kroky flatteningu a odstranění konfiguračních primitiv a následně se výsledný soubor uloží na zvolené místo.

7.13 Build a distribuce aplikací

Build aplikací využívá nástrojů převzatých ze šablony *Omniverse Kit App Template*. Build se spouští zadáním příkazu `repo package`. Výsledkem buildu je *Thin package* (viz sekce 5.6.4.2) - archiv ve formátu `.zip`, který vznikne ve složce `_build`. Před spuštění aplikací je nutné archiv rozbalit a spustit skript `pull_kit_kernel`, který stáhne samotný framework **Kit**.

Implementace 8

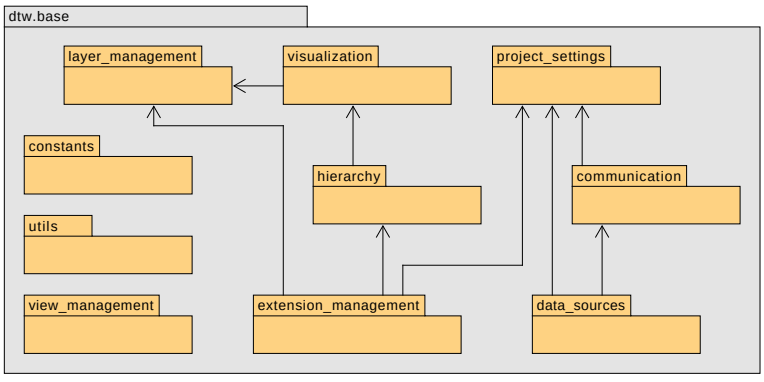
Tato kapitola se snaží poskytnout přehled implementace systému. Kapitola je členěna do sekcí které popisují jednotlivé moduly tvořící extensiony. V těchto sekcích je vysvětlena funkcionalita a implementace těchto modulů.

Každá sekce je doplněna *UML* diagramem modulu (obrázky 8.4 - 8.22), který slouží k lepšímu pochopení struktury tříd a jejich závislostí. Tyto diagramy obsahují i důležité závislosti na třídách jiných modulů. Vynechávají však závislosti, které nejsou pro pochopení implementace důležité a zkomplikovaly by celkový přehled.

8.1 Moduly

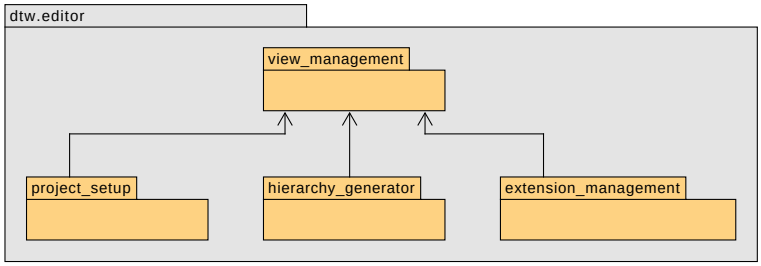
Jak již bylo zmíněno v úvodu kapitoly, jednotlivé extensiony jsou členěny do modulů. Třídy jsou do modulů sdružovány na základě společně poskytované funkcionality.

Tato sekce obsahuje *UML* diagramy (obrázky 8.1, 8.2 a 8.3), které popisují přehled modulů a jejich závislostí v rámci jednotlivých extensionů. Cílem těchto diagramů je poskytnout vysokoúrovňový přehled o rozdělení těchto modulů.

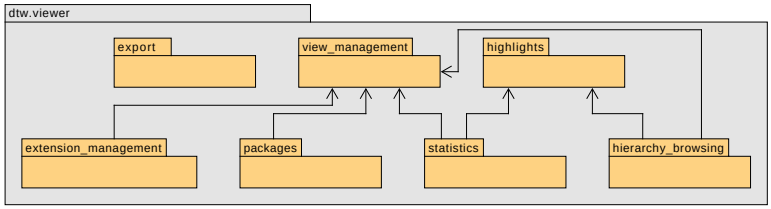


Obrázek 8.1: Moduly extensionu `dtw.base`[1]

[1]V diagramu byly záměrně vynechány závislosti na modulech `constants` a `utils`. Na těchto modulech jsou závislé téměř všechny ostatní moduly a diagram by tak byl velmi nepřehledný.



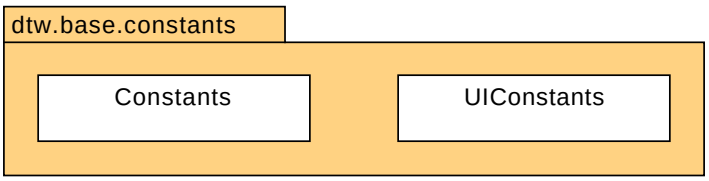
Obrázek 8.2: Moduly extensionu `dtw.editor`



Obrázek 8.3: Moduly extensionu `dtw.viewer`

8.2 Konstanty

Modul `constants` obsahuje pouze dvě třídy. Tyto třídy neobsahují žádnou logiku a slouží pouze k definici některých konstant, které jsou pak použity napříč různými moduly. Třída `Constants` obsahuje obecné konstanty jako například cesty ke složkám nebo *USD* cesty popisující některá důležitá primitiva ve scéně. Třída UIConstants pak definuje styly či velikosti použité v uživatelských rozhraních.

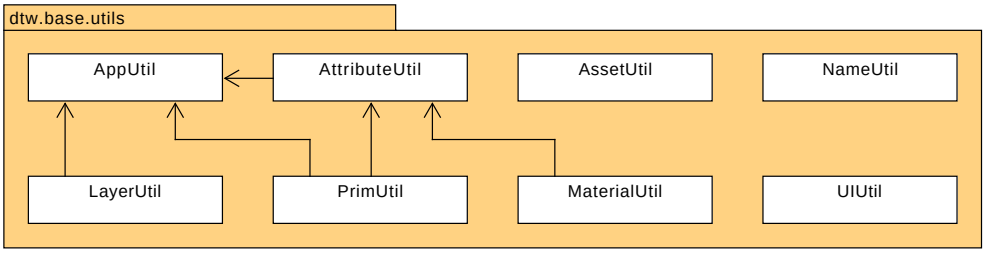


Obrázek 8.4: *UML* diagram konstant

8.3 Utility

Utility jsou knihovní třídy, které mají za cíl zjednodušit práci s některými funkcionalitami *USD* a frameworku **Omniverse**. Každá z těchto tříd sdružuje dohromady

funkce, které jsou pak využity napříč ostatními moduly. Nejdůležitější z těchto utilit je utilita `PrimUtil`, která usnadňuje práci s *USD* primitivy. Umožňuje pak například jejich vytváření, odstraňování a přesouvání nebo vytváření referencí a payloadů.



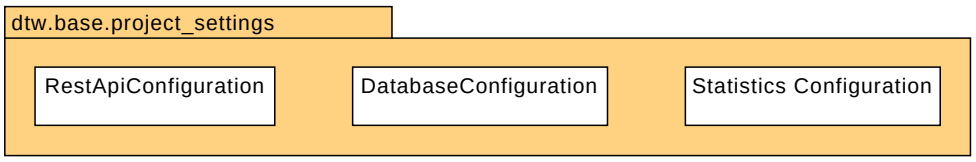
Obrázek 8.5: *UML* diagram utilit

8.4 Nastavení projektu

Základní nastavení projektu se skládá ze tří částí - nastavení komunikace s databází, nastavení komunikace s *REST API* a nastavení konfigurace statistik. Všechna tato nastavení jsou uloženy jako atributy primitiv součástí scény.

V rámci kódu jsou pak dále implementovány třídy, které hodnoty těchto atributů načítají a drží v proměnných. Jedná se o třídy `RestApiConfiguration`, `DatabaseConfiguration` a `StatisticsConfiguration`. Hlavním účelem těchto tříd je cachování a usnadnění přístupu k těmto nastavením z ostatních tříd.

Všechny tyto třídy implementují metodu `try_load_from_project_usd()`, která nastavení načte z atributů do paměti. Tato metoda se spouští při startu extensionu `dtw.base`.

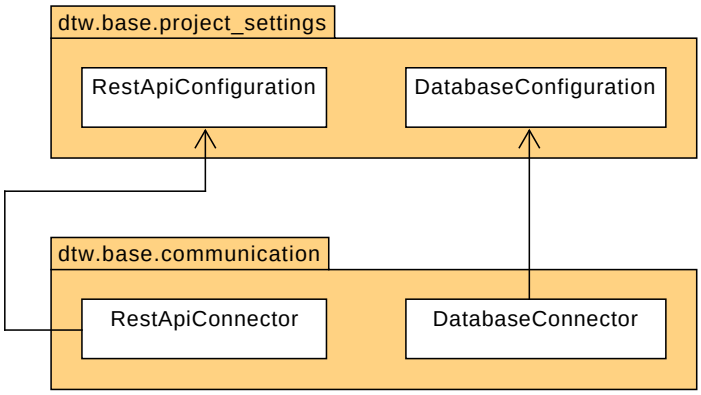


Obrázek 8.6: *UML* diagram nastavení projektu

8.5 Komunikace s externím systémem

Jak bylo zmíněno v sekci 7.2, aplikace využívají dvou způsobů komunikace s externím systémem **Aimtec DCIx** - přímý přístup do databáze a komunikace s *REST API*

pomocí *HTTP* requestů.



Obrázek 8.7: *UML* diagram komunikace s externím systémem

8.5.1 Přímý přístup do databáze

Komunikace využitím přímého přístupu do databáze je implementována ve třídě `DatabaseConnector`.

Pro komunikaci s databází třída využívá *Python* knihovnu `aioodbc`. Tato knihovna slouží k asynchronní komunikaci s `ODBC` databází. Oproti jiným *Python* knihovnám pro komunikaci s databází umožňuje použití `async/await` syntaxe a je ideální v kombinaci s knihovnou `asyncio`. Tato knihovna je v implementaci značně využita pro asynchronní operace, a tak je knihovna `aioodbc` vhodnou volbou.

8.5.2 REST API

Komunikace s *REST API* je implementována ve třídě `RestApiConnector`. Tato implementace využívá *Python* knihovnu `httpx` která podobně jako knihovna `aioodbc` dovoluje využití `async/await` syntaxe.

Třída se stará o *API* autentizaci a umožňuje ostatním třídám posílat *GET* a *POST* requesty na libovolné endpointy tohoto *API*.

8.6 Datové zdroje

Datové zdroje představují vrstvu, která slouží k abstrakci procesu načítání dat.

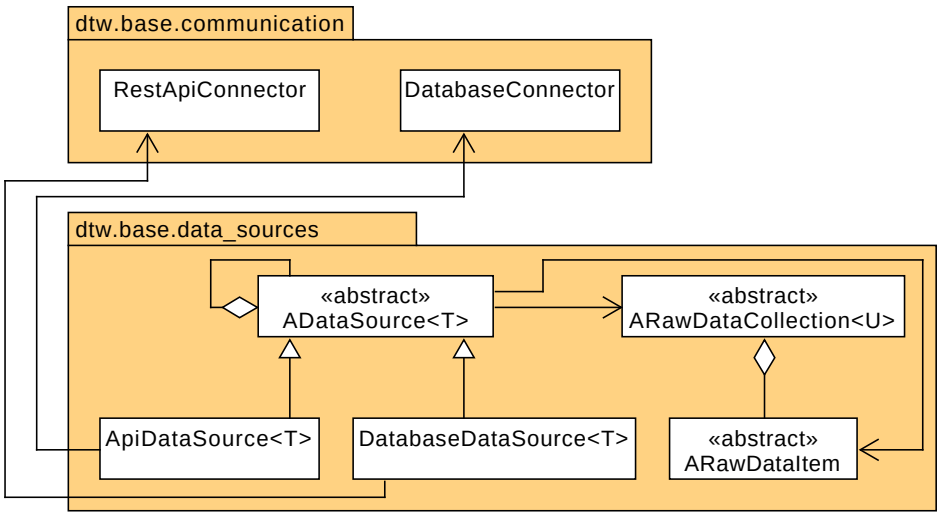
Základním prvkem je abstraktní generická třída `ADataSource<T>`, která popisuje obecný datový zdroj. Generickým parametrem `T` je zde jakákoliv třída dědící od třídy `ARawDataItemCollection`. Tato třída opět generická a reprezentuje kolekci

položek obsahujících surová data. Tyto položky jsou pak definované generickým parametrem `U`, kde `U` je třída dědící od třídy `ARawDataItem`.

Nejdůležitější metodou datových zdrojů je asynchronní metoda `load_data()`. Tato metoda slouží k načtení dat z datového zdroje a vrací kolekci dat definovanou generickým parametrem `T`. V rámci metody se načtou data z externího systému a z jednotlivých záznamů se vytvoří položky surových dat. Položkami je dále naplněna nová kolekce, která je metodou vrácena.

Metodou `load_data()` je také určeno, jakým způsobem se z externího systému budou načítat data. Implementovány jsou dva základní typy datových zdrojů. Třída `DatabaseDataSource` je implementací datového zdroje, která načítá data přímým přístupem do databáze. Třída `ApiDataSource` pak pro načítání dat využívá *REST API*.

Datové zdroje načítající konkrétní entity dále obsahují rodičovský datový zdroj. Rodičovský datový zdroj je tvořen instancí jedné ze tříd `DatabaseDataSource` nebo `ApiDataSource`. Při zavolání metody `load_data()` je pak pro načtení dat zavolána metoda `load_data()` rodičovského datového zdroje.



Obrázek 8.8: *UML* diagram datových zdrojů

8.7 Správa extensionu

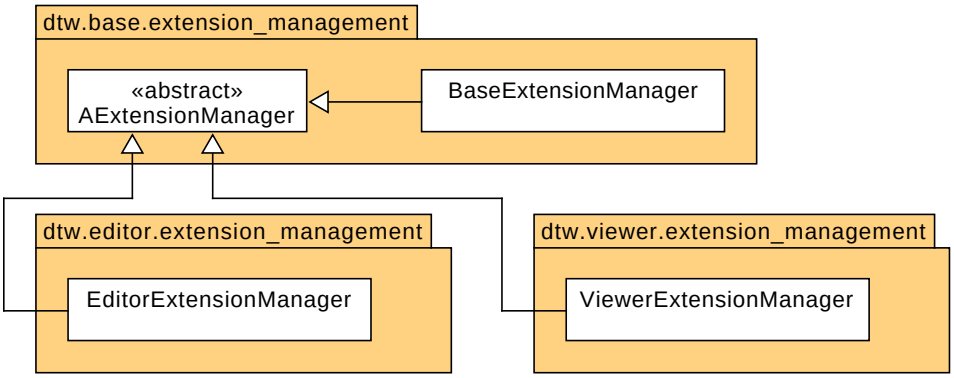
Správa extensionu řeší jeden důležitý problém. Při spuštění aplikace není ve výchozím stavu načtena scéna žádného projektu. Některé akce je ale potřeba provést

pokaždé při otevření projektu. Je tedy nutné nějakým způsobem odchytit okamžik načtení projektu a reagovat na něj.

Tato funkcionalita je implementována v abstraktní třídě `AExtensionManager`. Tato třída využívá eventů a subskripcí, které poskytuje framework `carb`, který je součástí frameworku **Omniverse**. Subskripce umožňují právě odchytávání eventů, které popisují určitou událost v rámci scény nebo frameworku.

Třída `AExtensionManager` dědí od třídy `omni.ext.IExt` popsané v sekci 5.2.3. Díky tomu se nám vytvoří v rámci spuštění aplikace instance této třídy a zavolá se její metoda `on_startup()`. V této metodě se pak registruje subskripce, která zařídí zavolání metody `on_loaded()` ve chvíli, kdy se scéna načte. Na stejném principu funguje také zavolání metody `on_reset()`, které se volá při odnačtení scény.

Metody `on_loaded()` a `on_reset()` jsou pak překryty ve třídách dědících od třídy `AExtensionManager`. Tyto metody spouštějí jednotlivé akce, které mají proběhnout ve chvíli, kdy se načte nebo odnačte scéna.



Obrázek 8.9: *UML* diagram správy extensionu

8.8 Uživatelská rozhraní

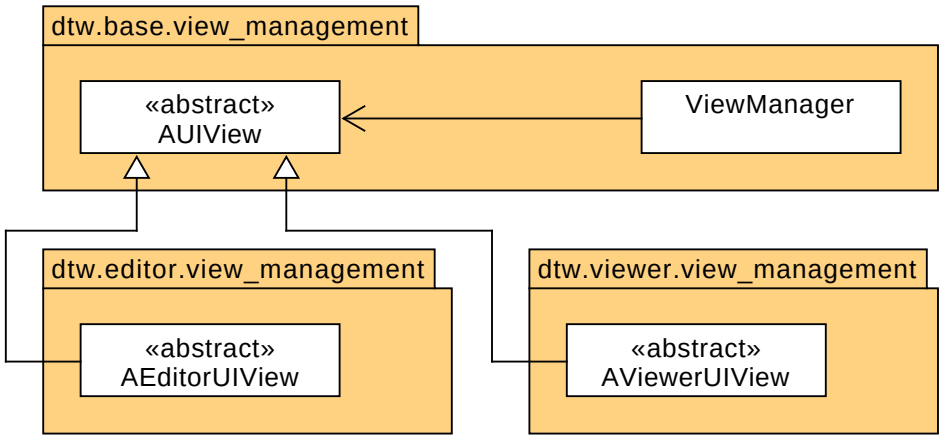
Uživatelská rozhraní jsou použita napříč několika moduly v rámci obou aplikací. Základem každého okna uživatelského rozhraní je abstraktní třída `AUIView`. Tato třída opět dědí od třídy `omni.ext.IExt` popsané v sekci 5.2.3. Díky tomu **Omniverse** automaticky při startu aplikace vytvoří jednu instanci každé třídy dědící od této třídy.

Třída má dále dvě důležité abstraktní metody, které je nutné v potomcích překrýt. První z těchto metod je metoda `create_window()`, která vytvoří okno s uživatelským rozhraním. Tato metoda je volána v konstruktoru abstraktní třídy `AUIView`. Dovoluje každému oknu definovat UI prvky a jejich layout.

Jak již bylo zmíněno v sekci 8.7, ne všechna data jsou ale načtena již při startu aplikace. Aplikace se ve výchozím stavu spouští bez načteného projektu, takže jakákoliv data projektu nejsou při volání metody `create_window()` k dispozici. Z tohoto důvodu je zde druhá metoda - metoda `load_data()`. Tato metoda je asynchronní a slouží právě pro načtení dat do uživatelského rozhraní poté, co jsou data k dispozici.

Pro zavolání metody `load_data()` ve správný čas slouží třída `ViewManager`. U instance této třídy musí být každé okno při vytvoření registrováno. V rámci extension managera zmíněného v kapitole 8.7 se pak po načtení projektu zavolá jeho metoda `load_views_data()`. Tato metoda zavolá metodu `load_data` všech oken registrovaných u tohoto managera.

Oba extensiony `dtw.editor` a `dtw.viewer` mají jednu instanci třídy `ViewManager`. Třídy `AEditorUIView` a `AViewerUIView` mají překrytý konstruktor, který okno při vytvoření automaticky registruje u příslušného managera. Od těchto tříd pak dědí třídy definující jednotlivá okna uživatelského rozhraní.



Obrázek 8.10: *UML* diagram uživatelských rozhraní

8.9 Konfigurace nastavení projektu

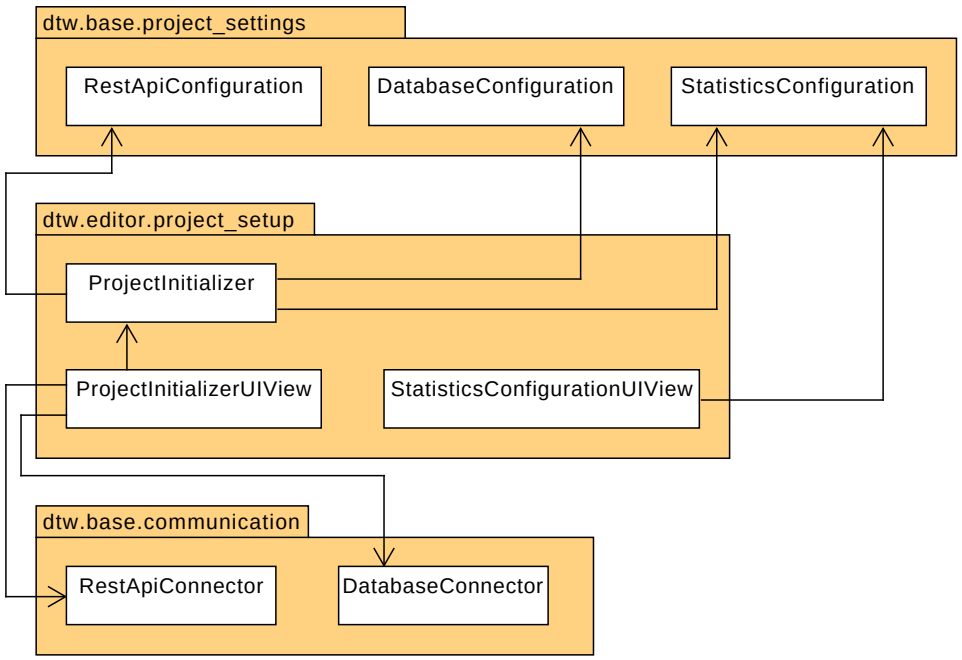
V rámci aplikace *Editor* je nutné konfigurovat nastavení projektu popsané v sekci 8.4. Dále je také nutné mít možnost generovaní základní hierarchie *USD* scény při tvorbě nového projektu.

Pro generování základní hierarchie *USD* scény slouží třída `ProjectInitializer`. Tato třída obsahuje jednu důležitou metodu - metodu `initialize_project()`. Tato metoda vytvoří všechny *USD* vrstvy projektu. Dále metoda projde *USD* scénu

a vygeneruje všechna základní primitiva, která ve scéně chybí. Výsledkem volání je, že ve scéně budou všechny vrstvy popsané v sekci 7.4.1 a všechna primitiva popsaná v sekci 7.4.2.

Modul pro konfiguraci projektu dále obsahuje také uživatelské rozhraní `ProjectInitializerUIView`. Toto rozhraní umožňuje inicializaci scény volaním metody `initialize_project()` třídy `ProjectInitializer`. Kromě toho je také v tomto rozhraní možné otestovat správnou konfiguraci databázového připojení a *REST API* připojení odesláním jednoduchých dotazů.

Kromě uživatelského rozhraní pro inicializaci projektu obsahuje modul ještě jedno uživatelské rozhraní - `StatisticsConfigurationUIView`. Toto rozhraní dovolí uživateli nakonfigurovat, které statistiky budou v rámci projektu k dispozici, jak bylo popsáno v sekci 7.10.



Obrázek 8.11: *UML* diagram konfigurace nastavení projektu

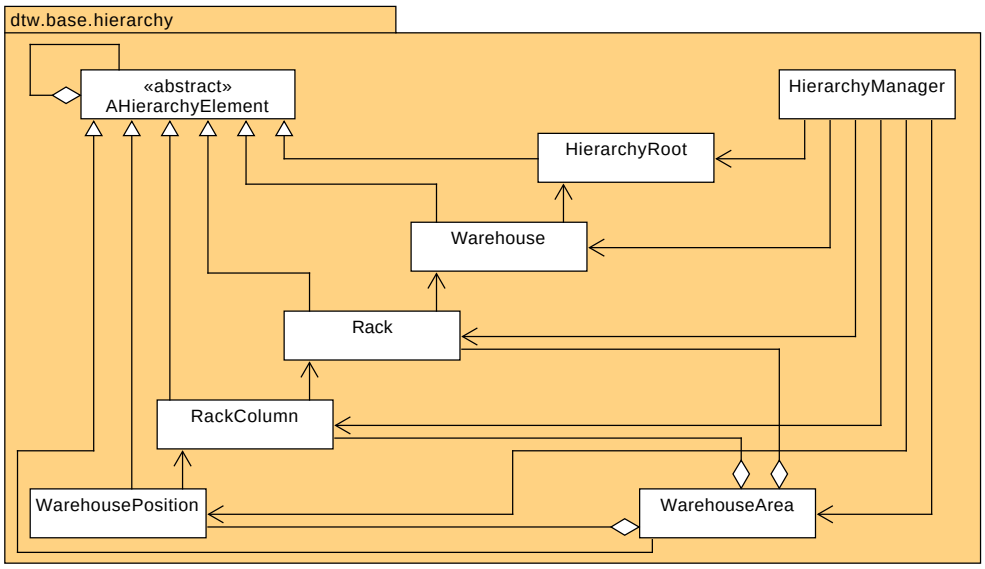
8.10 Hierarchie

Skladová hierarchie tvoří v rámci systému stromovou strukturu. Základním stavebním kamenem této struktury je abstraktní třída `AHierarchyElement`. Každá entita skladové hierarchie je dále popsána vlastní třídou, která od této abstraktní třídy dědí.

Jednotlivé instance těchto tříd vždy obsahují seznamy vygenerovaných a nevygenerovaných potomků. Kromě těchto seznamů obsahují všechny entity (kromě kořenu) také odkaz na rodiče. Struktura celé hierarchie pak vychází z jednoho kořenu, který je popsán třídou `HierarchyRoot`.

Speciální entitou hierarchie je skladová plocha popsaná třídou `WarehouseArea`. Tato třída dědí od třídy `AHierarchyElement`, zároveň ale využívá kompozici, která zahrnuje třídy `Rack`, `RackColumn` a `WarehousePosition`.

Pro správu hierarchie obsahuje modul ještě třídu `HierarchyManager`. Tato třída uchovává instanci kořenu hierarchie, stará se o načítání hierarchie z *USD* scény při otevření projektu. Dále ještě poskytuje pomocné metody pro práci s hierarchií, které se využívají například při jejím generování.



Obrázek 8.12: *UML* diagram hierarchie

8.11 Generování hierarchie

Skladovou hierarchii je v rámci aplikace *Editor* možné generovat. K tomu slouží uživatelské rozhraní implementované třídou `HierarchyGenratorUIView`. V rámci rozhraní si uživatel může nastavit regulární výraz sloužící k popisu a filtrování skladových pozic. Tento regulární výraz se pak zkompiluje a uchová v rámci třídy `RegexMatcher`.

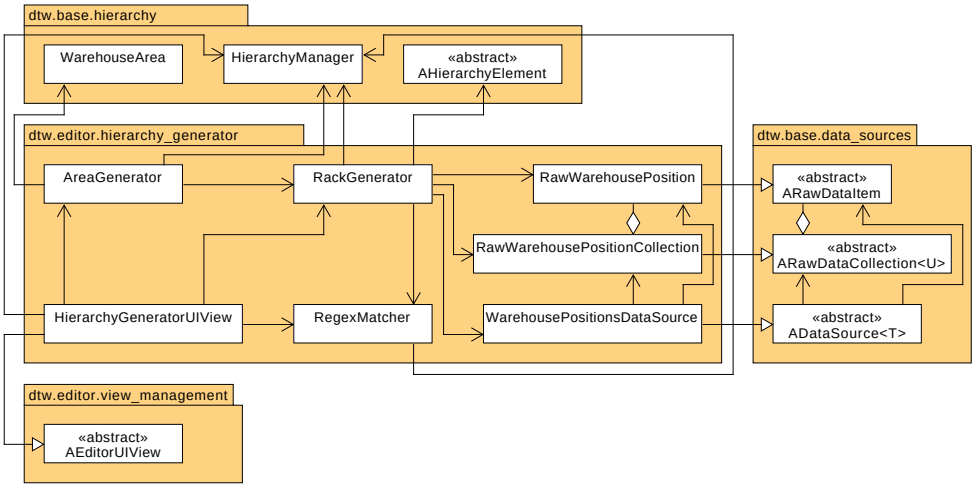
Samotné generování objektů pak mají na starosti třídy `RackGenerator` a `AreaGenerator`. V rámci třídy `RackGenerator` se pomocí datového zdroje definovaného

třídou `WarehousePositionsDataSource` načte seznam kódů skladových pozic. Ten to seznam je pak dále pomocí regulárního výrazu ve třídě `RegexMatcher` vyfiltrován a jednotlivé kódy pozic parsovány.

Parsované kódy pozic jsou dále předány třídě `HierarchyManager`, která vytvoří instance tříd reprezentující jednotlivé entity skladové hierarchie. Potomci jsou v tomto kroku rodičům vždy přidáni pouze do seznamu nevygenerovaných potomků.

Po zvolení objektu pro vygenerování v uživatelském rozhraní jsou objekty pomocí třídy `RackGenerator` vygenerována *USD* primitiva ve scéně. Dále jsou nastaveny jejich atributy tak, aby všechny důležité informace byly persistentně uloženy a bylo je možné načíst při otevření projektu. Až pak jsou objekty v rodičích přesunuty do seznamu vygenerovaných potomků. Kromě primitiv reprezentujících jednotlivé entity hierarchie je ještě vygenerováno primitivum 3D modelu regálu. Pro toto je využit payload, který odkazuje do globálních assetů.

Při generování skladových ploch je proces až do fáze zvolení a vygenerování objektu identický. V této fázi se pak místo třídy `RackGenerator` využije pro generování objektů třída `AreaGenerator`. V hierarchii *USD* scény je skladová plocha vygenerována jako jedno primitivum. Místo 3D modelu regálu jsou vygenerovány čáry, které plochu ohraničují.



Obrázek 8.13: *UML* diagram generování hierarchie

8.12 Přepínání USD vrstev

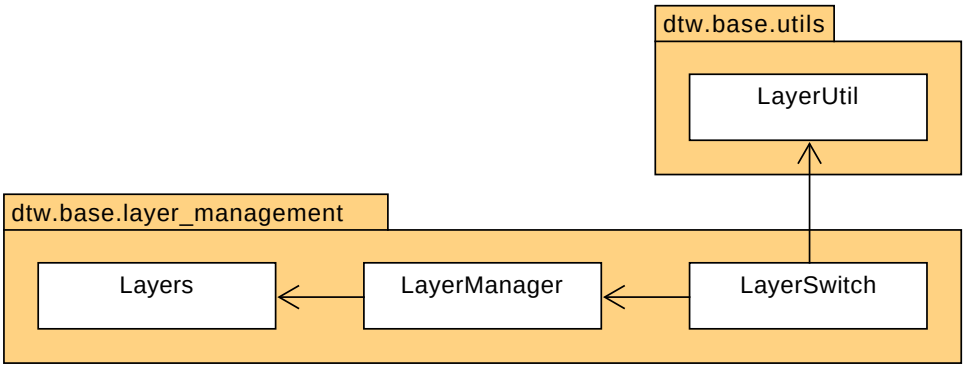
Scéna projektu je členěna do čtyř vrstev mezi kterými je nutné v určitých okamžicích přepínat. Přepínání *USD* vrstev jako takových je implementováno v utilitě

`LayerUtil`.

Pro většinu akcí, které vyžadují přepnutí *USD* vrstvy ale platí, že po vykonání akce chceme vrstvu přepnout zase zpět. Přesně k tomuto účelu slouží třída `LayerSwitch`. Třída implementuje metody `__enter__()` a `__exit__()`, což umožňuje její použití v kombinaci s *Python* strukturou `with`. Při vstupu do bloku kódu, který strukturu `with` následuje je vrstva přepnuta na zvolenou vrstvu. Při výstupu z bloku je pak přepnuta zase zpět na původní vrstvu.

Pro tuto funkcionalitu je potřeba mít možnost zjistit aktuálně zvolenou vrstvu, což řeší třída `LayerManager`. Tato třída drží identifikátor aktuální vrstvy. Při každé změně aktuální vrstvy je tento identifikátor změněn. Aktuální vrstvu lze ale změnit uživatelem v aplikaci *Editor* i ručně. Toto je ošetřeno využitím frameworku `carb` a pomocí subskripcí je reagováno na změnu vrstvy uživatelem.

Modul obsahuje ještě třídu `Layers`, která pouze definuje identifikátory vrstev jako konstanty.



Obrázek 8.14: *UML* diagram přepínání *USD* vrstev

8.13 Vizualizace a vizuály

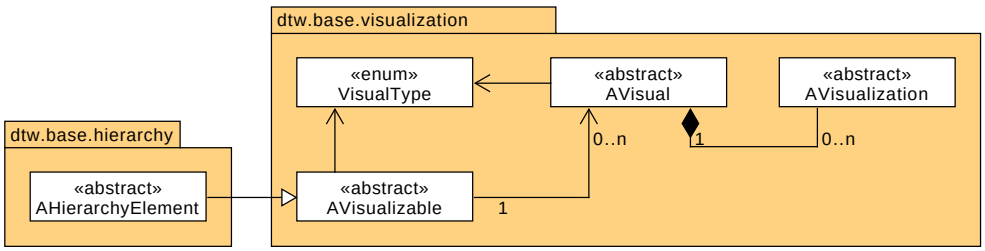
Vizualizace a vizuály vznikly za účelem abstrakce a generalizace společné implementace v rámci balení a highlightů. Tato společná implementace řeší jeden problém - kompozici celku z více částí.

Vizuál reprezentovaný abstraktní třídou `AVisual` je zde celkem, který se skládá z libovolného počtu vizualizací reprezentovaných abstraktní třídou `AVisualization`.

Vizualizace tvořící jeden vizuál se můžou navíc během času měnit. Je tedy nutné mít možnost je přidávat a odebírat. Vizuál pak musí mít možnost na tuto změnu nějakým způsobem reagovat - nejčastěji změnou vzhledu.

Vizuály je také možné navázat na konkrétní objekt. Tímto objektem je potomek abstraktní třídy `AVisualizable`. Instance této třídy umožňuje zaregistrovat vizuál pod určitým typem. Pro každý typ ale může být registrovaný pouze jedem vizuál. Vizuál registrovaný pro konkrétní typ je z instance pak možné získat a využít například pro přidání či odebrání vizualizace. Jako výčet možných typů vizuálu slouží enum `VisualType`.

V momentální stavu od třídy `AVisualizable` dědí pouze třída `AHierarchyElement` a tak je možné přidávat vizuály pouze prvkům této třídy - nejčastěji skladovým pozicím. Modul je ale navržen tak, aby bylo v budoucnu snadné tuto funkcionalitu přidat i jiným entitám jednoduchým oddděděním od této třídy.



Obrázek 8.15: *UML* diagram vizualizací a vizuálů

8.14 Balení

Balení jsou první z dvou využití vizualizací a vizuálů.

Nejdůležitější třídou, která se stará o balení je třída `PackageManager`. Tato třída se stará o načítání a správu balení a správu vizualizací a vizuálů těchto balení.

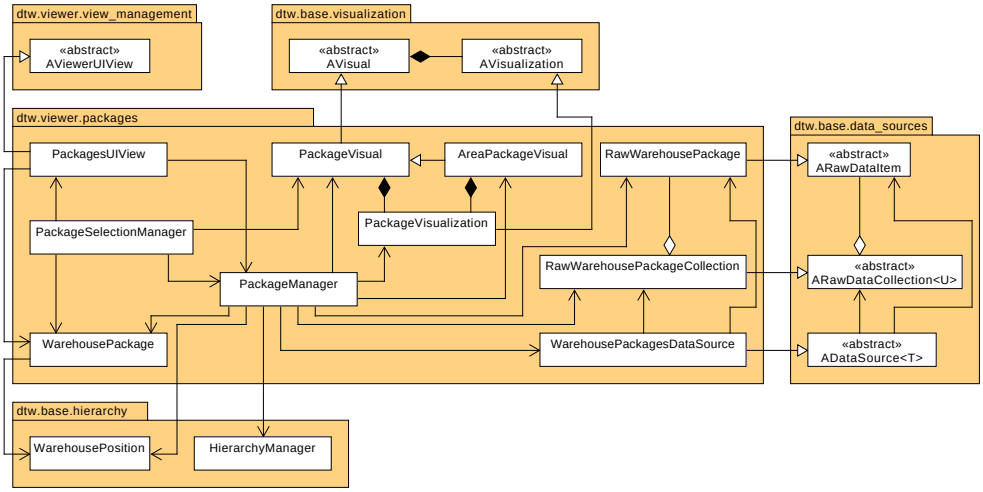
Při otevření projektu nejprve třída vytvoří a zaregistruje vizuály pro všechny skladové pozice projektu. Pro standardní skladové pozice, které jsou součástí regálům se registrují vizuály typu `PackageVisual`. V případě skladových ploch se pak registrují vizuály typu `AreaPackageVisual`, které mají možnost více balení (vizualizací) rozmístit v prostoru v rámci své plochy.

Při načítání balení třída zavolá datový zdroj definovaný třídou `WarehousePackagesDataSource`, ze kterého získá kolekci surových dat balení (instanci třídy `RawWarehousePackageCollection`). Pro každou položku této kolekce (instanci třídy `RawWarehousePackage`) pak dohledá příslušnou skladovou pozici a vytvoří instanci třídy `WarehousePackage`. Dále pro každé balení vytvoří vizualizaci (instanci třídy `PackageVisualization`) a přidá tuto vizualizaci do příslušného vizuálu.

Načítání balení proběhne automaticky při otevření projektu. Je ale také možné manuálně provést refresh balení, či je odnačíst pomocí uživatelského rozhraní defi-

novaného třídou `PackagesUIView`.

Reakci na zvolení balení pomocí myši má na starosti třída `PackageSelectionManager`. Využívá k tomu eventů a subskripcí frameworku `carb`. Při odchycení eventu popisujícího změnu zvoleného objektu v rámci scény zjistí, který objekt je zvolen. Dále pak prochází primitiva v hierarchii *USD* scény od zvoleného primitiva po kořen. Pokud v rámci procházení narazí na primitivum vytvořené vizuálem balení, zjistí z jeho atributu, o které balení se jedná. Informace tohoto balení jsou pak zobrazeny v rámci uživatelského rozhraní definovaného třídou `PackagesUIView`. V případě zvolení více primitiv tažením myši probíhá procházení hierarchie pro všechna tato primitiva.



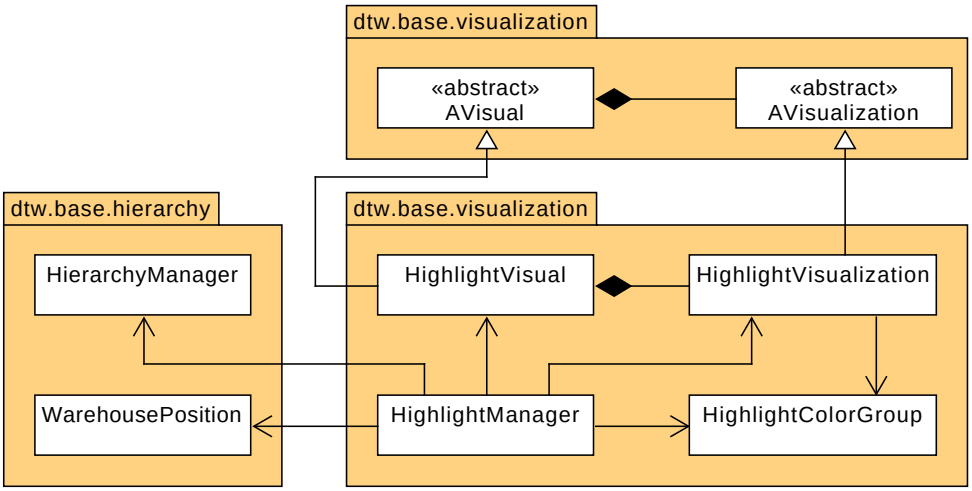
Obrázek 8.16: *UML* diagram balení

8.15 Highlighty

Highlighty jsou druhým využitím vizualizací a vizuálů. Hlavní třídou, která highlighty spravuje je třída `HighlightManager`. Tato třída stejně jako třída `PackageManager` při otevření projektu vytvoří a zaregistruje vizuály typu `HighlightVisual` pro všechny skladové pozice. Dále poskytuje ostatním třídám funkci `create_highlight_visualization()`, která umožňuje snadné vytoření a přidání vizualizace highlightu typu `HighlightVisualization` do vizuálu konkrétní skladové pozice.

U jednotlivých vizualizací je také potřeba určit jejich barvu. Vzhledem k časté přítomnosti více highlightů stejné barvy ve scéně není efektivní vytvářet pro každý z nich samostatný materiál. Mnohem optimálnějším řešením je tyto highlighty seskupit podle barev. Pro každou skupinu tak postačí pouze jedna instance materiálu s požadovanou barvou. Tyto barevné skupiny jsou určeny třídou `HighlightCo-`

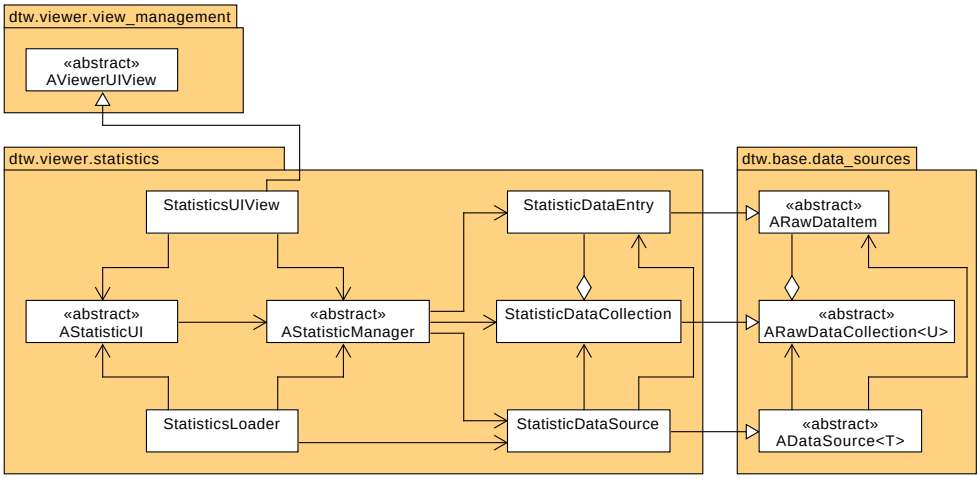
`lorGroup`. Při vytvoření instance této třídy se ve scéně vytvoří nebo ze scény načte instance materiálu. V rámci vytváření instance třídy `HighlightVisualization` pak předáváme instanci třídy `HighlightColorGroup`. Tím definujeme barevnou skupinu a materiál, které budou pro highlight použity při zobrazení této vizualizace.



Obrázek 8.17: *UML* diagram highlightů

8.16 Statistiky

Statistiky se skládají z celkem čtyř částí - definice, datový zdroj, manager a uživatelské rozhraní. Tyto části jsou dále podrobněji rozebrány v následujících sekcích.



Obrázek 8.18: *UML* diagram statistik

8.16.1 Definice

Každá statistika je popsána v definici - `JSON` souboru. V tomto souboru se definuje její název, typ použitého vizualizačního modulu, detaily použití tohoto modulu a datový zdroj. Pro načítání dat statistik je vždy použito *REST API* a datový zdroj je tak definován jako endpoint, na který se bude posílat request, a samotná data tohoto requestu.

8.16.2 Datový zdroj

Datové zdroje používané statistikami jsou instance třídy `StatisticDataSource`. Jak již bylo zmíněno v sekci 8.16.1, tyto datové zdroje načítají data pomocí *REST API* endpointu určeného v definici statistiky. Na tento endpoint pošlou data, která jsou také načtena z této definice. Tato data ovšem nejprve doplní o informace kódu projektu (pole `projectCode`) a typu spojení (pole `connectionType`), které *REST API* back-endu systému **GDTW** při načítání statistik vyžaduje.

8.16.3 Manager

Každá statistika musí mít dále managera. Ten je pro jednotlivé vizualizační moduly definovaný třídou dědící od třídy `AStatisticManager`. Při zvolení statistiky manager nejprve pomocí datového zdroje načte data statistiky. Tato data následně transformuje a seskupuje do různých datových struktur pro lepší manipulaci. Manager má dále na starosti vizualizaci těchto dat, pro kterou poskytuje funkce ostatním třídám.

8.16.4 Uživatelské rozhraní

Všechny statistiky jsou součástí jednoho okna uživatelského rozhraní definovaného třídou `StatisticsUIView`. V tomto uživatelském rozhraní si může uživatel změnit zvolenou statistiku. Každý modul má ale jinou strukturu dat a je tak nutné mít možnost měnit prvky a rozložení tohoto uživatelského rozhraní. Toto je umožněno pomocí abstraktní třídy `AStatisticUI` obsahující abstraktní metodu `create_ui()`.

Každý modul statistik musí obsahovat třídu dědící od třídy `AStatisticUI`. Třída překrývá metodu `create_ui()`, ve které definuje, jak má uživatelské rozhraní statistiky využívající tento modul vypadat. Uživatelské rozhraní definované třídou `StatisticsUIView` při změně zvolené statistiky zavolá tuto metodu a přegeneruje svůj obsah.

8.16.5 Načtení a sestavení

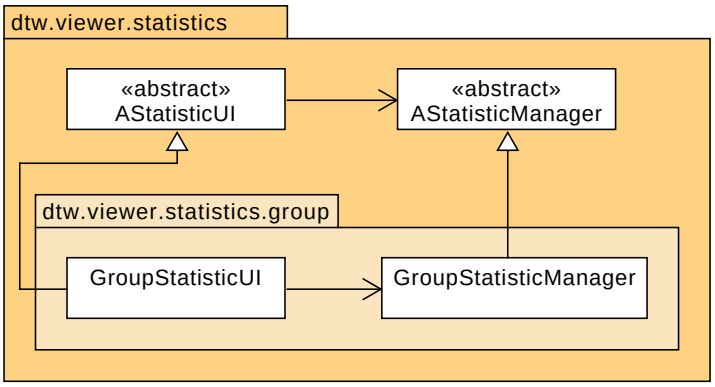
K načtení a sestavení všech částí statistik slouží třída `StatisticsLoader`. Tato třída při otevření projektu nejprve pomocí třídy `StatisticsConfiguration` zjistí, které statistiky jsou v projektu použity. Pro všechny tyto statistiky pak načte jejich definice. Na základě definic pak pro každou statistiku vytvoří instanci datového zdroje (třídy `StatisticDataSource`), managera (potomka `AStatisticManager`) a uživatelského rozhraní (potomka `AStatisticUI`). Vytvořené statistiky nakonec zaregistruje u uživatelského rozhraní definovaného třídou `StatisticsUIView`.

8.16.6 Vizualizační moduly

8.16.6.1 Skupiny pozic

U vizualiačního modulu skupiny pozic jsou data z *REST API* načtena jako pole záznamů, kde každý záznam obsahuje kód skladové pozice a kód skupiny. Tato data jsou dále managerem definovaným třídou `GroupStatisticManager` seskupena do slovníku. Klíčem tohoto slovníku je kód skupiny a hodnotou je další slovník reprezentující skupinu. V tomto druhém slovníku je pak klíčem skladové pozice a hodnotou vizualizace highlightu (třída `HighlightVisualization`). Toto seskupení dat umožňuje pro každou pozici skupiny vytvořit a uchovat vizuál highlightu, který pak lze zpětně použít pro jeho skrytí.

Uživatelské rozhraní definované třídou `GroupStatisticUI` po načtení dat získá od managera seznam skupin. Poté pro každou skupinu vytvoří řádek, který pomocí checkboxu umožní vizualizaci této skupiny a pomocí colorpickeru změnu barvy této skupiny.

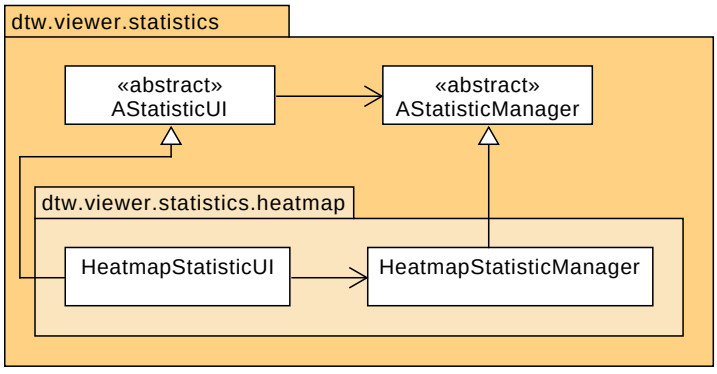


Obrázek 8.19: *UML* diagram vizualizačního modulu skupiny pozic

8.16.6.2 Heatmapy

U vizualizačního modulu heatmapy tvoří data dvojice skladová pozice - hodnota. Hodnoty mohou být čísla nebo datumy. Tuto informaci je pro konkrétní statistiku nutné specifikovat v definici v detailech použití modulu polem `heatmapField`. Při načtení dat manager definovaný třídou `HeatmapStatisticManager` nejprve projde všechna data a případně parsuje datumy. Při průchodu dat také nalezne minimální a maximální hodnoty. Při vizualizaci dat určí barvy pro jednotlivé skladové pozice na základě lineární interpolace mezi minimální a maximální hodnoutou. Dále pak vytvoří vizualizace highlightů, které uchová pro jejich zpětné odstranění.

Uživatelské rozhraní tohoto vizualizačního modulu je definováno třídou `HeatmapStatisticUI`. Toto rozhraní obsahuje gradient s colorpickery, kterými lze určit barvy gradientu. Dále obsahuje informace o minimální a maximální hodnotě pozic.

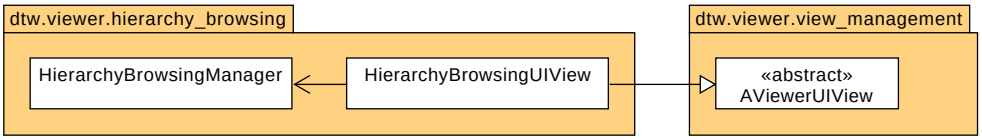


Obrázek 8.20: *UML* diagram vizualizačního modulu heatmapy

8.17 Procházení skladové hierarchie

Hlavní třídou, která ovládá procházení skladové hierarchie je třída `HierarchyBrowsingManager`. Tato třída poskytuje metody realizující vizuální změny, které mají nastat při zvolení a zrušení volby jednotlivých prvků skladové hierarchie. Plynulé pohyby primitiv reprezentující jednotlivé prvky jsou realizovány asynchronní funkcí pomocí lineární interpolace.

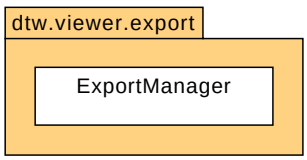
Uživatelské rozhraní procházení skladové hierarchie je definováno třídou `HierarchyBrowsingUIView`. Jednotlivá tlačítka sloužící k ovládání procházení jsou dynamicky vytvářena a odstraňována za běhu. Tato tlačítka volají metody třídy `HierarchyBrowsingManager` pro vyvolání vizuálních změn ve scéně.



Obrázek 8.21: *UML* diagram procházení skladové hierarchie

8.18 Export

O export se stará třída `ExportManager`. Tato třída při spuštění aplikace přidá položku `Export` do menu File. Pro zvolení umístění exportovaného souboru pak využije možnosti extensionu `omni.kit.window.file_exporter`. Tento extension umožňuje zobrazit okno pro zvolení složky a pojmenování souboru a reagovat na stisk tlačítka `Save`. V reakci na toto tlačítko se pak provede flattening a odstranění konfiguračních primitiv, jak bylo zmíněno v sekci 7.12. Výsledná scéna se pak uloží do zvoleného souboru.



Obrázek 8.22: *UML* diagram exportu

Dosažené výsledky 9

9.1 Aplikace

V rámci práce byl pomocí frameworku **NVIDIA Omniverse** vytvořen systém pro tvorbu digitálních dvojčat skladů a továren. Tento systém se skládá ze dvou aplikací. Aplikace *Editor* slouží pro návrh a tvorbu projektu digitálního dvojčete. Umožňuje založení nového projektu, konfiguraci projektu, generování skladové hierarchie a přidávání designových prvků. Aplikace *Viewer* slouží pro prohlížení projektů. Tato aplikace umožňuje prohlížení 3D modelu, zobrazování balení, statistických veličin a procházení skladové hierarchie. Screenshoty aplikací se nachází v příloze B na obrázcích B.1 a B.2.

9.2 Modelové digitální dvojče

Pro ověření funkcionality obou aplikací byl vytvořen projekt digitáního dvojčete *MOFA* (**Mo**del **Fa**ctory). Jedná se o digitální dvojče skladu fiktivního zákazníka. Projekt byl zvolen z důvodu, že neobsahuje žádná citlivá data zákazníků, zároveň však obsahuje dostatek fiktivních (mock) dat potřebných pro demonstraci všech funkcí vytvořených aplikací. Projekt *MOFA* je vidět na screenshotech aplikací v příloze B.

9.3 Požadavky na zdroje

Požadavky na zdroje aplikací se odvíjejí od požadavků **Omniverse Kit**u. Společnost **NVIDIA** doporučuje následující minimální požadavky: [48]

- Podporované operační systémy - *Windows 10/11, Ubuntu 20.04/22.04* nebo *CentOS 7*
- CPU - *Intel i7 Gen 5* nebo *AMD Ryzen*
- RAM - 16 GB

- GPU - *GeForce RTX 3070*
- Disk - 250 GB

Je však důležité poznamenat, že v průběhu celého vývoje jsem pracoval na počítači s grafickou kartou *GeForce RTX 3060*, který se ukázal jako dostatečně výkonný pro potřeby práce.

9.4 Kritéria

Doba spouštění vybuilděných aplikací se pohybuje v rozmezí 3 - 20 sekund. Relativně velký rozptyl je zde způsoben cachováním, kde při prvním spuštění aplikace po buildu se hodnota nacházela blíže horní hranici a při následujících spuštěních blíže té dolní.

Počet snímků aplikace *Editor* se za normálních okolností pohybuje mezi 50 a 70. U aplikace *Viewer* se tyto hodnoty pohybují mezi 35 a 45 snímků za sekundu. Rozdíl mezi aplikacemi je způsoben větším rozlišením viewportu.

Velikost projektu *MOFA* je kolem 160 kB, jedná se ovšem o relativně malý projekt.

Parametry byly měřeny na počítači s grafickou kartou *GeForce RTX 3060*, osmijádrovým procesorem *AMD Ryzen 7 5800H* a 16 GB operační paměti.

9.5 Zhodnocení

Práce ve frameworku **NVIDIA Omniverse** a jeho použití pro implementaci digitálních dvojčat má řadu výhod i nevýhod.

Mezi hlavní výhody použití frameworku by patřila hlavně rozsáhlá funkcionalita, kterou s sebou přináší technologie **Universal Scene Description**. Další výhodou je možnost podrobného členění aplikací na extensiony a moduly, což poskytuje velkou kontrolu nad strukturou a architekturou projektů. Velkými výhodami jsou také skvělý realistický vzhled a vizuální dojem dosažený využitím technologie RTX. Přínosem je také množství předpřipravených assetů, které je možné při tvorbě digitálních dvojčat použít.

Na druhou stranu mezi nevýhody frameworku by patřila hlavně jeho krátká historie. Jelikož se jedná o relativně novou technologii, dokumentace a tutoriály jsou značně omezené ve srovnání s jinými staršími platformami, jako například **Unity**. Tato nevýhoda je ještě zesílena velkou komplexností frameworku, kde naučit se pracovat se všemi jeho funkcemi a do hloubky jim porozumět zabere relativně velký čas a úsilí.

Nicméně dle mého názoru výhody frameworku převažují jeho nevýhody a framework **NVIDIA Omniverse** bych tak určitě označil jako vhodný pro implementaci digitálních dvojčat.

Závěr 10

Cílem práce bylo vytvoření systému pro tvorbu a prohlížení digitálních dvojčat skladů využítím frameworku **NVIDIA Omniverse**. Za tímto účelem jsem tento framework důkladně nastudoval. Dále jsem důkladně prostudoval také technologii **Universal Scene Description**, na které je **NVIDIA Omniverse** postavený.

Na základě získaných poznatků jsem ve frameworku navrhnul a implemenoval systém o dvou aplikacích. Jedna aplikace slouží pro tvorbu digitálních dvojčat skladů, druhá pro jejich prohlížení.

Funkčnost vzniklého systému jsem ověřil na jednoduchém projektu digitálního dvojčete skladu fiktivního zákazníka.

Zhodnotil jsem klady a zápory využití frameworku **NVIDIA Omniverse** pro implementaci digitálních dvojčat. Framework jsem uznal vhodným pro tyto účely.

Zadání práce považuji za splněné.

Bibliografie

1. CAMERON HASHEMI-POUR, David Essex. *Logisitics*. TechTarget, 2023. Dostupné také z: https://www.techtarget.com/searcherp/definition/logistics.
2. AXISADMAN. *NVIDIA Robotics*. NVIDIA, 2024. Dostupné také z: https://s3.amazonaws.com/cms.ipressroom.com/219/files/20240/659a52363d6332fd16032963%5C_nvidia-robotics-image/nvidia-robotics-image%5C_c5b3d5ea-3f12-4933-9fee-29bf10d1ec69-prv.jpg. NVIDIA Press Kit.
3. O'DONNELL, Jim. *Warehouse Management System (WMS)*. TechTarget, 2020. Dostupné také z: https://www.techtarget.com/searcherp/definition/warehouse-management-system-WMS.
4. *O společnosti*. AIMTEC a.s., 2023. Dostupné také z: https://www.aimtecglobal.com/o-spolecnosti.
5. *Aimtec DCIx*. AIMTEC a.s., 2023. Dostupné také z: https://www.aimtecglobal.com/dcix.
6. *What Is a Digital Twin*. Unity Technologies, 2023. Dostupné také z: https://unity.com/solutions/digital-twin-definition.
7. *Use Cases of Integrating Digital Twins in Warehouse Operations*. Cerexio, 2023. Dostupné také z: https://cerexio.com/blog/use-cases-of-integrating-digital-twins-in-warehouse-operations.
8. RAIZADA, Astha. *Exploring the Benefits of Digital Twin Technology in Warehouse Management*. Copper Digital, 2023. Dostupné také z: https://copperdigital.com/blog/digital-twin-technology-in-warehouse-management/.
9. PAŠKO, Patrik. *twinzo - Digital Twin platform*. 5.0 technologies j.s.a., 2022. Dostupné také z: https://www.twinzo.eu/blog/solutions-2/twinzo-digital-twin-platform-4.
10. *Digital Twins*. Unity Technologies, 2023. Dostupné také z: https://unity.com/solutions/digital-twins.

11. *FlexSim + Digital Twin*. Flexsim. Dostupné také z: https://www.flexsim.com/digital-twin/.
12. *Introduction to USD*. Pixar Animation Studios, 2021. Dostupné také z: https://openusd.org/release/intro.html.
13. *Open Source Release*. Pixar Animation Studios, 2021. Dostupné také z: https://openusd.org/release/press%5C_opensource%5C_release.html.
14. *Pixar, Adobe, Apple, Autodesk, and NVIDIA Form Alliance for OpenUSD to Drive Open Standards for 3D Content*. The Linux Foundation, 2023. Dostupné také z: https://www.linuxfoundation.org/press/announcing-alliance-for-open-usd-aousd.
15. *Alliance for OpenUSD Unveils Roadmap for Core USD Specification and Ecosystem Collaboration*. The Linux Foundation, 2023. Dostupné také z: https://www.linuxfoundation.org/press/alliance-for-openusd-unveils-roadmap-for-core-usd-specification.
16. *USD Terms and Concepts*. Pixar Animation Studios, 2021. Dostupné také z: https://openusd.org/release/glossary.html. TODO.
17. *UsdRelationship Class Reference*. Pixar Animation Studios, 2023. Dostupné také z: https://openusd.org/release/api/class%5C_usd%5C_relationship.html. USD API Documentation.
18. *UsdStage Class Reference*. Pixar Animation Studios, 2023. Dostupné také z: https://openusd.org/release/api/class%5C_usd%5C_stage.html. USD API Documentation.
19. *SdfPath Class Reference*. Pixar Animation Studios, 2023. Dostupné také z: https://openusd.org/release/api/class%5C_sdf%5C_path.html. USD API Documentation.
20. *UsdReferences Class Reference*. Pixar Animation Studios, 2023. Dostupné také z: https://openusd.org/release/api/class%5C_usd%5C_references.html. USD API Documentation.
21. *Usdz File Format Specification*. Pixar Animation Studios, 2021. Dostupné také z: https://openusd.org/release/spec%5C_usdz.html.
22. *Hydra*. Disney/Pixar, 2019. Dostupné také z: https://openusd.org/files/Siggraph2019%5C_Hydra.pdf.
23. *NVIDIA Omniverse*. NVIDIA Corporation, 2023. Dostupné také z: https://www.nvidia.com/en-us/omniverse/.
24. *Platform Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/platform/latest/index.html. NVIDIA Omniverse Documentation.

25. *Nucleus Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/nucleus/latest/index.html. NVIDIA Omniverse Documentation.

26. *Connect Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/connect/latest/index.html. NVIDIA Omniverse Documentation.

27. *NVIDIA Omniverse Multi-User Collaboration*. NVIDIA, 2024. Dostupné také z: https://s3.amazonaws.com/cms.ipressroom.com/219/files/20224/627ee719b3aed31115af1a26%5C_nvidia-omniverse-multi-user-collaboration-2/nvidia-omniverse-multi-user-collaboration-2%5C_850cda5e-1521-480f-86a7-5d4a21400be5-prv.png. NVIDIA Press Kit.

28. *Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/kit/docs/kit-manual/latest/guide/kit%5C_overview.html. NVIDIA Omniverse Documentation.

29. *Scripting*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/kit/docs/kit-manual/latest/guide/python%5C_scripting.html. NVIDIA Omniverse Documentation.

30. *Omniverse RTX Renderer Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/materials-and-rendering/latest/rtx-renderer%5C_overview.html. NVIDIA Omniverse Documentation.

31. LLAMAS, Ignacio. *Omniverse RTX Real-Time Ray Tracer*. NVIDIA, 2019. Dostupné také z: https://on-demand.gputechconf.com/siggraph/2019/pdf/sig916-omniverse-rtx-real-time-ray-tracer.pdf.

32. *Simulation*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/platform/latest/simulation.html. NVIDIA Omniverse Documentation.

33. *Extensions in-depth*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/kit/docs/kit-manual/latest/guide/extensions%5C_advanced.html. NVIDIA Omniverse Documentation.

34. *Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/kit/docs/omni.ui/latest/Overview.html. NVIDIA Omniverse Documentation.

35. *UI Window*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/workflows/latest/extensions/ui%5C_window%5C_tutorial.html. NVIDIA Omniverse Documentation.

36. *Building an App*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/kit/docs/kit-manual/105.1/guide/creating%5C_kit%5C_apps.html. NVIDIA Omniverse Documentation.

37. *Omniverse Glossary of Terms*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/platform/latest/common/glossary-of-terms.html. NVIDIA Omniverse Documentation.

38. *Omniverse Kit App Template*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/kit/docs/kit-app-template/latest/intro.html. NVIDIA Omniverse Documentation.

39. *USD Presenter Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/presenter/latest/index.html. NVIDIA Omniverse Documentation.

40. *USD Composer Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/composer/latest/index.html. NVIDIA Omniverse Documentation.

41. *Code Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/code/latest/index.html. NVIDIA Omniverse Documentation.

42. *Launcher Overview*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/launcher/latest/index.html. NVIDIA Omniverse Documentation.

43. *Digital Assets*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/digital-twins/latest/building-full-fidelity-viz/digital-assets.html. NVIDIA Omniverse Documentation.

44. *NVIDIA Robotics*. NVIDIA, 2024. Dostupné také z: https://s3.amazonaws.com/cms.ipressroom.com/219/files/20240/659a52363d6332fd16032963%5C_nvidia-robotics-image/nvidia-robotics-image%5C_c5b3d5ea-3f12-4933-9fee-29bf10d1ec69-prv.jpg. NVIDIA Press Kit.

45. *Movie Capture for View*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/extensions/latest/ext%5C_movie-capture%5C_view.html. NVIDIA Omniverse Documentation.

46. *Asset Converter*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/extensions/latest/ext%5C_asset-converter.html. NVIDIA Omniverse Documentation.

47. *Package App*. NVIDIA, 2023. Dostupné také z: https://docs.omniverse.nvidia.com/kit/docs/kit-app-template/latest/packaging%5C_app.html. NVIDIA Omniverse Documentation.

48. *Technical Requirements*. NVIDIA, 2024. Dostupné také z: `https://docs.omniverse.nvidia.com/platform/latest/common/technical-requirements.html`. NVIDIA Omniverse Documentation.

Struktura ZIP souboru

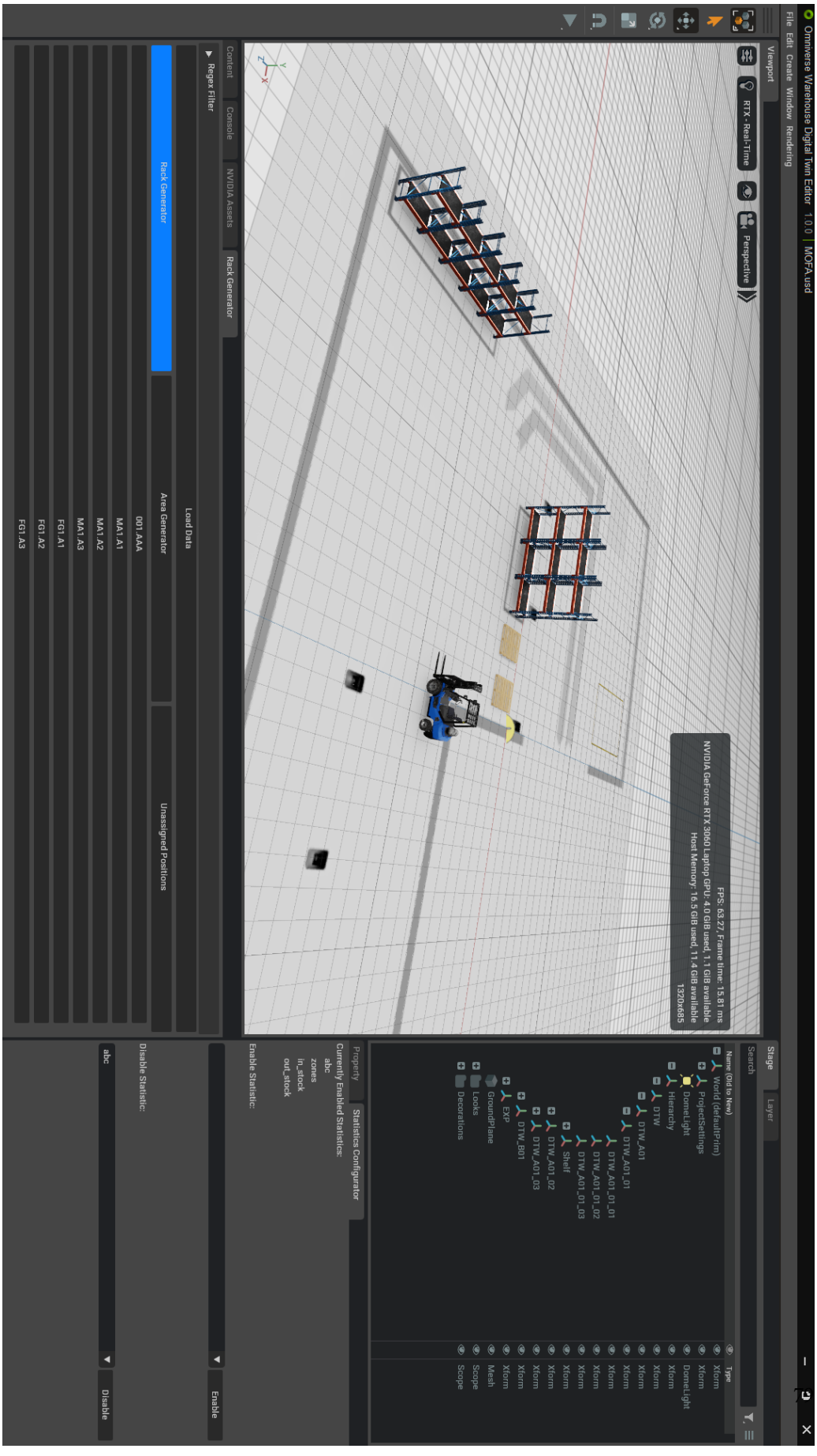
Práce je uložena v `ZIP` souboru s následující adresářovou strukturou:

- `Text_prace` - adresář obsahující text práce a jeho zdrojové kódy
  - `Text` - podadresář odsahující `PDF` soubor s textem práce
  - `Zdrojove_kody` - podadresář obsahující zdrojové kódy textu v LaTeXu
- `Aplikace_a_knihovny` - adresář obsahující vytvořený systém pro tvorbu digitálních dvojčat
  - `OmniverseWarehouseDigitalTwin` - kořenový adresář systému, jehož struktura byla detailně popsána v sekci 7.3
- `Readme.txt` - soubor popisující adresářovou strukuturu `ZIP` archivu

Screenshoty aplikací

B

Na následujících stranách se nacházejí obrázky B.1 a B.2 obsahující snímky obrazovky výsledných aplikací *Editor* a *Viewer*.



Obrázek B.1: Screenshot aplikace *Editor*

Obrázek B.2: Screenshot aplikace *Viewer*

Uživatelská příručka aplikace Editor

C

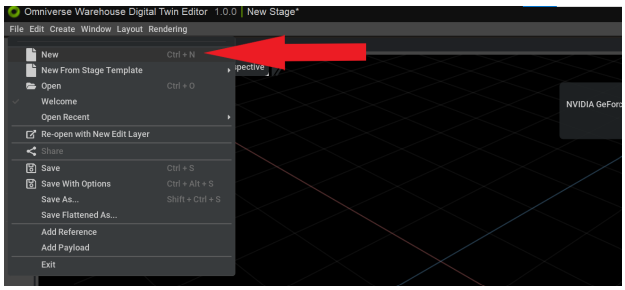
Tato uživatelská příručka provede uživatele tvorbou jednoduchého projektu v aplikaci *Editor*. Předpokladem pro tvorbu projektu je běžící back-end systému **GDTW** a databáze systému **Aimtec DCIx**.

C.1 Základní ovládání

- Ve scéně se můžete otáčet pomocí tažení myši se stisknutým pravým tlačítkem.
- Při držení pravého tlačítka myši se ve scéně zároveň můžete pohybovat pomocí kláves `W`, `S`, `A` a `D`.
- Pohyb můžete zrychlit či zpomalit podržením kláves `shift`, nebo `ctrl`.
- Pomocí levého tlačítka myši můžete volit objekty ve scéně a interagovat s prvky uživatelského rozhraní (například s gizmy).

C.2 Založení a inicializace projektu

1. Vytvořte novou scénu - v okně *Welcome*, které se zobrazí při spuštění aplikace nebo v menu horní lišty s cestou *File/New* (viz obrázek C.1).

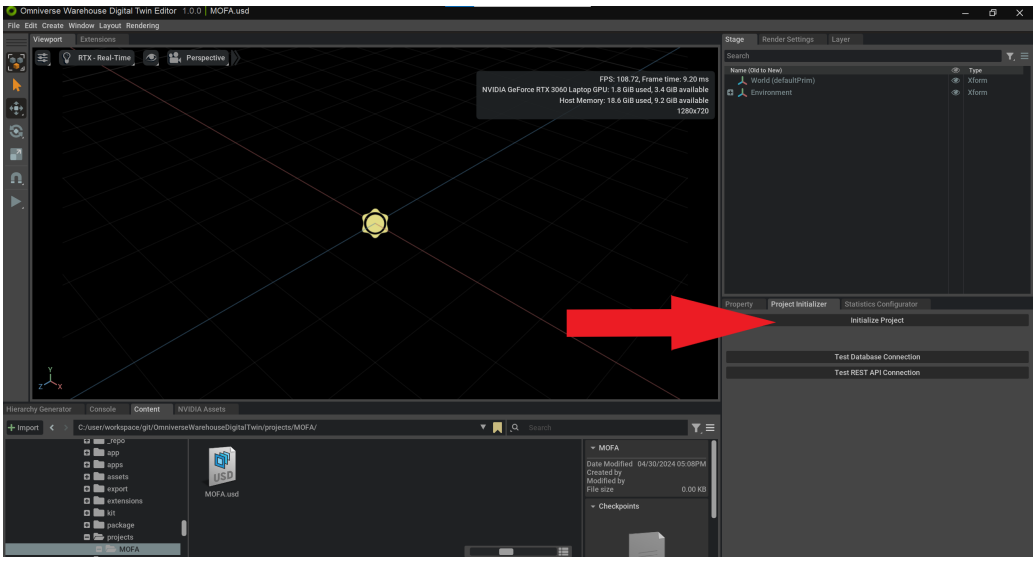


Obrázek C.1: Založení nového projektu

2. Ve složce `projects` vytvořte novou složku projektu s názvem shodným jako název projektu.

3. Uložte scénu v menu horní lišty s cestou *File/Save As...* do nově vytvořené složky projektu.

4. V okně uživatelského rozhraní *Project Initializer* zvolte *Initialize Project* (viz obrázek C.2). Vznikne základní hierarchie scény (včetně konfiguračních primitiv), jednotlivé *USD* vrstvy a složka `assets/textures` ve složce projektu.



Obrázek C.2: Inicializace projektu

C.3 Konfigurace komunikace s externím systémem

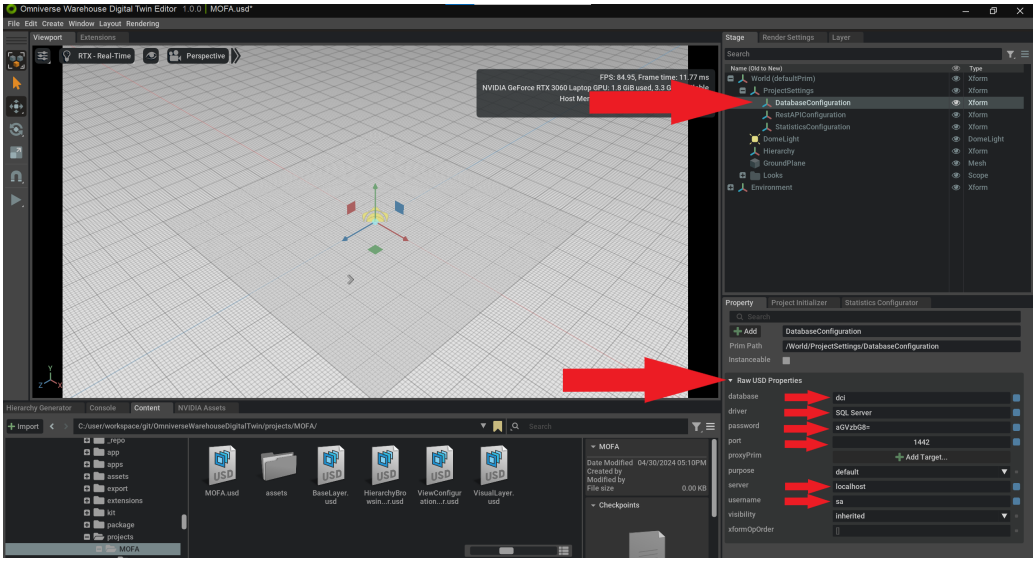
1. V okně *Stage* zvolte primitivum s cestou `World/ProjectSettings/DatabaseConfiguration`.

2. V okně *Property* rozbalte frame s popiskem *Raw USD Properties*.

3. Vyplňte hodnoty konfigurace komunikace s databází (viz obrázek C.3):

   - driver - název ovladače
   - server - URL databázového serveru
   - port - port komunikace s databázovým serverem
   - database - název databáze

- username - uživatelské jméno
- password - heslo šifrované pomocí base64



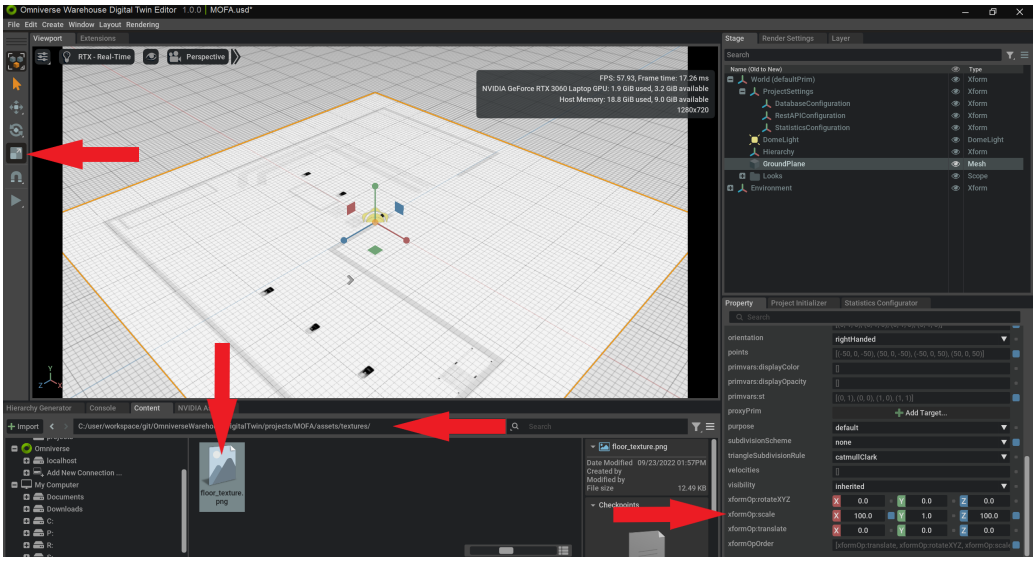
Obrázek C.3: Konfigurace databázového připojení

4. Připojení můžete otestovat pomocí tlačítka *Test Database Connection*.
5. Stejné kroky opakujte i pro primitivum s cestou `World/ProjectSettings/-RestAPIConfiguration`. Zde vyplňte hodnoty konfigurace komunikace s *REST API* back-endu systému **GDTW**:
   - serverUrl - URL serveru
   - apiUrlContext - kontext, pod kterým je dostupné *API* (při výchozí konfiguraci **GDTW** hodnota "`api`")
   - projectCode - název projektu (projekt musí na back-endu existovat)
   - connectionType - určuje, odkud bude **GDTW** načítat data (nejčastěji hodnota "`dci`")
   - username - uživatelské jméno
   - password - heslo šifrované pomocí base64

C.4 Úprava podkladové desky

- Do složky `assets/textures` vložte texturu pro podkladovou desku pod názvem `floor_texture.png` (viz obrázek C.4). Textura se pro podkladovou desku automaticky použije.

- Velikost desky dále můžete upravit pomocí gizma nebo přímou úpravou atributů v okně uživatelského rozhraní *Property* (viz obrázek C.4).



Obrázek C.4: Úprava podkladové desky

C.5 Generování skladové hierarchie

C.5.1 Generování regálů

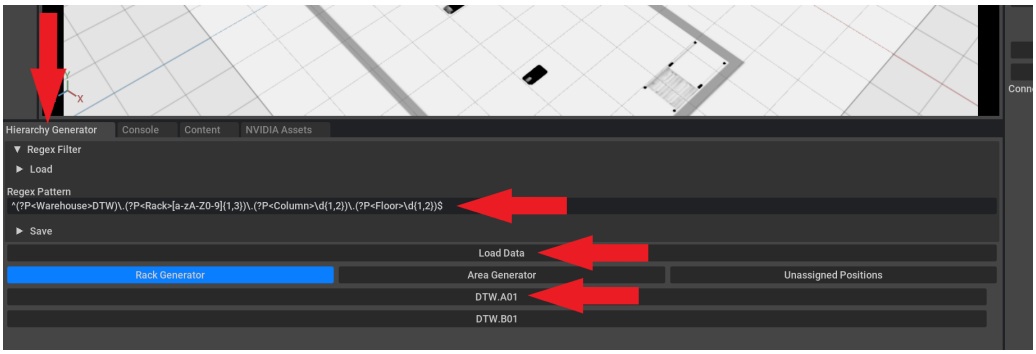
1. Pro generování skladové hierarchie využijte okno uživatelského rozhraní s názvem *Hierarchy Generator* (viz obrázek C.5).

2. V tomto okně můžete nastavit regulární výraz pro filtrování pozic, či použít výchozí (viz obrázek C.5). Regulární výraz lze také uložit a zpětně načíst.

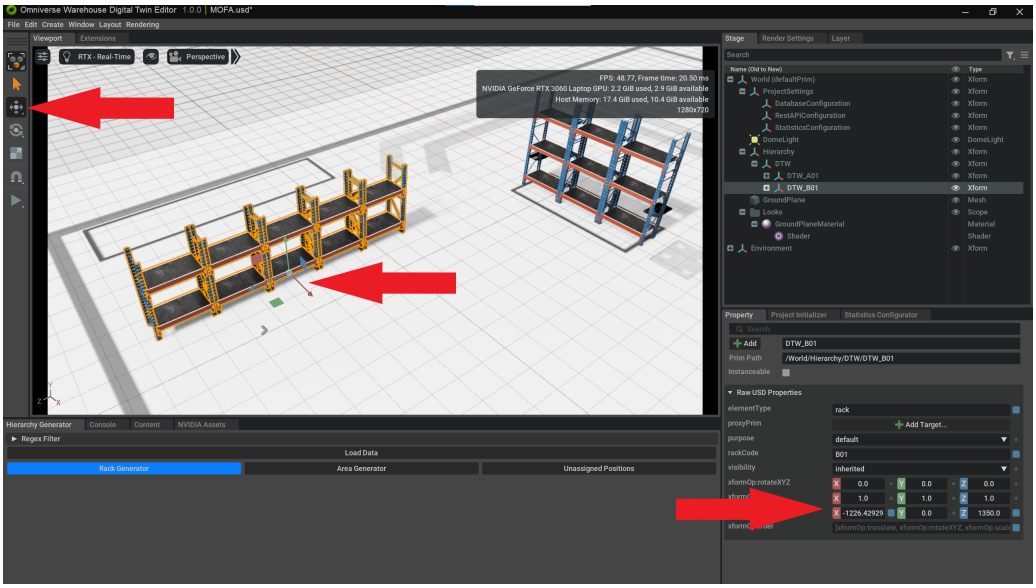
3. Pro vyfiltrování pozic stiskněte tlačítko *Load Data* (viz obrázek C.5).

4. Regál následně vygenerujete stisknutím tlačítka s kódem regálu (viz obrázek C.5).

5. Regál se vygeneruje uprostřed scény a pomocí gizma či v okně uživatelského rozhraní *Property* ho můžete umístit kamkoliv do scény (viz obrázek C.6).



Obrázek C.5: Generování regálu



Obrázek C.6: Umístění regálu

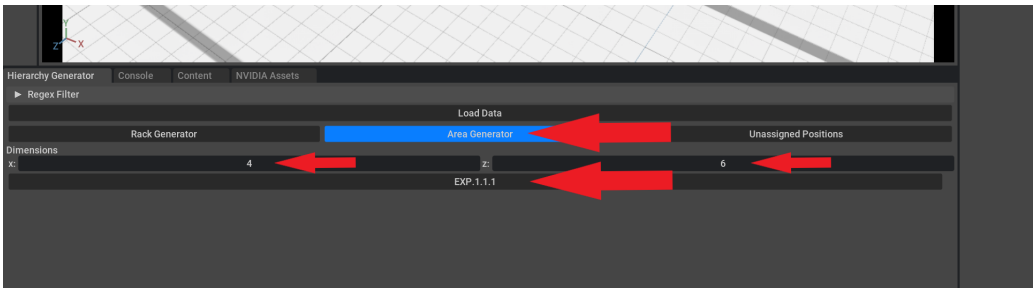
C.5.2 Generování skladových ploch

Postup generování skladových ploch je prakticky identický s generováním regálů. Rozdíly jsou pouze následující:

1. Před stiskem tlačítka *Load Data* je nutné zvolit možnost *Area Generator* (viz obrázek C.7).

2. Poté lze upravit velikost vygenerované plochy pomocí dvou polí (viz obrázek C.7).

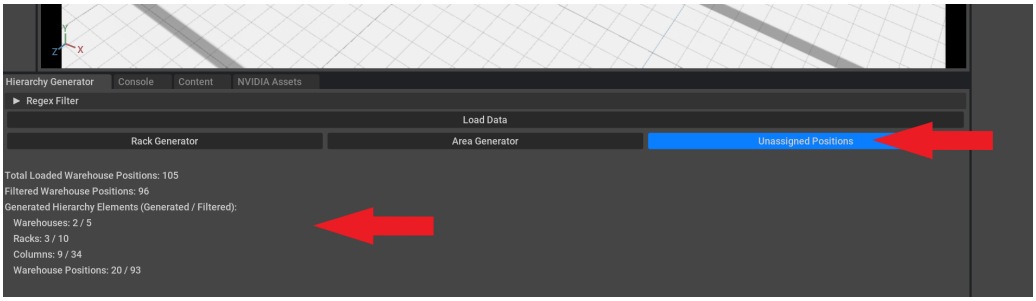
3. Následně je možné vygenerovat skladovou plochu stiskem tlačítka s jejím kódem a umístit na požadované místo.



Obrázek C.7: Generování skladové plochy

C.5.3 Statistiky vygenerované hierarchie

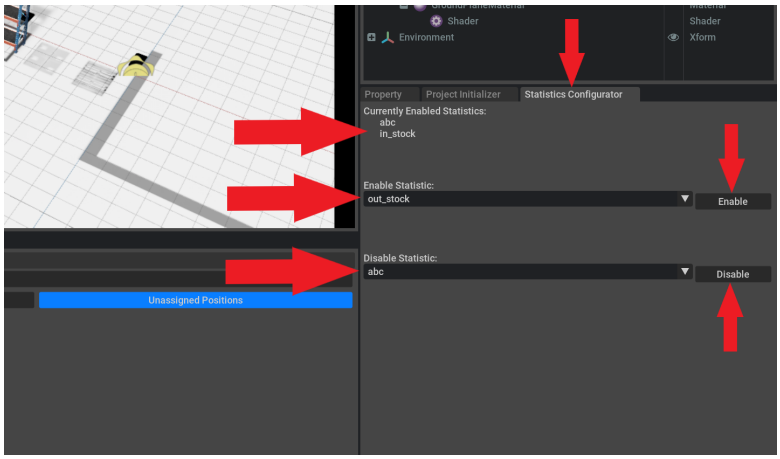
V uživatelském rozhraní je k dispozici ještě třetí možnost - *Unassigned Positions*, kde si lze zobrazit statistiky o načtené a vygenerované hierarchii (viz obrázek C.8).



Obrázek C.8: Statistiky o vygenerované hierarchii

C.6 Konfigurace statistik

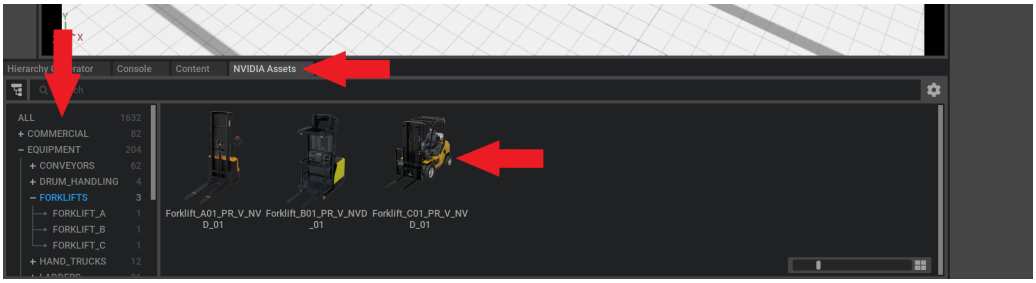
- Pro úpravu dostupných statistik použijte uživatelské rozhraní s názvem *Statistics Configurator*.
- V horní části uživatelského rozhraní se zobrazuje seznam aktivovaných statistik.
- Statistiky můžete přidávat či odebírat pomocí zvolení v příslušném comboboxu a stisknutí tlačítka *Enable* či *Disable*.



Obrázek C.9: Konfigurace statistik

C.7 Přidání dekorací

- Pro přidání dekorací do scény slouží uživatelské rozhraní s názvem *NVIDIA Assets* (viz obrázek C.10).
- V levé části rozhraní můžete dostupné assety filtrovat podle kategorií (viz obrázek C.10).
- Dekoraci do scény přidáte přetažením myší.
- Pozici a rozměry dekorace lze opět upravit pomocí gizma či v uživatelském rozhraní *Property*.



Obrázek C.10: Přidání dekorací

C.8 Uložení

- Po dokončení práce na projektu nezapomeňte projektu uložit v menu pod cestou *File/Save* nebo použitím klávesové zkratky `ctrl + S`.

- Rozpracovaný projekt vždy můžete otevřít z menu pod cestou *File/Open* nebo z okna *Welcome*, které se zobrazí při spuštění aplikace.

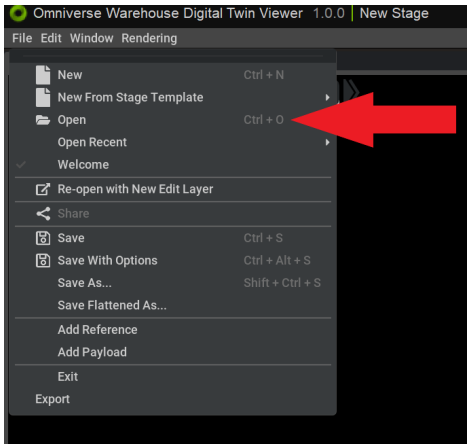
Uživatelská příručka aplikace Viewer

D

Tato uživatelská příručka popisuje ovládání aplikace *Viewer*. Předpokladem správné funkcionality je běžící back-end systému **GDTW** a správně nastavený projekt.

D.1 Načtení projektu

- Projekt můžete načíst z okna uživatelského rozhraní s názvem *Welcome*, nebo z menu pod cestou *File/Open* (viz obrázek D.1).



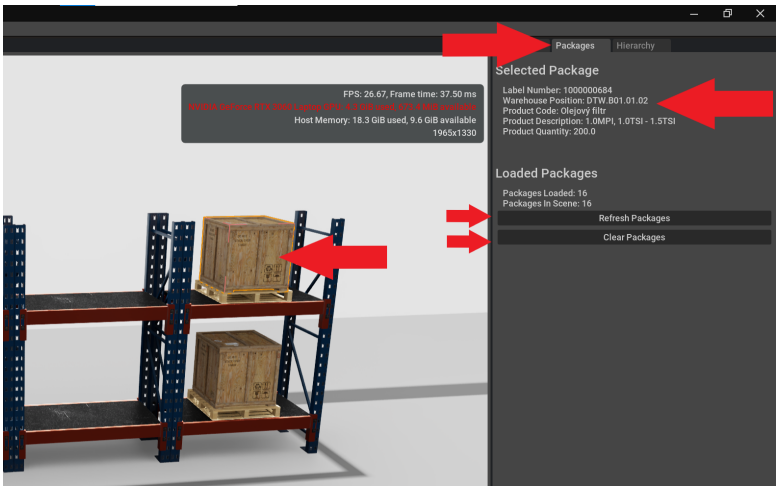
Obrázek D.1: Načtení projektu

D.2 Základní ovládání

- Ve scéně se můžete otáčet tažením myši se stisknutým pravým tlačítkem.
- Při držení pravého tlačítka myši se ve scéně zároveň můžete pohybovat pomocí kláves `W`, `S`, `A` a `D`.
- Pohyb můžete zrychlit či zpomalit podržením kláves `shift`, nebo `ctrl`.

D.3 Balení

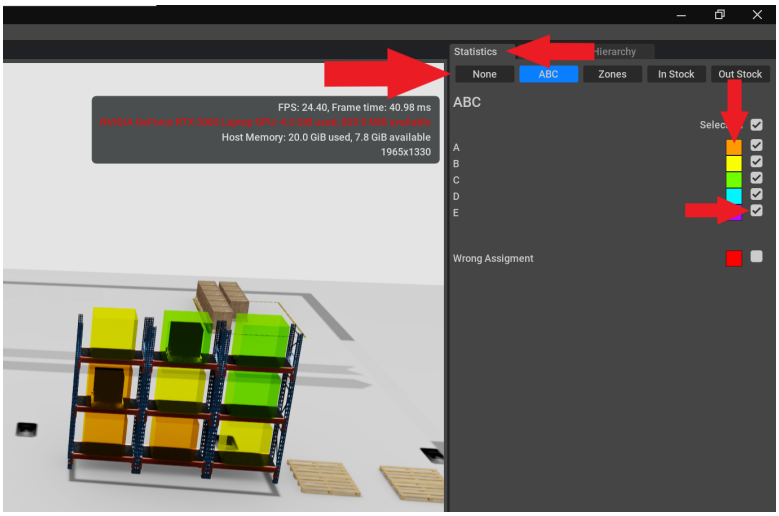
- Balení jsou načtena automaticky po načtení projektu.
- Kliknutím na objekt balení ve scéně levým tlačítkem myši se vám v horní části okna uživatelského rozhraní *Packages* zobrazí informace o daném balení (viz obrázek D.2).
- Pomocí tlačítek *Refresh* a *Clear* v dolní části uživatelského rozhraní můžete balení znovu načíst či je skrýt (viz obrázek D.2).



Obrázek D.2: Balení

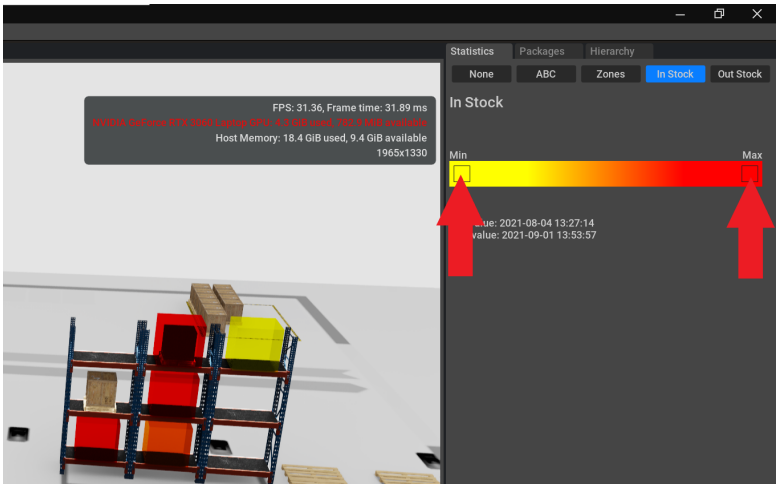
D.4 Statistické veličiny

- Do modelu si pomocí uživatelského rozhraní s názvem *Statistics* můžete zobrazovat statistické veličiny.
- V horní liště vyberte požadovanou statistiku, tím jí zobrazíte zobrazí do modelu (viz obrázek D.3).
- U statistik typu skupiny pozic můžete pomocí checkboxů upravovat, které skupiny pozic budou v modelu zobrazeny. Pomocí colorpickerů můžete měnit barvy skupin pozic. (viz obrázek D.3)



Obrázek D.3: Statistiky typu skupiny pozic

- U statistik typu heatmapy můžete upravovat barvy minimální a maximální hodnoty pomocí colorpickerů na krajích barevného gradientu. (viz obrázek D.4)



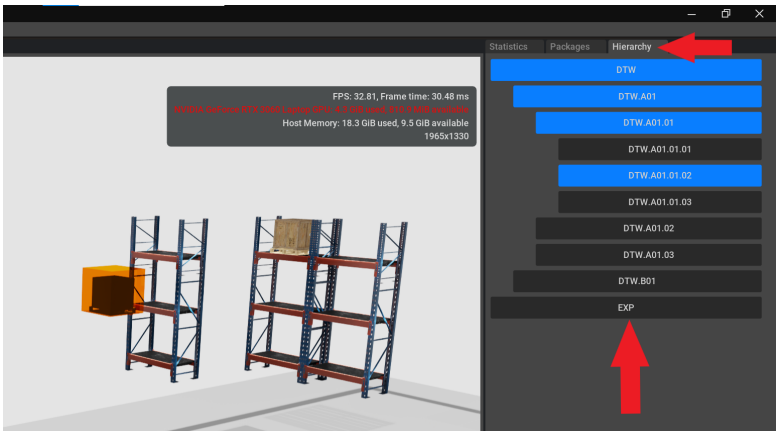
Obrázek D.4: Statistiky typu heatmapy

- Statistiku můžete skrýt zvolením možnosti *None* v horní liště.

D.5 Procházení skladové hierarchie

- Pomocí okna uživatelského rozhraní s názvem *Hierarchy* můžete procházet skladovou hierarchii projektu.

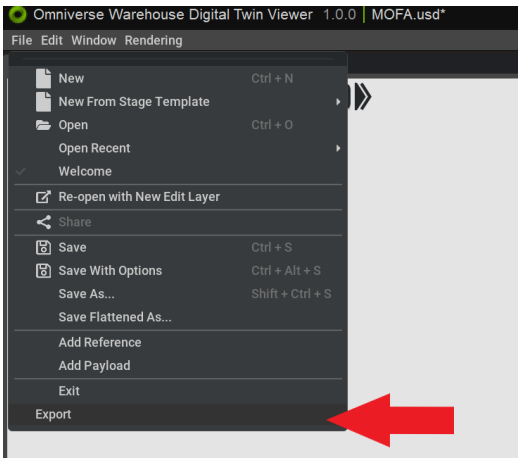
- Skladovou hierarchii rozbalujete klikáním na tlačítka s kódy jednotlivých objektů (viz obrázek D.5).
- Kliknutím na tlačítko již rozbaleného (modře vyznačeného) objektu ho zpět jeho potomky zpět sbalíte.
- procházení hierarchie je doprovázeno animací zvoleného objektu ve scéně.



Obrázek D.5: Procházení skladové hierarchie

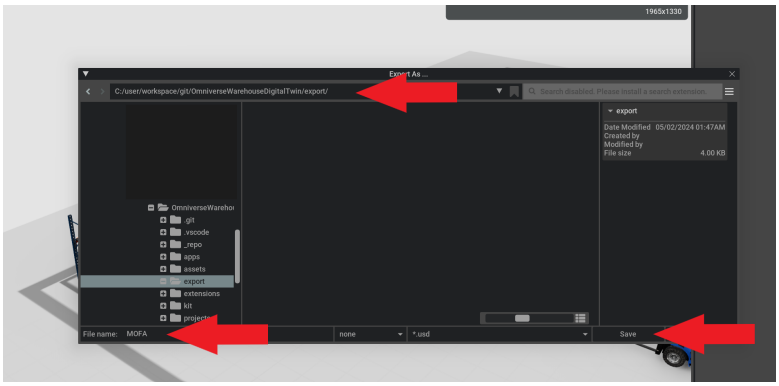
D.6 Export

- Aktuální stav 3D modelu můžete exportovat v menu pod cestou *File/Export* (viz obrázek D.6).



Obrázek D.6: Export

- Zobrazí se vám okno ve kterém specifikujete název a umístění exportovaného souboru (viz obrázek D.7).
- Po kliknutí na tlačítko *Save* (viz obrázek D.7) je model vyexportován.



Obrázek D.7: Určení umístění exportovaného souboru