FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Parametrické modelování halových konstrukcí
Student:	Ing. Robin Blažek
Vedoucí:	Ing. Pavel Štěpán
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Tekla Structures je desktopová aplikace umožňující 3D modelování stavebních konstrukcí. Lze v ní kombinovat různé stavební materiály, vytvářet konstrukční detaily, nebo umisťovat výztuž do stavebních prvků. Jedním z nejčastějších případů použití je tvorba konstrukčních detailů ocelových hal. Uživatel v takovém případě dostane zadání základních parametrů konstrukce, většinou od architekta či projektanta, a v následující fázi také vypočítané a ověřené návrhy ocelových spojů. Modelování těchto detailů je hlavní přidaná hodnota, kterou uživatel do projektu vnáší. Aplikace nabízí několik vestavěných maker, která tuto práci urychlují. Naopak sestavení základního 3D rastru konstrukce je proces, který je manuálně relativně pracný a zdlouhavý.

Cílem práce je využít API možností aplikace Tekla Structures pro generování základního rastru typických halových konstrukcí. Aplikace bude na vstupu od uživatele přijímat několik základních nezbytných údajů pro sestavení konstrukce (např. rozpětí haly, rozpětí mezi vazbami, výška sloupů atd.) a na jejich základě bude následně možné konstrukci vygenerovat přímo do 3D scény aplikace Tekla Structures. Výsledkem pro uživatele tak bude úspora času stráveného ručním modelováním a možnost se zaměřit na detaily spojů.

- proveďte rešerši současných řešení pro podobný typ problému
- na základě schopností API vyberte vhodné vstupní parametry pro generování halových konstrukcí

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 19. září 2023 v Praze.

- vyberte vhodnou architekturu a technologie pro danou aplikaci
- Vytvořte základní návrh uživatelského rozhraní tak, aby umožňoval uživateli přehledně a jednoduše zadat potřebné parametry
- implementujte prototyp aplikace a proveďte uživatelské testování

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 19. září 2023 v Praze.

Bakalářská práce

PARAMETRICKÉ MODELOVÁNÍ HALOVÝCH KONSTRUKCÍ

Ing. Robin Blažek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Pavel Štěpán
20. prosince 2023

Obsah

Chtěl bych poděkovat především Ing. Pavlu Štěpánovi za jeho odborné vedení a užitečné rady při zpracování bakalářské práce.

Prohlášení

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

V Praze dne 20. prosince 2023

Abstrakt

Tato bakalářská práce se zabývá návrhem aplikace sloužící k parametrickému modelování halových konstrukcí v inženýrském softwaru Tekla Structures. V teoretické části práce je provedena analýza dostupných aplikací, které jsou určeny k parametrickému vytváření stavebních modelů. Výstupem práce je funkční prototyp desktopové aplikace, který umožňuje na základě zadaných vstupních dat od uživatele vygenerovat hlavní rastr stavebního modelu přímo ve 3D scéně programu Tekla Structures. Navržené uživatelské prostředí přispívá k urychlení práce při zadávání potřebných parametrů. Implementace využívá platformu .NET Framework v kombinaci s MVVM architekturou, díky které má aplikace potenciál pro budoucí rozšíření v podobě napojení jiného stavebního modelovacího softwaru. Spojení s programem Tekla Structures je realizováno pomocí jeho veřejného API.

Klíčová slova desktopová aplikace, Tekla Structures, halové konstrukce, parametrické modelování, 3D strukturální model, .NET Framework

Abstract

This bachelor thesis deals with the design of an application for parametric modelling of hall structures in the engineering software Tekla Structures. In the theoretical part of the thesis a research of available applications that are designed for parametric creation of building models is made. The output of the thesis is a functional prototype of a desktop application that allows to generate the main grid of the building model directly in the 3D scene of Tekla Structures software based on the input data provided by the user. The proposed user interface contributes to speed up the work when entering the necessary parameters. The implementation uses the .NET Framework platform in combination with the MVVM architecture, thanks to which the application has the potential for future extensions in the form of connecting other structural modelling software. The connection to Tekla Structures is implemented using its public API.

Keywords desktop application, Tekla Structures, hall structures, parametric modelling, 3D structural model, .NET Framework

Seznam zkratek

AEC	Architecture, Engineering And Construction
API	Application Programming Interface
BIM	Building Information Modelling
CIL	Common Intermediate Language
CLR	Common Language Runtime
DXF	Drawing Exchange Format
FEA	Finite Element Analysis
IFC	Industry Foundation Classes
MAUI	Multi-platform App User Interface
MVVM	Model–View–ViewModel
openBIM	Open Building Information Modelling
PDF	Portable Document Format
POC	Proof Of Concept
RAM	Random Access Memory
TS	Tekla Structures
UI	User Interface
UWP	Universal Windows Platform
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

Úvod

Přes nemalé zpoždění ve srovnání s ostatními obory, i stavebnictví přeci jen nakonec přijímá dobu digitální za svou. Architektonické studie se přesunuly z náčrtů do renderovaných vizualizací, analýzu konstrukce si dnes již jen málokdo představí bez pokročilých FEA aplikací a projekční papírové výkresy jsou nahrazovány PDF soubory. Digitalizace tohoto odvětví byl a stále je proces nevyhnutelný, byť zdaleka ne jednoduchý. Snahy standardizovat výměnu potřebných dat zatím nevedly k jednoznačnému úspěchu, a tak je situace v tomto směru stále otevřená.

Jednoho společného jmenovatele napříč jednotlivými řešeními však nalézt lze. Je jím centrální 3D datový model budovy, konstrukce, či jakéhokoli stavebního objektu. Model, který je během celého životního cyklu budovy plněn daty. Tyto informace mohou využívat a doplňovat jednotlivé profese, které s modelem potřebují pracovat. Ty v některých případech musí použít svůj vlastní, určitým způsobem specifický, druh modelu (např. statický model pro analýzu konstrukce). Centrální model ale mohou použít jako zdroj potřebných dat pro tvorbu svého submodelu. Následně mohou na základě svých výpočtů centrální model doplnit o další data.

Ve snaze sjednotit formát přenášených informací a minimalizovat opakovanou tvorbu téhož modelu byl zaveden formát IFC. Většina dnešních softwarů specializovaných na AEC tento formát v nějaké míře podporuje. Problémem však zůstává kvalita výstupů, problémy s kompatibilitou napříč jednotlivými verzemi a v neposlední řadě také chyby z neznalosti na straně inženýrů, kteří se málokdy zabývají nastavením exportu. Obzvláště pak v menších projekčních, architektonických či statických kancelářích se lze stále setkat s tím, že se veškerá dokumentace tvoří pouze ve 2D, tedy bez jakéhokoli 3D modelu, natož sdíleného.

Opětovným modelováním částí konstrukce tak ztrácí čas mnoho stavebních inženýrů s různou specializací. Jedním z mnoha konkrétních příkladů je ten, kdy stavební inženýr na pozici modeláře používá software Tekla Structures a jeho úkolem je namodelovat konstrukční detaily spojů na halové konstrukci. V ideálním případě by takový specialista obdržel jako podklad 3D stavební model konstrukce v nějakém univerzálním formátu, který by mohl použít ve svém softwaru. Tekla Structures umožňuje například import formátu IFC. Jak však bylo zmíněno výše, tento scénář ještě není samozřejmou realitou a obzvláště pak v českém prostředí je stále běžné, že takovýto 3D model není k dispozici. Jako podkladem mu tak mnohdy slouží PDF výkresová dokumentace se základními parametry konstrukce jako rozpětí halových rámů, výšky sloupů, použité profily a další.

Tato bakalářská práce se zabývá tvorbou aplikace, která má v tomto případě dotyčnému modeláři co nejvíce práci usnadnit, pokud možno urychlit a umožnit mu soustředit se na jeho hlavní náplň, tedy na modelování spojů.

Cílem bakalářské práce je vytvořit prototyp aplikace, která uživateli umožní na základě zadaných parametrů vygenerovat halovou konstrukci v softwaru Tekla Structures. Mezi dílčí cíle patří analyzovat dostupné aplikace, které slouží k parametrickému modelování halových stavebních konstrukcí nebo k parametrickému modelování v aplikaci Tekla Structures. Dalším cílem

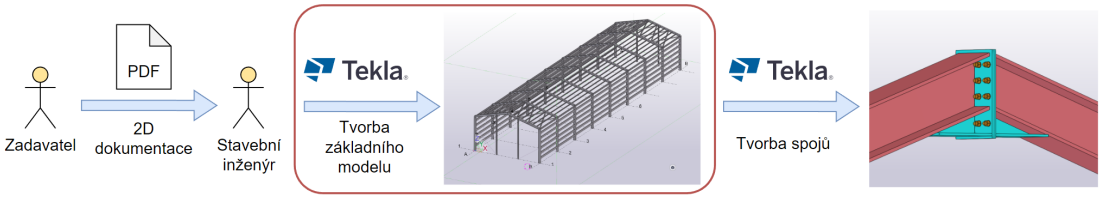
je vybrat vstupní parametry potřebné pro vygenerování konstrukce v závislosti na možnostech API aplikace Tekla Structures a zvolit vhodné implementační technologie. Dílčím cílem je také otestovat vytvořený prototyp na uživatelské úrovni.

Kapitola 1
Kontext problematiky

Kapitola navazuje na myšlenky zmíněné v úvodu a podrobněji zasazuje téma bakalářské práce do kontextu konkrétního pracovního procesu. Součástí je definování uživatele, pro kterého bude aplikace navržena, a také stručný popis aplikace Tekla Structures.

1.1 Cílový uživatel

Cílovými uživateli tvořené aplikace jsou pracovníci projekčních kanceláří, kteří se specializují na vytváření 3D stavebních modelů se zaměřením na detaily spojů v aplikaci Tekla Structures. Dotyčným bývá v nejčastějším případě stavební inženýr nebo modelář pod inženýrským dohledem. Forma potřebných informací, které tito pracovníci dostávají jako vstup pro svou práci, se napříč jednotlivými společnostmi a prostředími velmi výrazně liší. Situace v tuzemsku je pak taková, že zde existuje velmi mnoho menších architektonických studií, statických nebo projekčních kanceláří, které v tomto procesu figurují jako tvůrci zadání. A právě tyto menší subjekty často neposkytují zadání ve formě 3D stavebního modelu. Naopak daleko častější je varianta, že jako podklad se napříč projektem pro jednotlivé profese používá 2D výkresová dokumentace.

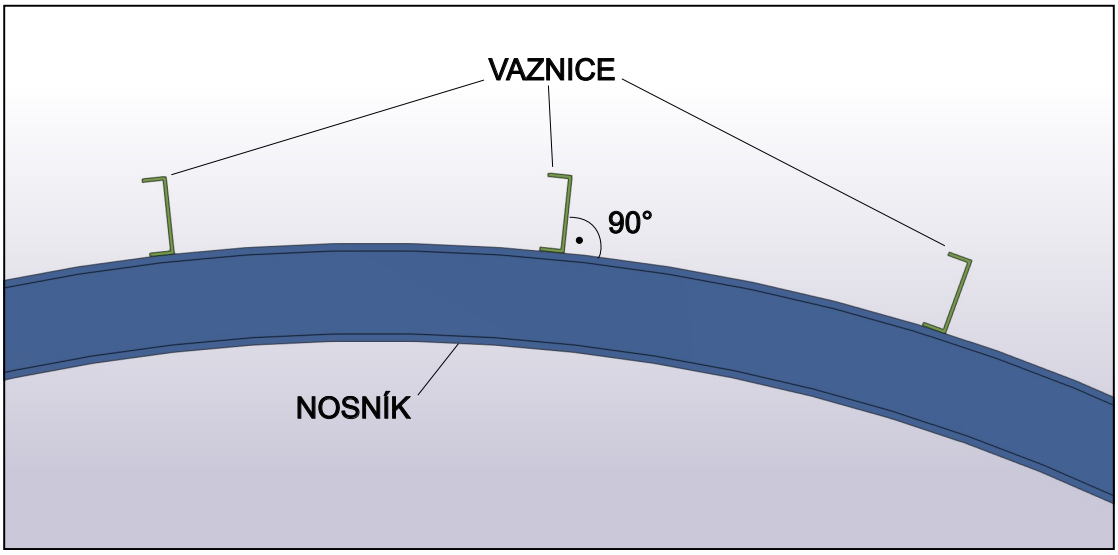


Obrázek 1.1 Schéma dotčeného pracovního procesu

Pro cílového uživatele to pak znamená, že si musí nejdříve ručně namodelovat základní skelet konstrukce, a až poté se může začít věnovat detailnímu modelování spojů (včetně šroubů, svarů, plechů, apod.). Schéma pracovního procesu lze vidět na obrázku 1.1 i s vyznačenou problematickou částí.

1.2 Hlavní přínosy navrhovaného řešení

Tvorba 3D modelu halové konstrukce v aplikaci Tekla Structures může být časově náročná. Běžným postupem je nejdříve vymodelování hlavního rámu. Následuje jeho zkopírování tak, aby bylo vytvořeno první pole konstrukce. A poté dochází k modelování podružných prvků, tedy jednotlivých vaznic a paždíků (prvky v bočních stěnách). Obzvláště u vaznic (spojující prvky ve střešní rovině) je ruční modelování relativně náročné. Důvodem je to, že průřez těchto nosníků musí být natočen tak, aby svíral s hlavními střešními nosníky pravý úhel (obrázek 1.2). Jejich rozmístění se navíc většinou měří v této nakloněné střešní rovině, tudíž ani kopírování není jednoduché. Nejvíce se pak tento vliv projeví u hal s obloukovým střešním nosníkem. I zde je totiž nutné dodržet podmínku kolmosti průřezu vaznice na rovinu nosníku. Vzhledem k zakřivené geometrii je však hodnota pootočení pro každou vaznici odlišná. Díky navrhované aplikaci se uživatel nebude muset zabývat ručním výpočtem úhlů v jednotlivých místech ani správným rozmístěním. Pouze zadá požadavek na zakřivený typ střešní konstrukce, určí si osovou vzdálenost mezi jednotlivými vaznicemi měřenou na zakřivené trajektorii nosníku a konstrukce se sama vygeneruje se správně pootočenými a umístěnými prvky.

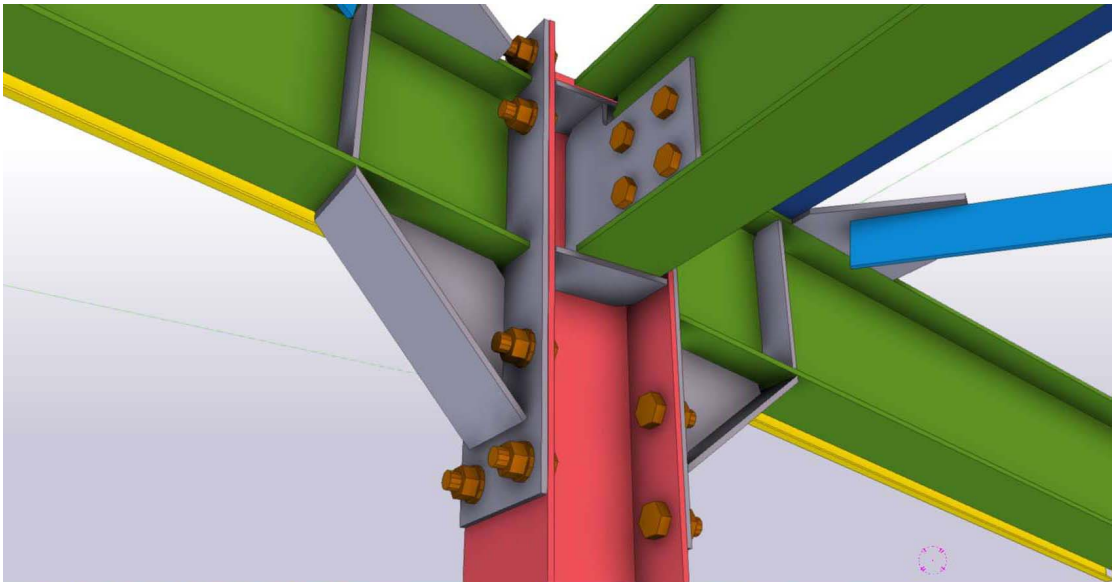


Obrázek 1.2 Uložení vaznice na zakřivený nosník

1.3 Tekla Structures

Trimble Inc. je mezinárodní společnost původem z Finska, která se zabývá technologickými a softwarovými řešeními v oblastech stavebnictví, zemědělství a dopravy. Jejich produkty se specializují na zjišťování a popis přesné polohy, zefektivnění komunikace a umožnění sdílení dat ve výše zmíněných odvětvích. [1]

Jejím hlavním produktem na poli stavebnictví je aplikace Tekla Structures. Ta umožňuje uživatelům vytvářet podrobné 3D digitální modely budov a konstrukcí. Jedná se o takzvané konstrukční modely, používané mimo jiné pro výkazy materiálů, tvorbu dokumentace nebo jako podklad pro další profese. Tekla Structures se také využívá pro návrh a analýzu různých částí stavebního projektu. Mezi nejsilnější stránky aplikace patří detailní modelování konstrukčních spojů (obrázek 1.3). Aplikace dnes podporuje nejrůznější materiály a jejich kombinace, ale své místo na trhu si historicky vydobyla především původním zaměřením na konstrukční spoje ocelových



Obrázek 1.3 Ukázka konstrukčního spoje ocelové konstrukce v Tekla Structures [2]

konstrukcí. Software je široce využíván architekty, inženýry, dodavateli a dalšími profesionály zapojenými do stavebního procesu. [3]

Uživatel vytváří modely buď manuálně přímo ve 3D scéně programu nebo může využít několika předpřipravených maker. Ta se ve většině případů specializují na zmiňované detaily a spoje, ale některá slouží i k tvorbě komplexnějších částí modelů (například dřevěný sbíjený příhradový vazník).

Tekla Structures je také součástí iniciativy openBIM, která se zaměřuje na proces spolupráce napříč stavebním projektem. Principem openBIM je zajištění spolehlivé výměny dat pomocí otevřených a neutrálních formátů. Nabízí tak dodavatelům flexibilitu při výběru konkrétní technologie a řešení. Projekt je realizován pod organizací buildingSMART International Ltd. [4]

Kapitola 2
Rešeršní část

Urychlení tvorby digitálního stavebního 3D modelu pomocí parametrického modelování začíná být v současné době stále častěji využíváno. Následující kapitola se věnuje analýze dostupných řešení, která lze pro tuto činnost použít. Součástí je přehled konkrétních aplikací, jejich stručný popis a specifikace zaměření.

2.1 Definice cílů a vytyčení rozsahu rešerše

Téma využití parametrického přístupu v procesu modelování stavebních konstrukcí je značně rozsáhlé. Proto byly pro rešerši vybrány následující dvě konkrétní kategorie aplikací.

- Parametrizace modelů pro Tekla Structures
  Do této kategorie jsou zařazeny taková řešení, která umožňují na základě zadaných parametrů vytvářet stavební model nebo jeho část v aplikaci Tekla Structures.
- Parametrické modelování halových konstrukcí
  Tato kategorie obsahuje řešení, která umožňují modelovat typické halové konstrukce na základě zadaných parametrů bez ohledu na použitý software na výstupu.

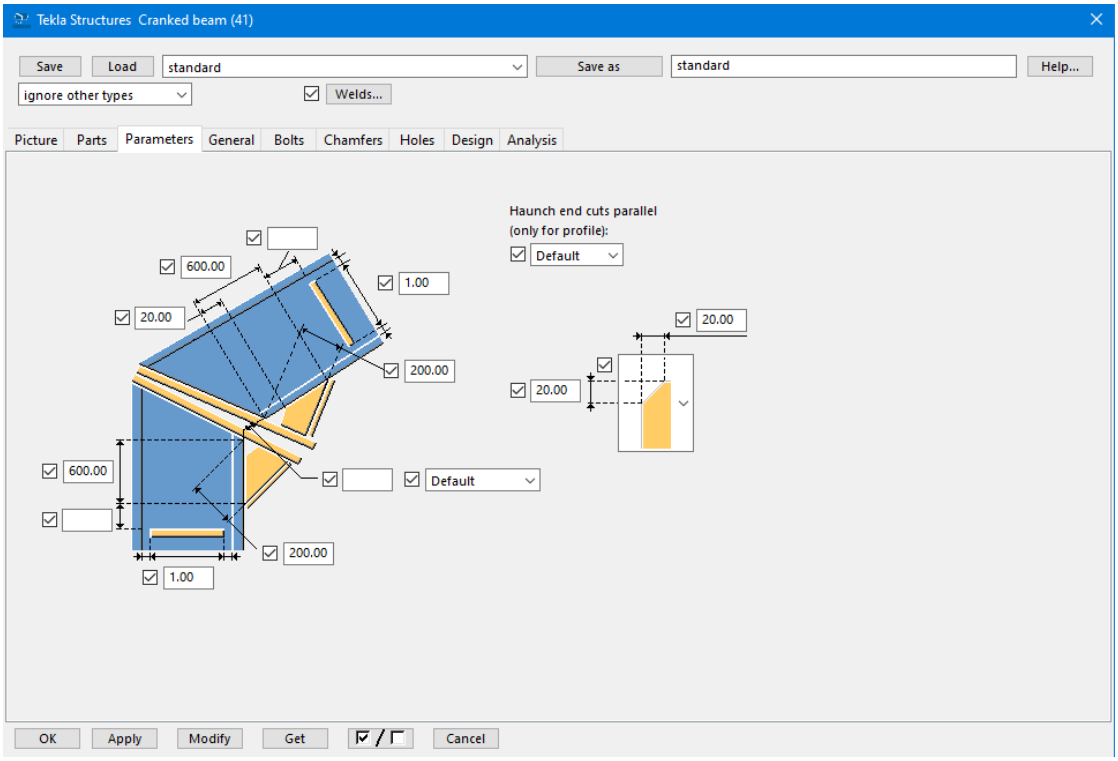
Cílem rešeršní části práce je prozkoumat stávající dostupné aplikace, které lze zařadit do výše uvedených dvou kategorií. Rešerše je zaměřena nejenom na popis těchto aplikací, ale také na prozkoumání použitých parametrů pro modelování halových konstrukcí. Důležitým aspektem je i uživatelské rozhraní jednotlivých aplikací. Tyto poznatky budou využity při návrhu vlastní aplikace v dalších kapitolách této práce.

2.2 Parametrizace modelů pro Tekla Structures

2.2.1 Vestavěná makra Tekla Structures

Jednou z nejsilnějších stránek aplikace Tekla Structures je komplexní modelování spojovacích detailů konstrukcí, zejména pak těch ocelových. Usazení vazníku na sloup, vaznic na vazníky, přípoje ztužidel nebo uložení paty sloupu, to všechno jsou typické detaily, které je nutné navrhnout pro téměř každý typ halové konstrukce. Aplikace obsahuje několik vestavěných maker, která modelování těchto objektů velmi usnadňují. V zásadě se jedná o předem připravené šablony skupin objektů pro jednotlivé typy styčníků, které lze na základě zadaných parametrů upravit dle potřeb uživatele.

Jak lze vidět na obrázku 2.1, prostředí šablony je tvořeno jedním oknem, které bývá logicky rozčleněno na několik záložek. Typicky se zde vyskytuje jedna záložka se statickým obrázkem jako náhledem detailu. Na dalších záložkách pak jsou rozmístěny jednotlivé buňky pro zadání vstupních parametrů. Rozložení a uspořádání jednotlivých vstupů se liší dle konkrétního detailu, nicméně celkové téma vzhledu je pro všechna makra stejné. Vzhledově i funkčností jej lze přirovnat k formulářovým aplikacím WinForms.



Obrázek 2.1 Ukázka makra v Tekla Structures [5]

Výhodou těchto maker je bez pochyby efektivita a srozumitelnost. Záložky šablony vedou uživatele jednotlivými kroky při zadávání parametrů tak, aby byly vyplněny všechny potřebné údaje a byly tak dodrženy zásady správného návrhu. Mezi negativa lze uvést z dnešního pohledu relativně nemoderní design UI. Především pak na obrazovkách s vyšším rozlišením se objevují problémy s nepřizpůsobováním velikosti fontů, buněk a celkově vzato chybným škálováním. U složitějších detailů, které potřebují definovat mnohdy i desítky parametrů, pak dochází ke ztrátě přehlednosti.

Tekla Structures umožňuje také třetím stranám vytvářet svá vlastní makra a integrovat je přímo do rozhraní aplikace mezi ta vestavěná. Na oficiálních webových stránkách je uveřejněna potřebná dokumentace a na portále Github pak lze nalézt ukázkové příklady se vzorovými prvky, které mohou vývojáři použít. [6]

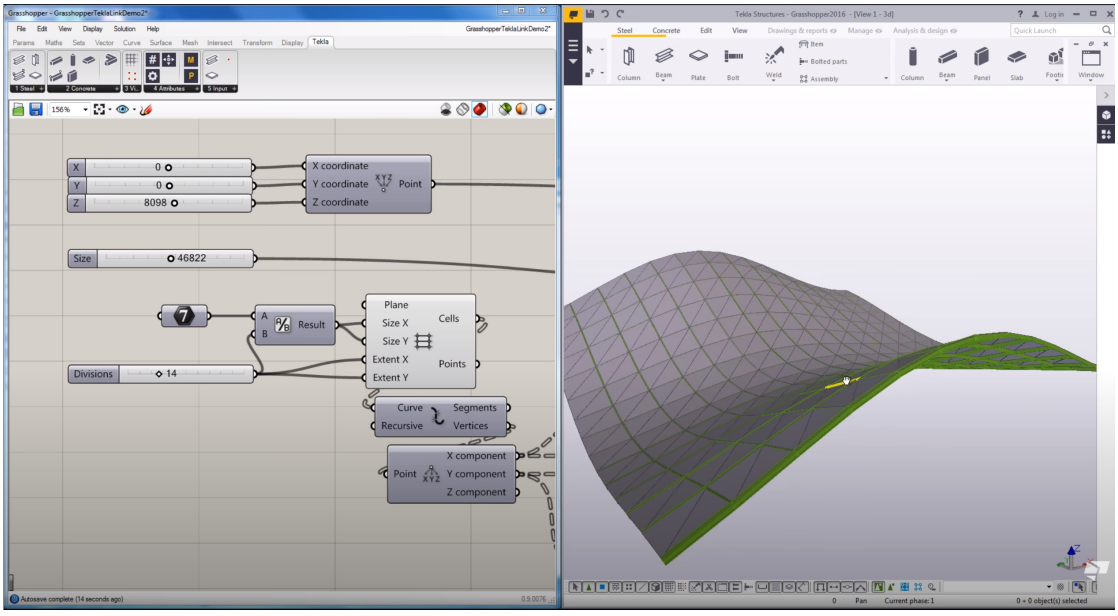
2.2.2 Grasshopper

Jedním z nejčastěji zmiňovaných důvodů pro parametrizaci modelování ve stavebním odvětví je optimalizace. Navrhnout co nejvhodnější tvar konstrukce, vybrat ten nejefektivnější profil, nebo vytvořit co nejvíce aerodynamickou obálku budovy. To všechno jsou typické příklady, které si vyžadují optimalizaci.

Téměř synonymem pro parametrické modelování ve stavebnictví za účelem optimalizace se stala aplikace Grasshopper. Jedná se o nástroj, který vznikl s účelem umožnit modelování a modifikaci tvarů popsaných generativními algoritmy. Rozšířil se natolik, že dnes je jeho využití velmi komplexní a je možné ho použít na téměř jakýkoli druh parametrizace. [7]

Původně byl velmi spjatý (dnes dokonce přímo integrovaný) s aplikací Rhinoceros 3D, která slouží ke komplexnímu 3D modelování objektů. Stal se ovšem tak oblíbený napříč uživateli, že dnes existují desítky pluginů, které zajišťují jeho komunikaci či integraci s mnoha dalšími softwary. Výjimkou není Tekla Structures. Ta má v nabídce svých oficiálních rozšíření modul s názvem Grasshopper-Tekla Live Link.

Tento nástroj pak nabízí uživateli samostatné grafické rozhraní, ve kterém lze kombinovat různé předpřipravené části a sestavit si parametricky model dle požadavků. Jedná se o takzvané *visual programming*, které umožňuje práci s parametry i bez toho, aniž by uživatel musel mít nějaké programátorské či skriptovací znalosti. Pracuje zde pouze s vizuálními objekty, které umisťuje na plochu, kde je spojuje a kombinuje dle jejich závislostí. Ukázku prostředí aplikace Grasshopper lze vidět v levé části obrázku 2.2. Veškerý kód a komunikace zajištěná přes API se odehrává na pozadí tak, aby se jí uživatel nemusel zabývat. To je jedna z největších výhod tohoto řešení a bez pochyby také jeden z důvodů, proč se Grasshopper těší mezi stavebními inženýry takové oblibě. [8]



Obrázek 2.2 Přímá interakce aplikací Grasshopper a Tekla Structures [8]

Kvalita finálního modelu jistě velmi závisí na zkušenostech uživatele. Mezi nevýhody lze zařadit značnou nepřehlednost pro komplexnější modely. S růstem počtu parametrů se začíná plocha s objekty zaplňovat, linie značící závislosti mají tendenci se častěji křížit a vyznat se v takovém modelu je pak mnohem náročnější. I přes to, že uživatel nemusí být programátor, je pro používání a správné modelování potřeba určitá znalost algoritmizace tvořící logiku a návaznosti jednotlivých objektů a parametrů. [9]

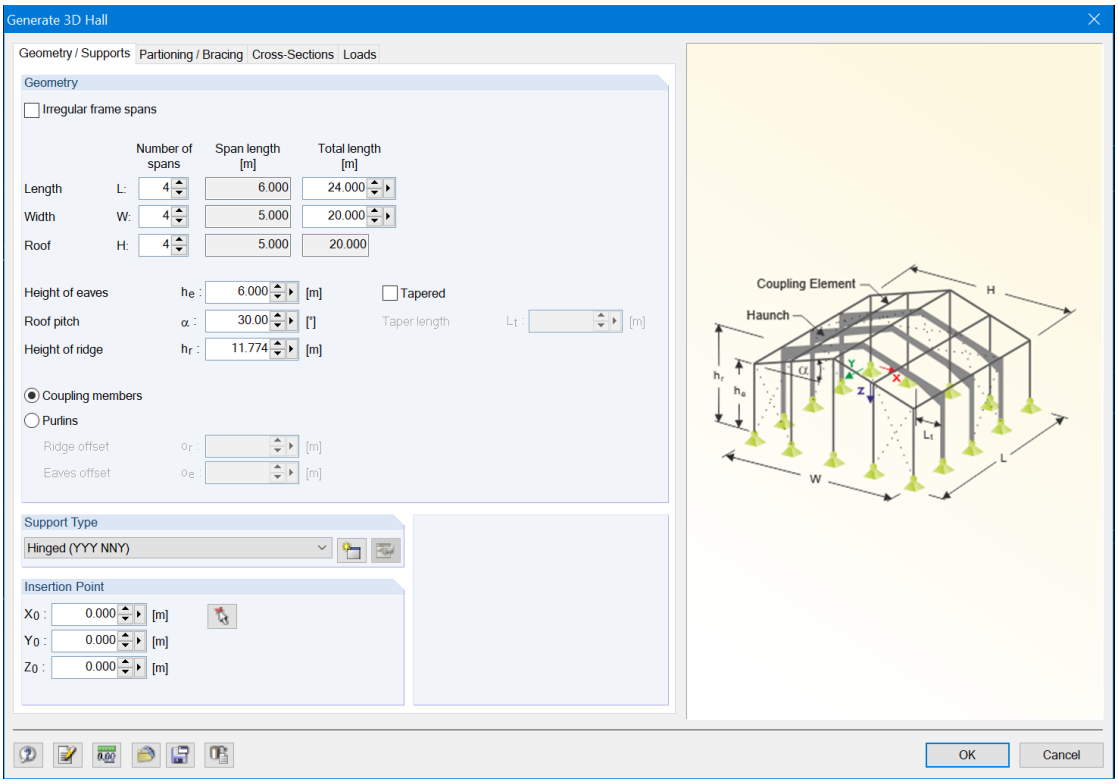
2.3 Generování halových konstrukcí

2.3.1 Dlubal 3D hall generator

Společnost Dlubal je na trhu již od druhé poloviny osmdesátých let. Za tu dobu si vydobila velmi silnou pozici a její hlavní produkty, aplikace RFEM a RSTAB, patří mezi nejpoužívanější software pro analýzu stavebních konstrukcí nejen u nás, ale i v Evropě. Aplikace RFEM funguje na takzvaných modulech, které uživateli poskytují vždy konkrétní danou funkcionalitu. Jedním z takových modulů je generátor konstrukcí. Ten mimo jiné umožňuje i generování 3D modelů ocelových hal. [10]

Ve své podstatě se jedná o vestavěnou aplikaci, ve které uživatel vyplní základní parametry typické pro halové konstrukce (například rozpětí haly, vzdálenost mezi jednotlivými rámy, výška sloupů, průřezy jednotlivých prvků, atd.) a na jejich základě dojde k vygenerování halové konstrukce přímo do 3D scény. Navíc je tu přidána i podpora pro generování zatížení, což je záležitost specifická pouze pro analytické výpočetní modely. Ukázka prostředí je přiložena na obrázku 2.3. [11]

Vzhledově by se aplikace opět dala přirovnat ke klasickému formulářovému WinForms stylu. Parametry jsou logicky rozděleny do několika záložek, kdy na záložce první se nachází údaje definující většinu geometrie, druhá záložka je věnována nastavení průřezů a materiálu, a v poslední části pak uživatel definuje hodnoty týkající se již zmiňovaného zatížení.



Obrázek 2.3 Generátor 3D analytického modelu halových konstrukcí v aplikaci Dlubal RFEM [12]

Silnou stránkou tohoto řešení je bez pochyby přehlednost a uživatelská přívětivost. Jako stavební inženýr používající tuto aplikaci chcete mít na očích pouze relevantní vstupní parametry, mít možnost je rychle zadat a vygenerovat si chtěný model. A to přesně generátor umožňuje.

Nejedná se o žádné komplexní řešení, které by dávalo možnost přizpůsobit generovaný mo-

del nějakým atypickým konstrukcím. Použít lze aplikaci pro haly obdélníkového půdorysu, se sedlovým typem střešních vazníků. Jednotlivé rámy mohou být propojeny vaznicemi, případně pouze rozpěrami. Relativně úzké cílové zaměření lze považovat v dané situaci spíše za výhodu, kdy uživatel sice nemá volnost se odchylovat od typického tvaru, ale díky tomu je pro něj použití této aplikace velmi jednoduché, rychlé a nabízí mu značnou přidanou hodnotu, především vygenerováním zatížení. [11]

Pro upřesnění je nutné zmínit, že tento generátor je dostupný pouze ve starších verzích RFEM 5.x. Nejnovější verze RFEM 6.x tuto funkcionalitu neobsahuje. Co je důvodem a zda dojde ke konverzi modulu i pro novou verzi v budoucích aktualizacích či nikoli, nebylo zatím oficiálně ohlášeno.

2.3.2 HiStruct Steel Building Generator

Česká firma HiStruct se zabývá parametrickým generováním 3D modelů nejrůznějších stavebních konstrukcí. Jedním z jejich produktů je Steel Building Configurator, který umožňuje modelování ocelových halových staveb. Aplikace je dostupná prostřednictvím webového prohlížeče. Nabízí uživatelsky přívětivé UI a velmi responzivní 3D scénu. Ukázku prostředí aplikace lze vidět na obrázku 2.4. [13]

Obrázek 2.4 Konfigurátor ocelových halových konstrukcí od společnosti HiStruct [14]

Uživatel má k dispozici několik dlaždic, které logicky rozčleňují nastavitelné parametry. Na rozdíl od předchozích řešení se zde jedná o komplexní stavební model. Konfigurovat lze jak samotnou nosnou konstrukci, tak i opláštění, celkový vzhled, okenní a dveřní otvory a spoustu dalšího. Další odlišností je také aplikování změn a regenerování modelu okamžitě při úpravě každého parametru. Uživatel má tak aktuální podobu stavby vždy doslova před očima. To je pro tento produkt velmi důležité, jelikož jejím cílovým uživatelem nemusí být, a pravděpodobně ve většině případů ani není, stavební inženýr. V mnoha případech to bude skutečně koncový zákazník, který si chce nechat vystavět novou halu. Případně projektový manažer nebo obchodní

zástupce dodavatelské společnosti, který může tuto službu využít pro velmi snadnou a rychlou vizualizaci výsledného řešení při diskusích se zákazníkem.

Aplikace nabízí množství výstupů, které tak umožňují její využití pro mnoho různých případů. Jedním z možných exportů je i ten do formátu IFC. Jak již bylo zmíněno, ten je v současné době zavedeným standardem v AEC odvětví a lze ho tak (v určité míře) použít téměř v jakémkoli stavebním softwaru. Další variantou je výstup do 2D DXF, případně rovnou do PDF ve formě zjednodušené výkresové dokumentace. V neposlední řadě pak nástroj nabízí i výkazy materiálu a seznamy dílů. Tyto údaje lze použít například pro rychlé sestavení odhadu potřebného rozpočtu. I přes výše uvedené možnosti je nutné zmínit, že v porovnání s Teklou Structures se jedná spíše o byznysově zaměřenou aplikaci, která neumožňuje žádné pokročilé úpravy, detailní modelování ani analýzu konstrukce na inženýrské úrovni.

Na oficiálních webových stránkách společnosti není k dispozici žádný veřejný ceník. Jak společnost zmiňuje, její řešení je velmi často dodáváno spolu s integrací do nějakého rozsáhlejšího systému, případně upravováno zákazníkům na míru. Konečnou cenu lze tedy jen těžko předem určit. [15]

2.4 Dodatek k rešerši podobných řešení

V předcházejících částech bylo uvedeno několik veřejně dostupných aplikací zabývajících se problematikou parametrického modelování v oboru stavebnictví. Dá se však předpokládat, že v praxi se vyskytuje daleko více nástrojů a aplikací, které slouží k podobnému účelu, avšak jsou vyvíjeny interně v rámci jednotlivých firem, a tudíž nejsou dostupné veřejnosti. Dnešní softwary používané ve stavební praxi mají často nějaké veřejné rozhraní (API, WebServices, apod.), které umožňuje ovládat aplikaci bez klasického použití UI, integraci aplikace do jiného systému, či import a export příslušných dat. Pracovní procesy jednotlivých společností jsou velmi rozdílné a takové nástroje jsou proto většinou tvořeny přímo na míru.

Kapitola 3
Analýza a návrh

První část této kapitoly se zabývá definováním vstupů, které jsou potřebné k vygenerování halových konstrukcí, a které tak musí být součástí aplikace. Kapitola se také věnuje analýze funkčních a nefunkčních požadavků. Dále popisuje případy užití, pro které bude tvořená aplikace uživatelům sloužit. V poslední části se pak zabývá prvotním návrhem uživatelského prostředí a architekturou celého řešení.

3.1 Vstupní parametry popisující halové konstrukce

3.1.1 Vymezení typu generovaných konstrukcí

Halové konstrukce mohou být obecně popsány velkým množstvím proměnných. Pro tvořenou aplikaci bylo proto zvoleno několik omezení, která mají za účel jednoznačně vytyčit typ generovaných konstrukcí. Důsledkem je také snížení počtu potřebných parametrů, což vede k přehlednějšímu prostředí pro uživatele.

Obdélníkový půdorysný tvar
Prvním omezením je obdélníkový půdorysný tvar konstrukce. Jedná se o v praxi nejčastější variantu halových staveb. Při manuálním modelování tohoto tvaru také dochází k mnoha opakujícím se úkonům, které je vhodné nahradit právě parametrickým modelováním.

Symetričnost konstrukce podle středové osy
Dalším omezením je podmínka symetričnosti konstrukce podle její podélné středové osy. Ve stavební praxi je toto opět velmi častý případ. Pokud by byla umožněna stranová asymetrie, například pro průřezy či umístění ztužidel, vedlo by to k nutnosti zadávat mnohem větší počet parametrů a uživatelský dialog by tak opět nabyl na komplexnosti.

Typy střešní geometrie
Jako podporované typy střešní geometrie byly zvoleny střecha sedlová (oboustranný sklon), pultová (jednostranný sklon) a střešní nosník v zakřiveném tvaru (část kružnice). Z důvodu zachování jednoduchosti zadávání vstupů a přehlednosti uživatelského rozhraní byla vynechána možnost volby příhradového nosníku (jedná se nosník, který je složen z několika dalších nosníků).

3.1.2 Vybrané vstupní parametry

Na základě rešerše provedené v předcházející kapitole a zkušeností ze stavební praxe byly vybrány parametry, které musí uživatel vyplnit, aby bylo možné halovou konstrukci vygenerovat. Zde je uveden jejich výčet se stručným popisem.

Type of roof Typ střešní konstrukce, která může být sedlová (gable), pultová (pitch) nebo zakřivená (curved).

Span [mm] Hodnota definující vzdálenost mezi osami sloupů v hlavní vazbě konstrukce (šířka konstrukce).

Spacing between frames [mm] Osová vzdálenost mezi jednotlivými vazbami. Tato hodnota je aplikována na všechny pole konstrukce.

Count of bays Počet polí halové konstrukce tvořených hlavními vazbami.

Column height [mm] Výška sloupů (měřena mezi hlavou a patou sloupů) v hlavních vazbách. Hodnota je pro obě strany stejná. Výjimkou je, pokud uživatel zvolí střešní konstrukci s jednostranným sklonem. Pro tuto variantu je mu umožněno zadat rozdílnou výšku levých a pravých sloupů. Zadaná hodnota vždy platí pro všechny hlavní vazby.

Roof height [mm] Vzdálenost mezi hlavou sloupů a vrcholem střešní konstrukce. V případě, že uživatel zvolí střechu s jednostranným sklonem, je tato volba neaktivní.

Spacing of purlins [mm] Osová vzdálenost mezi jednotlivými vaznicemi. Hodnota je měřena ve střešní rovině, v případě oblouku se jedná o délku měřenou na zakřiveném nosníku.

Spacing of side rails [mm] Osová vzdálenost mezi jednotlivými paždíky na bočních stěnách. První paždík je umístěn ve zvolené vzdálenosti od úrovně paty sloupů. Poslední pak maximálně zvolenou vzdálenost od osy rohového nosníku spojujícího jednotlivé vazby.

Count of gable mullions Počet doplňkových štítových sloupků. Do počtu se nazapočítávají hlavní sloupy rámu.

Spacing of gable mullions [mm] Osová vzdálenost jednotlivých štítových sloupků. Stejná vzdálenost je aplikována pro všechna pole.

Side bracing Volba, zda chce uživatel v bočních stěnách daného pole použít ztužující prvky.

Side bracing type Typ ztužení bočních stěn. Na výběr je ztužení křížem, levostranné nebo pravostranné.

Side bracing height [mm] Výška, do které zasahují boční ztužidla. Tato hodnota nesmí přesahovat výšku sloupů.

Roof bracing Volba, zda chce uživatel pro dané pole aplikovat ztužení ve střešní rovině.

Roof bracing type Typ ztužení ve střešní rovině. Na výběr je ztužení křížem, levostranné nebo pravostranné.

Count of purlins crossed Počet vaznic, která ztužidla kříží. Krajní vaznice, v jejichž počátečních a koncových bodech jsou ztužidla ukotvena, se do této hodnoty nezapočítávají.

Gable wall field bracing Volba, zda chce uživatel pro dané pole ve štítové stěně aplikovat ztužující prvky.

Gable wall bracing height [mm] Výška, do které zasahují ztužující prvky v daném poli štítové stěny. Maximální výška je dána výškou sloupů.

Material Uživatel má možnost zvolit konkrétní materiál pro následující skupiny prvků: sloupy, střešní nosník(y), spojující prvky v rozích rámů, vaznice, paždíky, sloupky ve štítových stěnách, ztužidla ve střešní rovině, ztužidla v bočních stěnách, ztužidla ve štítových stěnách.

Cross-section type and profile Volba tvaru průřezu a konkrétního profilu. Opět je možné zvolit zvlášť pro všechny výše uvedené skupiny prvků.

3.2 Funkční požadavky

Cílem práce je vytvořit aplikaci pro vygenerování modelu halové konstrukce na základě zadaných parametrů. Mezi funkční požadavky lze zařadit umožnění zadat potřebné parametry a vygenerovat konstrukční model. Dále pak také nakonfigurovat si aplikaci pomocí importu uživatelského seznamu materiálů a průřezů.

3.2.1 Zadání vstupních parametrů konstrukce

Jedním ze základních funkčních požadavků je možnost zadání vstupních geometrických parametrů nezbytných pro vygenerování konstrukce. Seznam parametrů vyplývá z analýzy v kapitole 3.1.2. Způsob zadávání parametrů musí být pro uživatele dostatečně přehledný a jasný tak, aby tento proces uživatele zbytečně nezdržoval. Pro jednoznačnost a srozumitelnost parametrů bude mít uživatel k dispozici i obecné schéma, ve kterém budou ty nejdůležitější parametry vyznačeny. Aplikace musí být schopná ošetřit nevalidní vstupy a poskytnout uživateli příslušnou odezvu.

3.2.2 Použití vlastních seznamů materiálů a průřezů

Ve stavební praxi je velmi typické, že v rámci firmy se používají vybrané materiály a průřezy, které jsou pro danou oblast typické a snadno dostupné. Aplikace musí umožnit uživateli změnit předdefinovaný seznam materiálů i průřezů, například pomocí načtení ze souboru. Formát souboru by měl být volen s ohledem na to, aby ho bylo možné otevřít, editovat a vytvářet pomocí volně dostupných editorů. Cílovými uživateli aplikace jsou stavební inženýři, u kterých se nepředpokládá žádná znalost programovacích jazyků. Je tedy důležité, aby i pro takové uživatele byl formát dostatečně srozumitelný a čitelný. Při importu také musí dojít k validaci dat. Uživatel bude zřetelně obeznámen s výsledkem importu (úspěch/neúspěch) a v případě neúspěšného importu nedojde k žádným změnám v aktuálním nastavení aplikace.

3.2.3 Vygenerování 3D modelu stavební konstrukce

Pokud uživatel vyplní všechny parametry, může si nechat vygenerovat 3D model stavební konstrukce. Pro úspěšné vygenerování modelu je nutné, aby konfigurace zadaných parametrů byla validní. Validace vstupů musí proběhnout automaticky na začátku procesu generování. Pokud dojde k porušení nějakých omezení (například výška ztužidel v boční stěně je vyšší než výška sloupů), nebo je zadaný parametr mimo své meze rozsahu, uživatel je o této skutečnosti obeznámen informační hláškou a export dat je přerušen. Pro vygenerování modelu do 3D scény softwaru Tekla Structures je také nutné, aby aplikace Tekla Structures byla spuštěna a byl v ní otevřen nějaký projekt, jinak se nepodaří navázat spojení. V takovém případě musí být uživatel opět informován. Pokud se konstrukci podaří úspěšně vygenerovat, uživatel obdrží informační hláškou potvrzení, že export je úspěšně ukončen.

3.3 Nefunkční požadavky

Aplikace je navržena pro interní použití v rámci jedné firmy, pro několik vybraných zaměstnanců. Její využití se očekává především jako jednorázové, vždy na začátku projektu pro vytvoření stavebního modelu. Z tohoto důvodu není vyžadována žádná integrace do ostatních interních systémů. Nutné je pouze zajištění bezproblémové komunikace a použití se softwarem Tekla Structures. Z výše uvedeného vyplývají níže uvedené požadavky jiného než funkčního typu.

3.3.1 Požadavky na vzhled UI

Aplikace bude používána především na pracovních stanicích. Rozvržení musí být uzpůsobeno na použití na desktopových systémech s běžnými monitory o velikosti 15–32". Vzhled UI by měl být dostatečně čitelný a přehledný pro standardní rozlišení 1920*1080, 2560*1440 a 3840*2160.

3.3.2 Systémové požadavky

Vzhledem k závislosti na softwaru Tekla Structures budou systémové požadavky do velké míry shodné i pro vyvíjenou aplikaci. Jediným podporovaným operačním systémem, na kterém bude aplikace používána je Windows 10/11 v 64-bitové verzi. Samotná Tekla Structures vyžaduje pro svůj optimální provoz 16 GB RAM paměti a alespoň čtyřjádrový procesor. Tyto hodnoty jsou dostatečné i pro vyvíjenou aplikaci.

3.3.3 Možnost budoucí výměny UI

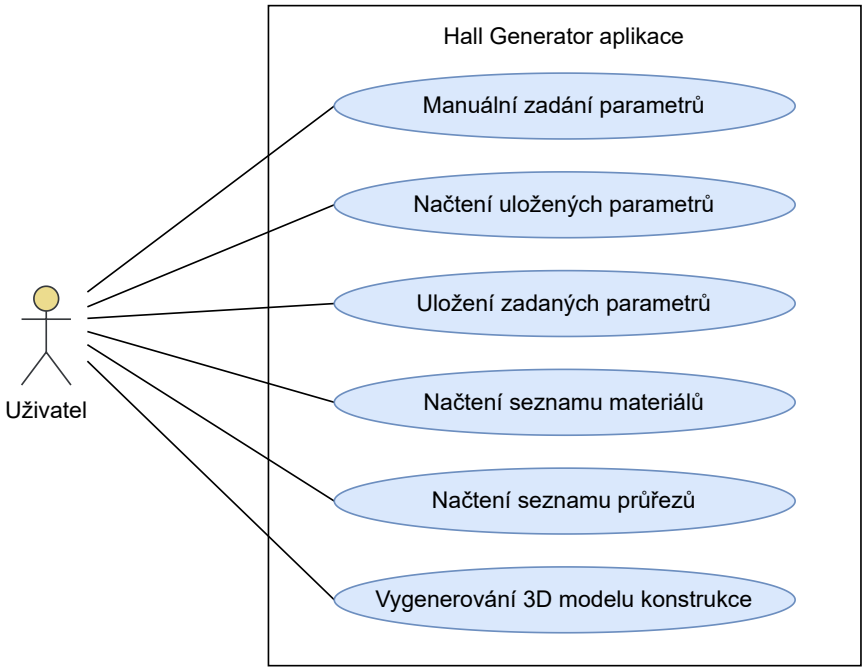
Jedním z požadavků je také co nejsnadnější budoucí výměna UI za jiné. Trendy ve vzhledu aplikací všeho druhu se v dnešní době velmi rychle mění a lze očekávat, že v následujících letech tomu nebude jinak. Aplikace by měla mít co nejvíce oddělenou část uživatelského prostředí od samotné logiky pracující s daty tak, aby případný budoucí požadavek na výměnu UI bylo možné realizovat bez zásahů do ostatních částí kódu.

3.4 Případy užití

Tato kapitola popisuje jednotlivé případy užití tak, jak vyplývají z analýzy a výše uvedených funkčních požadavků. Jako aktér vystupuje ve všech případech běžný uživatel, kterým je stavební inženýr pracující s navrhovanou aplikací a softwarem Tekla Structures.

3.4.1 Manuální zadání parametrů

Aplikace musí obsahovat příslušnou obrazovku, kde může uživatel manuálně zadávat jednotlivé parametry konstrukce, kterou chce vygenerovat. Ze seznamu parametrů definovaného v kapitole 3.1.2 vyplývá, že se jedná se o zadání několika číselných hodnot, výběr možností z předdefinovaných seznamů, případně volbu ano/ne (např. použitím checkboxů). Tyto vyžádané vstupy musí být logicky uspořádané. Jejich pořadí musí odpovídat důležitosti stavebních prvků, kterých se daný parametr týká. Za nejdůležitější části konstrukce jsou považovány nosné prvky, dále pak podružné konstrukce pro opláštění a na závěr konstrukční prvky zajišťující ztužení. Při zadání dat dojde také k jejich základní validaci. Pokud se uživatel pokusí zadat data, která nejsou validní (ať už svým typem či hodnotou), aplikace o této skutečnosti uživatele informuje, případně mu taková data vůbec nedovolí zadat. Na obrazovce se zadávanými parametry bude uživateli zobrazeno i obecné schéma konstrukce, ve kterém budou vyznačeny jednotlivé parametry. Jedná



Obrázek 3.1 Diagram případů užití

se o statický obrázek sloužící jako vizuální pomůcka k rychlejšímu identifikování významu jednotlivých parametrů.

3.4.2 Načtení uložených parametrů

Aby uživatel nemusel pokaždé zadávat všechny údaje manuálně, musí aplikace podporovat i načtení parametrů z uloženého souboru. Protože uživatelem bude ve většině případů stavební inženýr, je nutné aby formát souboru byl snadno čitelný a modifikovatelný v nějakém běžném textovém editoru. Kliknutím na tlačítko importu se uživateli zobrazí dialog pro výběr souboru. Po potvrzení souboru se parametry načtou a automaticky vloží do příslušných polí formuláře. Pokud je soubor ve špatném formátu, některé parametry chybí, nebo dojde k obecné chybě čtení, uživatel je informován, že data se nepodařilo načíst a aktuálně zadané hodnoty parametrů zůstanou nezměněny.

3.4.3 Uložení aktuálně zadaných parametrů

Pro zjednodušení tvorby souboru určeného k importu parametrů (případ užití 3.4.2) umožní aplikace také export aktuálně zadaných parametrů do souboru. Pro ten platí již výše zmíněné požadavky na čitelnost a možnost úpravy v textovém editoru. Při vyvolání funkce exportu parametrů se uživateli objeví dialog pro výběr cesty k souboru a po potvrzení se provede jeho export. V případě chyby dojde k jejímu oznámení uživateli a aktuálně zadané parametry v aplikaci zůstanou beze změny.

3.4.4 Načtení uživatelského seznamu materiálů

Uživatel má v aplikaci možnost přiřadit jednotlivým skupinám prvků konkrétní stavební materiál. Ten může vybrat z předdefinovaného seznamu podporovaných materiálů. Jelikož je běžné, že

uživatelé používají velké množství rozdílných materiálů, musí aplikace umožnit nahradit výchozí seznam materiálů seznamem uživatelským. Po kliknutí na tlačítko importu materiálů se uživateli zobrazí okno s dialogem pro výběr souboru. Po výběru souboru a jeho potvrzení dojde k nahrazení současného seznamu materiálů v aplikaci materiály z načteného souboru. Pokud je importovaný soubor v nesprávném formátu nebo dojde k chybě čtení, import se přeruší a aktuální seznam materiálů v aplikaci zůstane beze změny. Pokud k této události dojde, uživatel je informován.

3.4.5 Načtení uživatelského seznamu průřezů

Aplikace nabízí uživateli na výběr základní předdefinovaný seznam průřezů rozdělený do skupin podle jejich typu. Protože je ve stavební praxi typické, že uživatel používá svoje vybrané či jakkoli atypické průřezy, aplikace musí uživateli umožnit naimportovat svůj seznam průřezů. Tímto bude zajištěna dostatečná přizpůsobitelnost programu na míru uživateli. Stejně jako v předchozím případě (3.4.4), po kliknutí na tlačítko se uživateli zobrazí dialog pro výběr souboru. Po výběru a potvrzení dojde k nahrazení současné databáze průřezů v aplikaci. Ta je rozdělena do dvou částí. První z nich je seznam typů. Typem průřezu jsou myšleny například válcované průřezy ve tvaru „I“, jako jsou IPE, HEA nebo HEB. Ke konkrétnímu typu pak náleží seznam konkrétních provedení daných průřezů dle velikosti. Pro výše zmíněný příklad nosníků IPE jsou to profily IPE100, IPE120, IPE160, atd. Pro zachování konzistence umožňuje aplikace import pouze celého seznamu složeného ze seznamů tvarů i konkrétních průřezů. Při načítání dochází také ke kontrole formátu dat. Pokud je nalezena chyba ve formátu či dojde k chybě při čtení souboru, import se přeruší a v aplikaci zůstanou původně používané seznamy. Uživatel je o této události náležitě informován.

3.4.6 Vygenerování 3D modelu stavební konstrukce

Po vyplnění všech vstupních parametrů může uživatel využít funkce vygenerování stavebního modelu přímo do 3D scény aplikace Tekla Structures. Na začátku procesu generování modelu provede aplikace ověření vstupů. Pokud nastane při pokusu o vygenerování modelu nějaký problém, uživatel je řádně informován, že k exportu nedošlo. Zadané parametry zůstanou beze změn. Stejně tak je uživatel informován, pokud je export úspěšně dokončen.

3.5 Použité technologie a návrhové vzory

V následující kapitole jsou popsány technologie, nástroje a přístupy, které byly pro svoji vhodnost vybrány a následně použity v implementační části práce.

3.5.1 .NET Framework

Microsoft .NET Framework lze označit za ucelenou platformu, sloužící pro vývoj, běh a správu aplikací či knihoven na operačním systému Windows. Jedná se o soubor knihoven, služeb a nástrojů, které mohou vývojáři použít při tvorbě svých programů. Aktivní vývoj této platformy byl již ukončen, ale její nejnovější verze jsou stále podporovány včetně pravidelných bezpečnostních aktualizací a oprav kritických chyb. [16]

Velkou výhodou platformy je její nezávislost na programovacím jazyku. Podporováno jich je několik, mezi nejpoužívanější lze zařadit C#, Visual Basic a F#. Kód je nejdříve compilerem přeložen do tzv. mezikódu, který v tomto případě Microsoft označuje jako Common Intermediate Language (CIL). Dalším krokem je pak zpracování běhovým prostředím Common Language Runtime (CLR), které slouží jako interpret. Výsledkem je strojový kód, který je zpracován procesorem. [17]

Hlavním účelem této platformy je usnadnit a urychlit vývoj aplikací na operačním systému Windows. Platforma umožňuje vytvářet mnoho typů aplikací, od obyčejných konzolových, přes formulářové typu WinForms, až po ty s pokročilým grafickým uživatelským rozhraním, jako jsou aplikace typu WPF. [16, 18]

3.5.2 Windows Presentation Foundation

Windows Presentation Foundation (dále WPF) je vývojářské rozhraní pro návrh a zobrazení uživatelského prostředí. Základem WPF je vektorové vykreslovací jádro, které umožňuje využívat moderní grafický hardware. WPF jako takové zastřešuje mnoho dalších funkcí pro vývoj aplikací, které obsahují rozšiřitelný značkovací jazyk aplikací (XAML), ovládací prvky, rozvržení, 2D a 3D grafiku, datové vazby, styly, atd. Velkou výhodou WPF je skutečnost, že je součástí rozhraní .NET a lze ho tak využít pro tvorbu aplikací na této platformě. [19]

Historicky se jedná o nástupce WinForms technologie, oproti které přináší spoustu vylepšení a výhod. Mezi ty nejvýraznější patří především daleko větší volnost při tvorbě vzhledu jednotlivých prvků, stylování a využívání šablon. Dále přináší nové možnosti v oblasti bindingu dat, využívání triggerů, nebo synchronizaci pomocí tzv. dispatchers. I přes to, že Microsoft již nabízí další nové technologie, které jsou označovány za následníky WPF (např. UWP nebo .NET MAUI), tato technologie stále zůstává v praxi hojně využívána a především pro vývojáře čistě desktopových aplikací zatím často neexistuje důvod na tyto novější technologie přecházet. [20]

3.5.3 NUnit

NUnit je testovací framework určený pro unit testování kódu na platformě .NET. Jedná se o open-source projekt vydávaný pod MIT licencí (od verze 3.x). Vývojářům nabízí možnost snadné integrace do jejich projektu pomocí volně dostupných Nuget packages. Velkou výhodou NUnit je schopnost pouštět testy paralelně, což vede k značné úspoře času. Framework podporuje konfiguraci a definování testů pomocí atributů. Stejně tak lze použít data-driven testování, kdy díky použití atributů lze předem nadefinovat sadu vstupů a očekávaných výstupů. Test je pak spuštěn pro všechny předepsané případy. Pro zvýšení přehlednosti poskytuje NUnit možnost seskupení jednotlivých testů do tzv. test fixtures. [21, 22, 23]

3.5.4 NSubstitute

NSubstitute je framework pro mockování a substituci objektů pro platformu .NET a jazyk C#. Mockování je technika používaná při tvorbě testů během softwarového vývoje. Jedná se v podstatě o vytváření náhradních objektů (tzv. mocků), které pouze simulují chování skutečných objektů. NSubstitute umožňuje nejen vytvářet mocky, ale i definovat detailně jejich chování včetně návratových hodnot. Použití tohoto přístupu velmi usnadňuje testování jednotlivých tříd a pomáhá izolovat testovanou část kódu. [24]

Od ostatních podobných řešení na trhu se tato knihovna liší především jednoduchou a srozumitelnou syntaxí. Ta je snadno čitelná i pro začínající vývojáře. Velkou výhodou je také flexibilita při definování chování mock objektů a celkově podpora mnoha různých typů objektů. [25]

3.5.5 FluentAssertions

Jedná se o open-source knihovnu pro platformu .NET, která nabízí sadu metod pro psaní přehlednějších a srozumitelnějších unit testů. Jedním z problémů při použití klasických assertů v unit testech je, že uživatel obdrží pouze strohou informaci o tom, zda test byl úspěšně dokončen, či nikoli. FluentAssertions přichází s řešením, které umožňuje daleko přesněji identifikovat, proč

daný test selhal. Definice, zápis, ale i výsledek testu je navíc psaný syntaxí, která je velmi podobná přirozenému jazyku. Knihovna podporuje integraci s nejčastějšími testovacími frameworky používanými na platformě .NET jako jsou NUnit, MSUnit, xUnit, a další. [26]

3.5.6 Serilog

Serilog patří mezi nejpoužívanější logovací nástroje na platformě .NET. Umožňuje monitorování běhu aplikace pomocí zápisu do souboru, konzole, databáze a mnoha dalších umístění. Nechybí ani podpora strukturovaného logování, díky kterému lze zaznamenat o dané události mnohem více informací. Snadná integrace a použití ve vlastním projektu je zajištěna pomocí volně dostupných Nuget packages. Velmi užitečnou funkcionalitou je také možnost si téměř libovolně naformátovat podobu záznamu. Jedná se o open-source projekt, který je distribuován pod Apache 2.0 licencí. [27, 28]

3.5.7 SonarLint

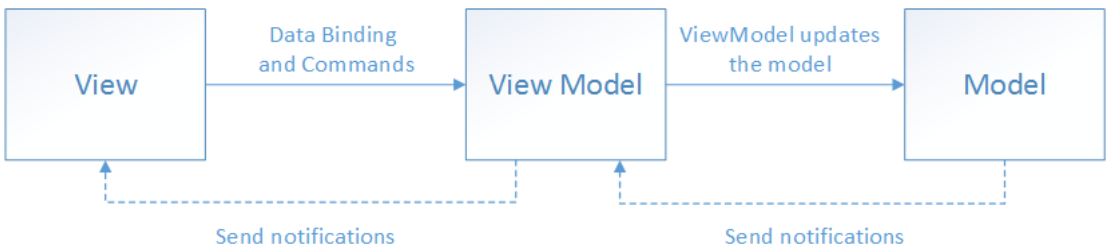
Doplněk do Visual Studia s názvem SonarLint lze zařadit do skupiny nástrojů označovaných anglickým výrazem *linter*. Tyto nástroje slouží ke statické analýze kódu, jejíž cílem je upozornit vývojáře na chyby, potenciální bezpečnostní rizika či nevhodné konstrukty v kódu. SonarLint podporuje analýzu kódu psaného v jazycích C#, C/C++, VB.NET, ale i JavaScript a TypeScript. Doplněk je vyvíjen jako bezplatný a open-source, tudíž jeho zdrojové kódy jsou veřejně přístupné na portále Github. [29]

3.5.8 CodeMaid

Dalším z doplňků do Visual Studia je knihovna s názvem CodeMaid. Jedná se o nástroj, který lze použít k čištění kódu po formální stránce. Sjednocení bílých znaků, odstranění nepotřebných *usings* nebo jejich následné seřazení, toho všeho lze dosáhnout automaticky s použitím CodeMaid. Jeho užití celkově přispívá k čitelnějšímu a přehlednějšímu kódu. Doplněk lze využít pro zdrojové kódy v mnoha jazycích, například v C#, C/C++, JSON, XAML, PHP a dalších. Jeho použití je bezplatné a vývoj je open-source. [30]

3.5.9 Model-View-ViewModel návrhový vzor

Model-View-ViewModel (zkráceně MVVM) je architektonický návrhový vzor pro vývoj aplikací. Je velmi často spojován s aplikacemi typu WPF od společnosti Microsoft, lze ho však použít i při vývoji software na jiných platformách. Jeho hlavním přínosem je oddělení logiky aplikace od uživatelského rozhraní. Použití tohoto vzoru vede na přehlednější, čistší kód, který se snadněji udržuje, modifikuje i testuje.



Obrázek 3.2 Princip fungování MVVM návrhového vzoru [31]

MVVM lze rozdělit na tři hlavní vrstvy: *Model*, *View* a *ViewModel* (viz obrázek 3.2). *Model* obsahuje a popisuje data, se kterými aplikace pracuje. Zároveň však nemá žádné informace o tom, v jakém stavu se aplikace nebo prvky uživatelského rozhraní nachází. *View* reprezentuje uživatelské rozhraní, v případě WPF aplikací například v podobě XAML souboru. Můžeme ho označit za prezentační vrstvu. Obsahuje veškeré ovládácí prvky pro uživatele. Data pro zobrazení čerpá pomocí bindingu z *ViewModelu*. Ten je středobodem celé myšlenky tohoto návrhového vzoru. Tato třída drží stav aplikace a na jeho základě filtruje data. Aby docházelo ke správné synchronizaci s *View*, musí třída implementovat pro své vlastnosti (properties) *INotifyPropertyChanged*, případně pak *INotifyCollectionChanged* pro kolekce. *ViewModel* komunikuje i s *Modelem*, ve kterém mění data na základě uživatelských příkazů a akcí. [31, 32]

3.6 Návrh architektury aplikace

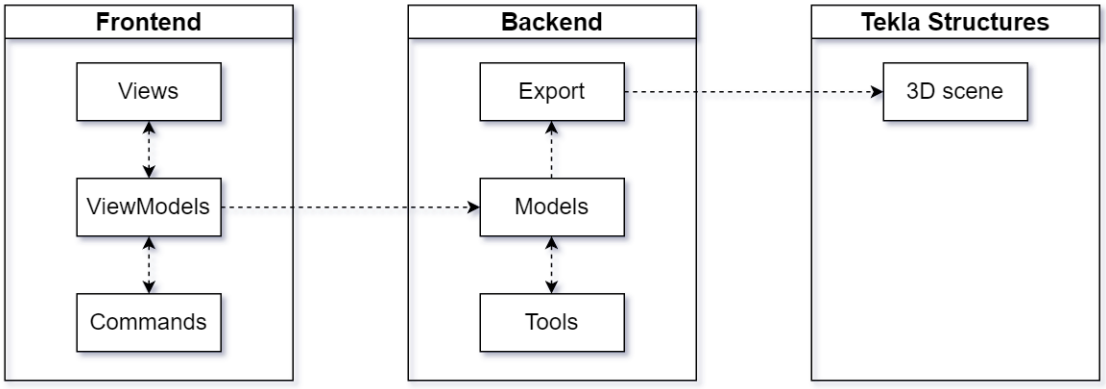
Tato podkapitola se věnuje návrhu základní architektury aplikace. Zahrnuje identifikaci hlavních částí, jejich propojení a definování doménového modelu pro část zabývající se byznys logikou související s halovou konstrukcí.

3.6.1 SW architektura

Základní návrh struktury aplikace byl uzpůsoben všem kladeným požadavkům, především pak podmínce na možnost v budoucnu vyměnit uživatelské prostředí za jiné. Z toho důvodu bylo nutné striktně oddělit část uživatelského prostředí (frontend) a část pracující s daty a byznys logikou (backend). Pro tento účel bylo zvoleno využití MVVM návrhového vzoru.

Do frontendové části lze zařadit *Views*, která se starají o uživatelské rozhraní a vstupy. Tyto vstupy jsou následně předávány do *ViewModels*, kde dochází k jejich zpracování. Balíček *Commands* se stará o vykonání konkrétních akcí na základě příkazů od uživatele. *ViewModels* slouží jako spojující část mezi frontendem a backendem. Přebírá vstupy z *Views* a následně je komunikuje (s využitím *Commands*) do příslušného balíčku v backednové části.

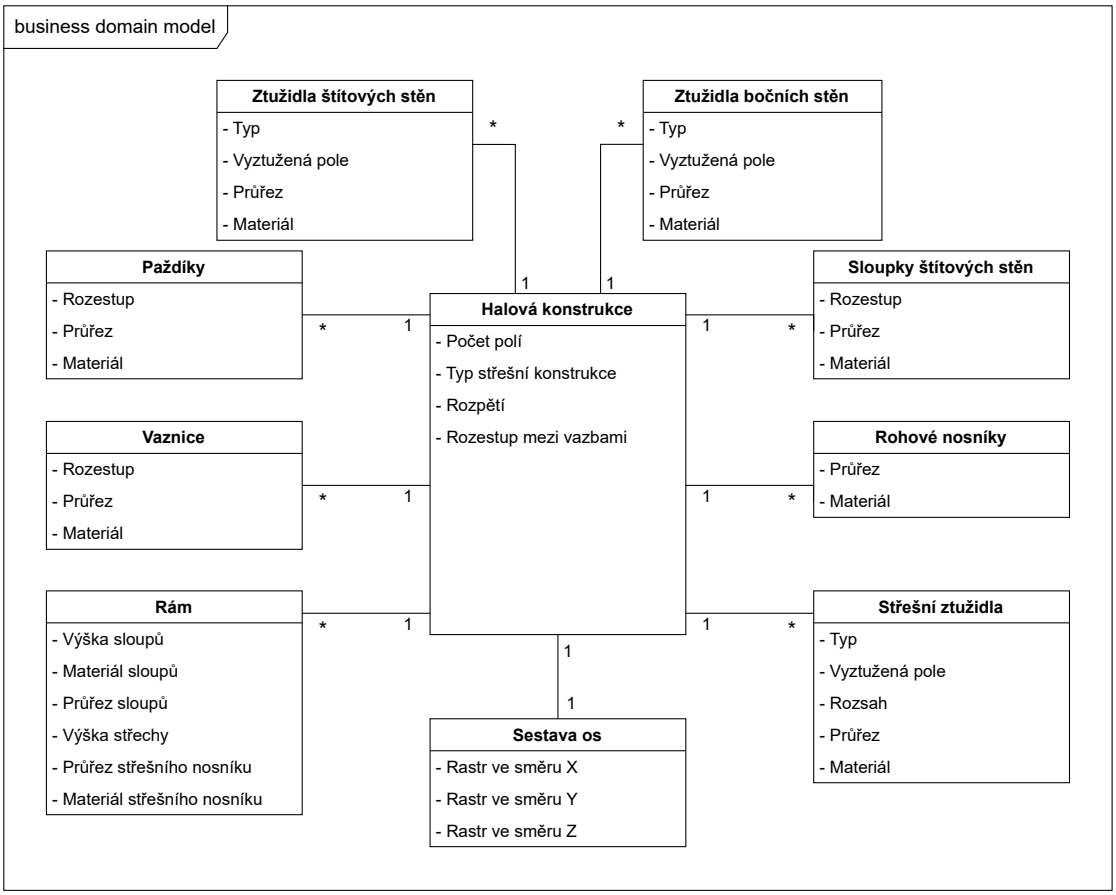
Celá část zabývající se byznys logikou a daty vůbec o frontendové části aplikace neví. *ViewModels* pouze naplní potřebné vstupy a zavolá příslušné metody z modelů přes definované rozhraní. Zde již dochází ke zpracování zadaných parametrů a následnému vygenerování modelu halové konstrukce. Samostatná část se pak stará o finální komunikaci s aplikací Tekla Structures a export modelu do 3D scény této aplikace.



Obrázek 3.3 Základní návrh architektury

3.6.2 Doménový byznys model

Na základě analýzy parametrů nutných pro vytvoření typické halové konstrukce byl vytvořen doménový byznys model (obrázek 3.4). Základní skelet konstrukce je tvořen hlavními nosnými rámy a rohovými nosníky, které propojují jednotlivé rámy. Ve střešní rovině jsou rámy spojeny vaznicemi, v rovině sloupů pak paždíky. Ve štítových stěnách konstrukce se nachází doplňkové štítové sloupky. Model obsahuje také prvky určené ke ztužení konstrukce. Mezi ty patří ztužidla ve střešní rovině, ztuždila v bočních stěnách a případně i ztužení mezi jednotlivými sloupky štítových stěn. Jednotlivé submodely většinou označují dané prvky v jednom poli konstrukce. Proto je zde aplikována vazba 1-N i přes to, že název je v množném čísle.



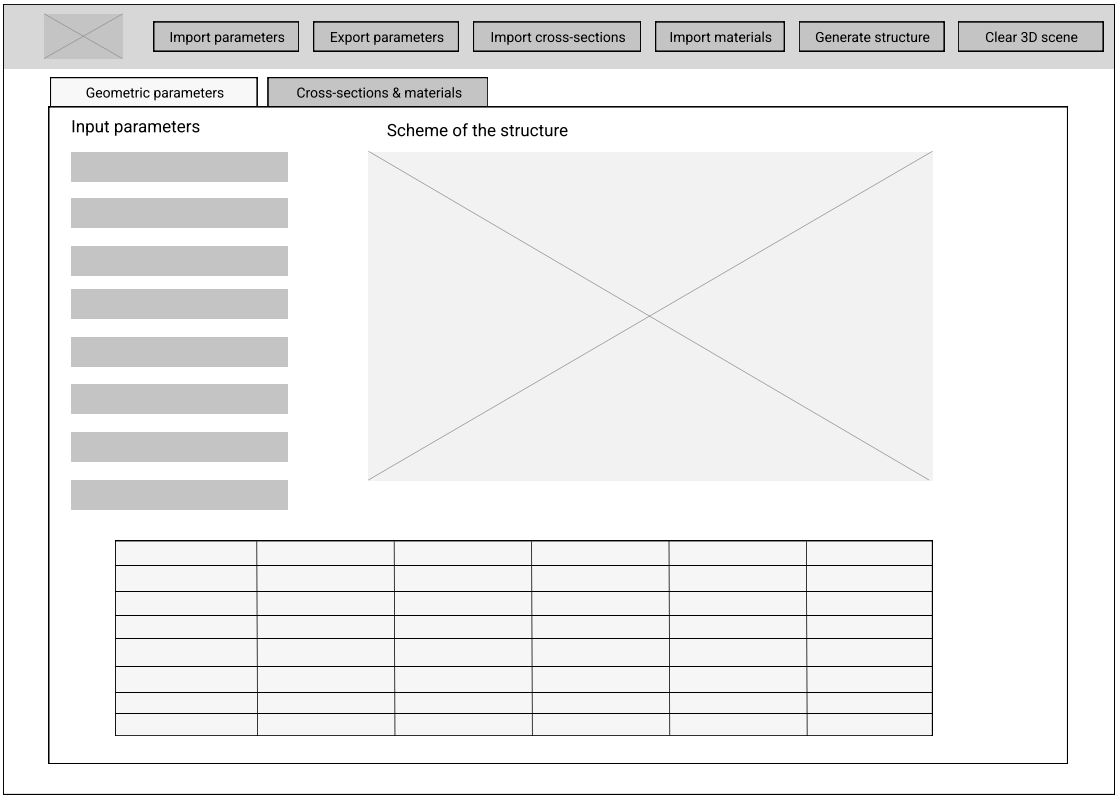
Obrázek 3.4 Byznys doménový model

Do modelu halové konstrukce byl zahrnut také submodel popisující sestavu základních os, ve stavebních modelech často označován jako *grid*. Jde o vyznačení hlavních směrů konstrukce, nejčastěji se tedy jedná o osy ve směru hlavních rámů a podélné osy spojující sloupy na jednotlivých stranách. Ve výškovém směru se pak často vyznačuje výšková úroveň hlav sloupů a případně vrchol střešní konstrukce. I přes to, že se jedná spíše o abstraktní objekt, pro účely této práce bylo vhodné zařadit ho jako součást modelu halové konstrukce. V praxi jsou osy konstrukce nedílnou součástí výkresové dokumentace, u halových konstrukcí obzvláště. Již od počátku se tento objekt používá i v digitálních stavebních modelech ze stejného důvodu, a to kvůli jednodušší orientaci v modelu (případně ve výkresu).

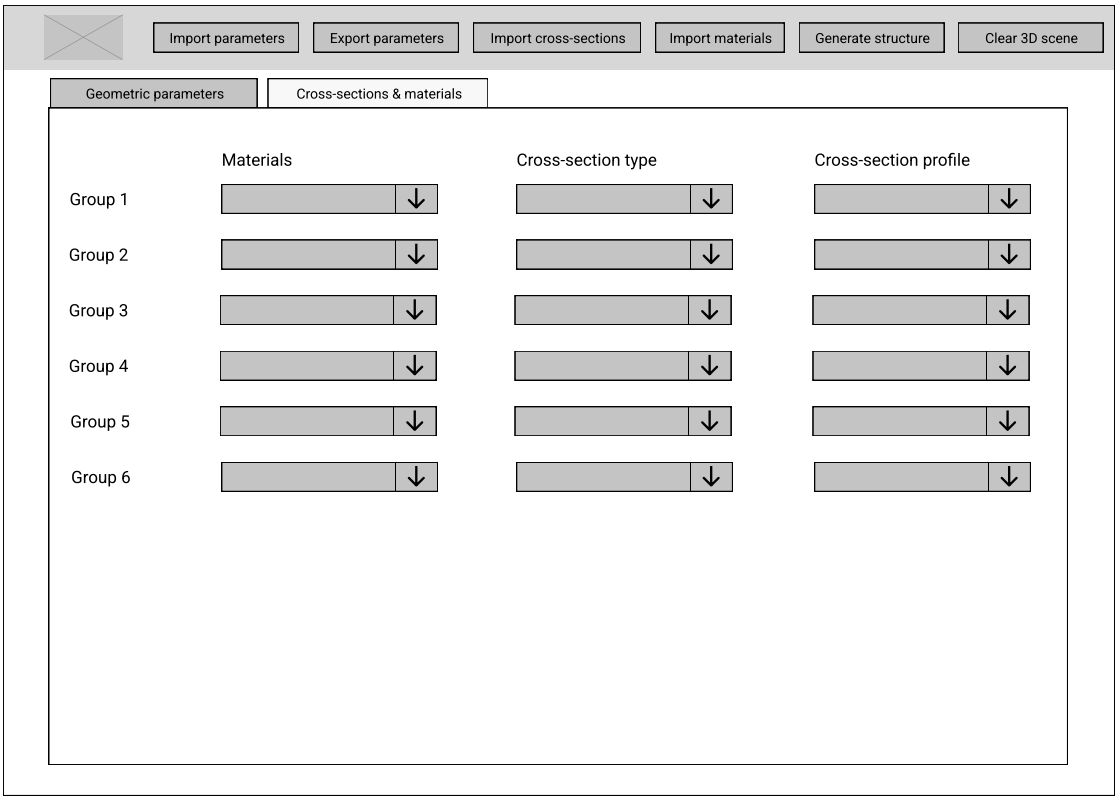
3.7 Návrh obrazovek uživatelského prostředí

Navrhovaná aplikace bude obsahovat 2 obrazovky. Na první z nich bude mít uživatel možnost zadat veškeré parametry týkající se geometrie výsledné konstrukce. V levé části budou v logickém pořadí umístěny jednotlivá pole pro uživatelské vstupy. Pravá část je vyhrazena pro schéma konstrukce. Jedná se o statický obrázek, který má pomoci uživateli s identifikací jednotlivých parametrů. Ve spodní části se bude nacházet tabulka, která bude mít dynamický počet řádků v závislosti na zadaném počtu konstrukčních polí. Z důvodu měnící se velikosti je tento prvek umístěn právě ve spodní části tak, aby neovlivňoval žádné konstantní prvky pod sebou. Návrh této obrazovky se nachází na obrázku 3.5.

Druhá obrazovka je pak vyhrazena pro nastavení materiálů a průřezů jednotlivých skupin prvků (obrázek 3.6). Obě obrazovky budou sdílet shodný vrchní panel, ve kterém se budou nacházet tlačítka s danými funkcionalitami. Mezi ty patří import/export parametrů, možnost načtení uživatelských seznamů a vygenerování konstrukce.



Obrázek 3.5 Obrazovka 1 - zadání geometrických parametrů



Obrázek 3.6 Obrazovka 2 - nastavení materiálů a průřezů

Kapitola 4
Implementace

Následující kapitola se zabývá samotnou implementací aplikace. Je v ní představena architektura výsledného řešení, konkrétní použití návrhových vzorů či přístupů, a také problémy, které se během implementační fáze vyskytly.

4.1 Výběr technologické platformy

4.1.1 Platforma a programovací jazyk

Pro volbu konkrétní platformy a programovacího jazyka jako takového byla nejdůležitější podpora ze strany aplikace Tekla Structures. Jelikož celé její API je vytvořené na platformě Microsoft .NET Framework, byla logickou volbou stejná platforma i pro navrhovanou aplikaci. Nejběžněji používaným jazykem na této platformě je C#. Vzhledem k tomu, že i veškeré ukázkové příklady a dokumentace související s API Tekla Structurers jsou napsány v tomto jazyku, byla i v tomto případě volba relativně jednoduchá.

Je nutné podotknout, že druhým aspektem, který byl při výběru zvažován, byla rozšířenost platformy (i jazyka) v komunitě stavebních inženýrů. Vyvíjená aplikace je určená právě pro stavební inženýry a mohlo by být užitečné, aby si ji oni sami byli schopni modifikovat v případě, že mají nějaké programátorské schopnosti. V tomto ohledu přicházel v úvahu i programovací jazyk Python, který je aktuálně asi nejpoužívanějším jazykem ve stavební komunitě. Ten je navíc všeobecně považován za jazyk velmi vhodný pro začátečníky či nezkušené programátory. V tomto případě by však výsledné řešení vyžadovalo použití rozšiřujících knihoven zajišťujících komunikaci s .NET platformou. Musel by být vytvořen samostatný .NET projekt, který by jako prostředník spojoval kód Pythonu a .NET API aplikace Tekla Structures. Technicky by to bylo možné, ale z podstaty věci se volba .NET platformy ve spojení s C# jazykem jevila jako přímočařejší rozhodnutí, ve kterém byla mnohem menší pravděpodobnost, že dojde v průběhu projektu k nějakým problémům s kompatibilitou.

4.1.2 Typ aplikace

Na platformě .NET Framework bylo na výběr mezi aplikací typu WinForms nebo WPF. Po technologické stránce by pro potřeby daného řešení vyhovovaly obě varianty. Po uvážení byl zvolen typ WPF z následujících důvodů. Jedním z nich byla rozšířitelnost aplikace do budoucna. V aplikacích typu WinForms je daleko složitější dosáhnout striktního oddělení UI a byznys logiky. To bylo pro implementaci velmi zásadní, jelikož jedním z požadavků, které byly na aplikaci stanoveny, bylo umožnění co možná nejsnadnější výměny UI. Dalším důvodem bylo také stáří a celková

aktuálnost technologie. Aplikace WinForms mají stále i dnes své využití. Většinou se však jedná o použití v nějakém větším systému, který novější technologii nepodporuje nebo má závislosti, které již nejsou v novějších frameworcích podporovány. V případě, kdy dochází k tvorbě kompletně nové aplikace, bylo zvoleno WPF i z důvodu očekávané delší podpory ze strany Microsoftu.

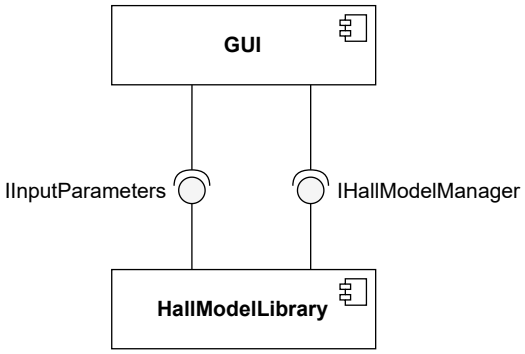
4.2 Implementace POC

Pro ověření možností API aplikace Tekla Structures byl na začátku vývoje implementován malý testovací projekt, v praxi často označovaný jako proof-of-concept (POC). Cílem bylo především zjistit, co vše obnáší spojení mezi vlastní aplikací a Teklou Structures. Jako vzor bylo využito ukázkových příkladů, které jsou dostupné pro vývojáře používající API Tekla Structures v oficiálním Github repozitáři společnosti. Jednalo se o velmi primitivní aplikaci, která vytvořila nosník a přenesla ho do 3D scény Tekla Structures. Zpětně lze tento krok hodnotit jako jeden z nejdůležitějších a nejhodnotnějších v celém vývojovém procesu. Na základě této zkušenosti byla sestavena celá architektura aplikace a bylo zjištěno, jaké části bude nutné vyhradit pro komunikaci s API, a také jaké objekty bude potřeba využívat.

4.3 Realizace Model-View-ViewModel vzoru

Jedním z požadavků na aplikaci bylo umožnění snadné budoucí výměny UI za jiné. Toto byl jeden z hlavních argumentů, který vedl k volbě WPF aplikace s použitím MVVM návrhového vzoru. Ten má sám o sobě spoustu dalších výhod, ale jedním z těch nejzásadnějších znaků je právě velmi dobré zapouzdření a oddělení kódu, který tvoří uživatelské prostředí. Stejně tak je striktně oddělena část, která má na starosti práci s byznys logikou a daty.

Kód byl tedy v prvotní fázi rozčleněn v rámci jednoho projektu do standardních balíčků na *Views*, *Models* a *ViewModels*. S rostoucím množstvím tříd a přidáváním funkcionalit došlo k oddělení modelů do samostatné komponenty. Byznys logika s potřebnými daty tak byla reprezentována jako samostatná knihovna (.dll), kterou lze z celého řešení vyjmout, nahradit, nebo použít v jiném projektu. Tato knihovna disponuje potřebným veřejným rozhraním, které slouží ke komunikaci s *ViewModelem*. Více podrobností o struktuře a uspořádání modelu lze nalézt v samostatné podkapitole 4.5.



Obrázek 4.1 Diagram komponent - oddělení modelů

Při spuštění aplikace dojde k vytvoření instance *TeklaHallModelManageru*, který je vstupním bodem do knihovny s byznys logikou. K samotné komunikaci mezi komponentami pak dochází pouze v rámci hlavního *ViewModelu*. Zde se při použití příslušného *Commandu* vytvoří a naplní instance objektu určená k zapouzdření všech uživatelských parametrů a následně se použije jako vstupní parametr při volání metody na vygenerování halové konstrukce. Propojení obou komponent je tedy definováno použitým rozhraním a dochází k němu pouze v jednom místě.

4.4 Komponenta uživatelského rozhraní

4.4.1 ViewModels

Tato komponenta obsahuje části patřící do prezentační vrstvy MVVM architektury. Konkrétně jde pak především o *Views* a *ViewModels*. Implementace *ViewModelu* byla realizována pomocí základní třídy *ViewModelBase*, která implementuje rozhraní *INotifyPropertyChanged*. To je nezbytné k tomu, aby docházelo ke správným aktualizacím změn stavu aplikace na základě změny vstupu od uživatele. Od této základní třídy pak dědí jednotlivé *ViewModely*. Díky použití základní třídy je přidání dalšího *ViewModelu* do projektu velmi snadné, navíc bez nutnosti duplikace kódu.

```
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?
            .Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Obrázek 4.2 Základní *ViewModel* implementující *INotifyPropertyChanged*

Implementovaný *ViewModel* pak obsahuje veškeré property, které uživatel svými vstupy nastavuje. Tyto informace jsou komunikovány mezi *View* a *ViewModelem* pomocí data bindingu a implementace již zmíněného rozhraní *INotifyPropertyChanged*.

```
//Vstupní parametr definovaný v XAML souboru
<TextBox
 Text="{Binding Span, UpdateSourceTrigger=LostFocus}"
 ... />

//Definice ve ViewModelu
private double _span;
public double Span
{
    get { return _span; }
    set {
            OnPropertyChanged(nameof(Span));
            _span = value;
            ...
        }
}
```

Obrázek 4.3 Data binding mezi *View* a *ViewModelem*

Druhým způsobem, jak dochází k interakci *View* s *ViewModelem*, jsou takzvané *Commandy*. Ty spravují jednotlivé akce, které se mají stát ve chvíli, kdy dojde k použití tlačítka v uživatelském rozhraní. Opět byl vytvořen základní *CommandBase*, který implementuje rozhraní *ICommand*. Všechny použité *Commandy* v projektu pak od této třídy dědí. Výhodou je, že přidání dalšího *Commandu* je jednodušší a nevyžaduje duplikovat již vytvořený kód.

Jedním z problémů, které bylo potřeba vyřešit, bylo nastavení výchozích hodnot parametrů při spuštění aplikace. Jedním z možných řešení je přiřadit konkrétní hodnotu proměnné rovnou při její definici ve *ViewModelu*. Tento přístup je funkční, ale při velkém množství nastavovaných hodnot se stává kód nepřehledný. Problematické je také nastavení výchozích hodnot pro properties, které jsou tvořené složitějším typem. V případě vyvíjené aplikace jde například o objekt *BracingItem*, který zastřešoval několik parametrů souvisejících se ztužujícím prvkem konstrukce. Výsledným použitým řešením tak nakonec bylo využití importu parametrů z předem připraveného souboru. Funkcionalita importu vstupních parametrů byla implementována na základě jednoho z funkčních požadavků. Díky tomu bylo její využití pro nastavení výchozích hodnot velmi přímočaré. Velkou výhodou tohoto přístupu je také to, že načítaný soubor má uživatel k dispozici v instalační složce aplikace. Může si ho tak libovolně upravit a dosáhnout chtěných výchozích hodnot. Stejný přístup byl použit i pro seznamy materiálů a průřezů, což opět přispívá k větší přizpůsobitelnosti aplikace konkrétním požadavkům uživatele.

4.4.2 Views

Dle návrhu měla aplikace obsahovat dvě obrazovky. Jednu s nastavením základních parametrů, které tvoří geometrii halové konstrukce, a druhou s konfigurací materiálů a průřezů pro jednotlivé části konstrukce. Jelikož spolu obě obrazovky velmi úzce souvisí a jejich struktura je navíc podobná, byly obě části implementovány v rámci jednoho *View*, pouze jako dvě různé záložky prvku zvaného *TabControl*.

V rámci *Views* je mimo jiné použit také prvek zvaný *DataGrid*. Ve vyvíjené aplikaci slouží k tabulkovému zobrazení nastavení parametrů souvisejících se ztužujícími prvky konstrukce. Pro snadnější použití a vytvoření kolekcí využívajících *binding* bylo vytvořeno několik speciálních objektů, které zastřešují vybrané parametry. Jedním z nich je například *GableWallBracing*, což je třída reprezentující ztužující prvek ve štítové stěně. Na obrázku 4.4 lze vidět část implementovaného *DataGridu*, který využívá kolekci těchto objektů (*ListOfGableWallFields*) a zároveň jednoduše přistupuje k jejich jednotlivým parametrům (*IsBraced* a *Name*).

```
<DataGrid
   ItemsSource="{Binding ListOfGableWallFields, UpdateSourceTrigger=LostFocus}"
   AutoGenerateColumns="False">
   <DataGrid.Columns>
       <DataGridTextColumn Header="Bay no."
        Binding="{Binding Name}"
        Width="60" />
       <DataGridCheckBoxColumn Header="Gable wall field bracing"
        Binding="{Binding IsBraced}" />
        ...
   </DataGrid.Columns>
   ...
</DataGrid>
```

Obrázek 4.4 Ukázka použití *DataGridu*

Při implementaci vzhledu uživatelského rozhraní bylo zvažováno použití knihoven pro stylování jednotlivých prvků. Žádná z prozkoumaných variant však plně nevyhovovala požadovanému vzhledu a tak je výsledné prostředí implementováno pomocí manuálně definovaných stylů. Ty byly použity pro jednotlivá tlačítka a především pak pro prvek *TabControl*. Styly jsou definované na začátku příslušného XAML souboru.

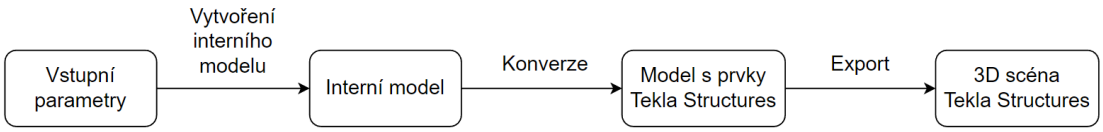
4.5 Struktura modelu

V této části došlo v průběhu implementace asi k největšímu počtu změn. Podoba jednotlivých modelů se proměnila v několika iteračních cyklech. Důvodem bylo především zřetelnější oddělení zodpovědností do jednotlivých tříd, což mimo jiné vedlo i ke zvýšení testovatelnosti a zpřehlednění struktury kódu.

V první fázi byla implementována zastřešující třída *HallModel*, která vystavovala veřejné rozhraní s jednotlivými potřebnými vstupními parametry. Ty byly plněny z *ViewModelu* a dále používány. Třída *HallModel* se starala o sestavení jednotlivých částí konstrukce a jejich export do Tekla Structures (dále již pouze jako TS). Velkým problémem tohoto řešení byla závislost všech modelů na knihovnách TS. Důsledkem toho bylo velmi problematické unit testování jednotlivých tříd. Nejen, že závislost se automaticky přenesla i do všech testů, ale ve většině případů nebylo možné pro instance tříd z knihoven TS nijak vytvářet tzv. mocky (náhrady za skutečné instance pro účely testování). Stejně tak nasimulování exportu v rámci jednotlivých testů bez nutnosti mít skutečně připojenou a otevřenou aplikaci TS bylo v tomto případě nemožné.

Řešením problému s exportem do TS bylo vytvoření třídy *TeklaInserter* a jejího příslušného rozhraní, která se o výsledný export do TS starala. Toto rozhraní bylo nutné do všech submodelů předávat a v rámci testů jej bylo možné nahradit. Jednalo se o funkční řešení, které ale bylo z hlediska kódu poměrně nepřehledné. Konstruktory všech tříd představujících jednotlivé části konstrukce musely být rozšířeny o jeden vstupní parametr a stále přetrvával problém se závislostí tříd na knihovnách TS.

Výše zmíněné skutečnosti vedly k přepracování struktury modelu, jehož hlavním cílem bylo striktně oddělit zodpovědnosti do jednotlivých tříd. Do samostatných celků tak bylo potřeba umístit části kódu starající se o vstupní parametry, o vytvoření modelu a o jeho následný export. Aby bylo možné implementovat třídy vytvářející model konstrukce bez závislosti na knihovnách TS, bylo nutné zavést interní model (schéma na obrázku 4.5). Ten využívá typologicky stejných prvků jako TS (body a nosníky) pouze s tím rozdílem, že se jedná o interní implementaci. Závislost na knihovnách TS tak zůstala omezena pouze na část kódu starající se o export do TS. V té dochází nejdříve ke konverzi interního modelu na model obsahující prvky z TS, a až poté k samotnému exportu.



Obrázek 4.5 Schéma zpracování zadaných parametrů

4.5.1 Veřejné rozhraní

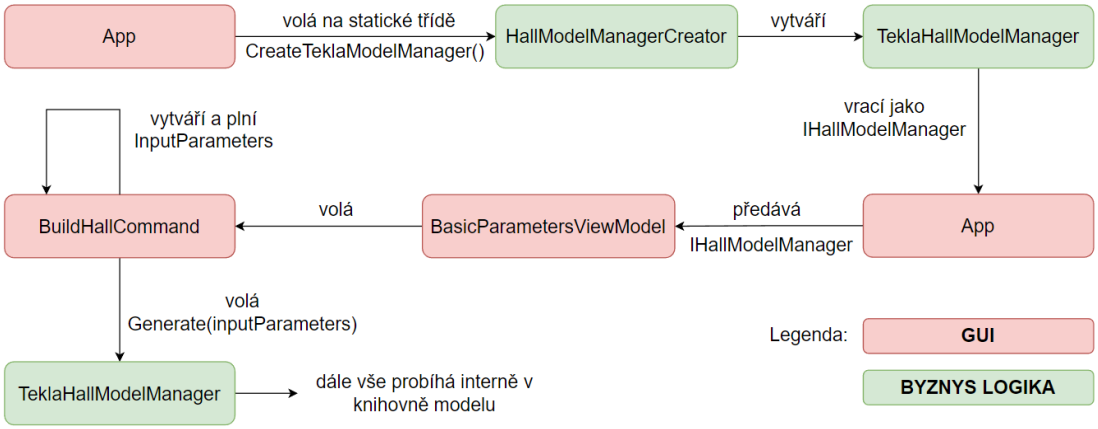
Vstupním bodem a zároveň řídícím prvkem celého projektu modelu je rozhraní *IHallModelManager*. To je veřejně přístupné a nabízí k dispozici metody *GenerateModel* a *Clear3Dscene*. Konkrétní implementací je třída *TeklaHallModelManager*, která je již závislá na knihovnách TS. O inicializaci a vytvoření instance této třídy se stará třída *HallModelManagerCreator*, která funguje na principu factory návrhového vzoru. Jedná se o veřejnou statickou třídu, která zajišťuje vytvoření konkrétní instance *HallModelManageru*. Vše se ale děje uvnitř této třídy interně. Třetí strana, v tomto případě ViewModel, si pouze zavolá metodu *CreateTeklaModelManager* a jako návratovou hodnotu dostane instanci chtěné třídy připravenou k použití (obrázek 4.6). Tento přístup umožnil velmi dobré zapouzdření celého modelu, kdy veřejně vystaveno je pouze nutné minimum funkcí a rozhraní.

```
public static class HallModelManagerCreator
{
    public static IHallModelManager CreateTeklaModelManager()
    {
        var builder = new HallModelBuilder();
        var exporter = new TeklaExporter();
        var teklaModel = new Model();
        Log.Logger = InitLogger();
        return new TeklaHallModelManager(builder, exporter, teklaModel);
    }
    ...
}
```

Obrázek 4.6 Ukázka statické třídy pro inicializaci hall model manageru

Kromě výše zmíněného *IHallModelManageru* je pak veřejná ještě třída *InputParameters* (a její příslušné rozhraní). Ta zapouzdřuje všechny parametry, které jsou nutné pro vygenerování modelu konstrukce. Naplněnou instanci této třídy je potřeba předat jako vstupní parametr do metody *GenerateStructure* volanou na *IHallModelManager* rozhraní. Proces komunikace mezi komponentami lze vidět na obrázku 4.7.

Všechny další části projektu obsahujícího modely jsou pouze interní a tudíž skryté pro jakoukoli třetí stranu, která tuto knihovnu používá. Výhodou tohoto zapouzdření je, že do budoucna je možné provést mnohé úpravy vnitřních implementací, nahradit je jinými, případně dále rozšiřovat funkcionalitu bez toho, aniž by to ovlivnilo existující integraci knihovny do jiných řešení. Asi největším benefitem je možnost rozšířit tento modul o propojení s jiným stavebním softwarem než s TS. K tomu je potřeba pouze doplnit novou implementaci *IHallModelManageru* a přidat nový exporter, zajišťující konverzi a export interního modelu. Třída *HallModelManagerCreator* by pak byla doplněna o novou metodu, vracející potřebnou implementaci *IHallModelManageru*. Ostatní části knihovny by nebyly dotčeny.



Obrázek 4.7 Schéma komunikace mezi komponentami

4.5.2 Tekla Hall Model Manager

Třída *TeklaHallManager* implementuje rozhraní *IHallModelManager* a řídí celý proces zpracování uživatelských dat, tvorbu interního modelu a následného exportu. Třída má viditelnost pouze interní v rámci knihovny *HallModelLibrary*, není tedy viditelná veřejně. O její instancování se stará *HallModelManagerCreator*, který nastaví veškeré instance tříd, které samotný model manager používá. Konkrétně jde o *HallModelBuilder*, který z uživatelských vstupů sestavuje interní model. A dále pak *TeklaExporter*, který na vstupu přijímá interní model a exportuje ho do prostředí Tekla Structures. Pro tento proces je také nutné mít instanci *TeklaStructures.Model*, která zajišťuje spojení do TS a umožňuje export prvků.

Největší výhodou této celé struktury je její zapouzdřenost. Samotná třída *TeklaHallModelManager* je konkrétní implementace rozhraní, která však není veřejně přístupná. Při dodržení rozhraní lze tedy udělat změny této vnitřní implementace bez toho, aniž by muselo dojít k nějaké změně na straně, která celou tuto knihovnu pro generování modelu používá. Stejně tak tento přístup umožňuje relativně snadno přidat implementaci novou, která by zajišťovala export modelu do jiného stavebního softwaru.

4.6 Správa zdrojového kódu

4.6.1 Verzování

Pro tvorbu kódu aplikace bylo využito Visual Studio 2022. Jedná se o komplexní nástroj, který poskytuje množství různých rozšíření a doplňků. V rámci bakalářské práce byl využíván jeho integrovaný manažer Git repozitáře, který umožňuje veškerou komunikaci a interakci s verzovacím prostředím. Konkrétně byl použit repozitář na portále Github. Kód byl vyvíjen v jednotlivých větvích a následně spojován do hlavní větve. Velkou výhodou tohoto přístupu byla možnost vracet se zpět k předchozím variantám kódu a také zkoušet si různé implementace bez ovlivňování kódu v hlavní větvi.

4.6.2 Kvalita kódu

Během implementace bylo také dbáno na čistotu a kvalitu kódu. Cílem bylo vytvořit čitelný, přehledný kód. Co se týče pojmenovávání, byly převážně dodržovány *coding guidelines* zveřejněné společností Microsoft pro .NET platformu. Výjimkou, kterou je vhodné zmínit, je označování názvů privátních proměnných podtržítkem na začátku.

Jednou z možností, jak zvýšit kvalitu kódu je jeho statická analýza. Ta se používá pro kontrolu kódu ještě před jeho spuštěním. Dají se pomocí ní odhalit nevhodné konstrukty, porušení definovaných zásad, nebo neefektivní části kódu. Při implementaci byl na statickou analýzu kódu použit nástroj SonarLint v podobě doplňku do Visual Studia. Mezi nejčastější případy, které byly tímto nástrojem odhaleny, patřilo neefektivní přistupování k položkám kolekce typu *List*, zapomenuté bloky komentářů a možnost optimalizace při inicializaci objektů.

Dalším použitým nástrojem je CodeMaid, opět dostupný pro Visual Studio v podobě doplňku. Hlavním účelem CodeMaid je formátování kódu tak, aby byl co nejlépe čitelný a jeho podoba byla konzistentní napříč celým projektem. Kromě sjednocování mezer a odsazení byla také používána funkce na odstraňování nepotřebných *usings*. Obzvláště nápomocným byl pak CodeMaid při práci v XAML souboru, kde docházelo velmi často k přidávání dalších úrovní a postupnému zanořování kódu. Díky automatickému formátování bylo jednoduché udržovat kód přehledný a čitelný.

4.6.3 Logování

Každá aplikace obsahuje nějaké chyby, které se dříve či později projeví u uživatele. V takovém případě je pak potřeba danou chybu opravit, nebo zákazníkovi objasnit, proč se program nechová tak, jak by očekával. A právě v těchto momentech vývojáři nejvíce ocení logování. Jedná se o zaznamenávání vybraných informací při používání programu, které mohou následně sloužit buď k analytickým účelům a nebo jako podklad při hledání chyb. Ve vyvíjené aplikaci je použit logger z externí knihovny Serilog.

Vzhledem k účelu aplikace bylo nakonfigurováno logování do textového souboru, který se nachází ve stejné složce jako spustitelný soubor aplikace. V implementaci je využito funkcionality *Log.ForContext<Type>*, která umožňuje zaznamenat do souboru aktuální umístění ve struktuře kódu. Pokud by uživatel narazil na nějaký problém, například pád aplikace, logovací soubor může posloužit jako první krok při analýze toho, co se stalo. Na obrázku 4.8 je ukázka použití loggeru v konkrétní třídě.

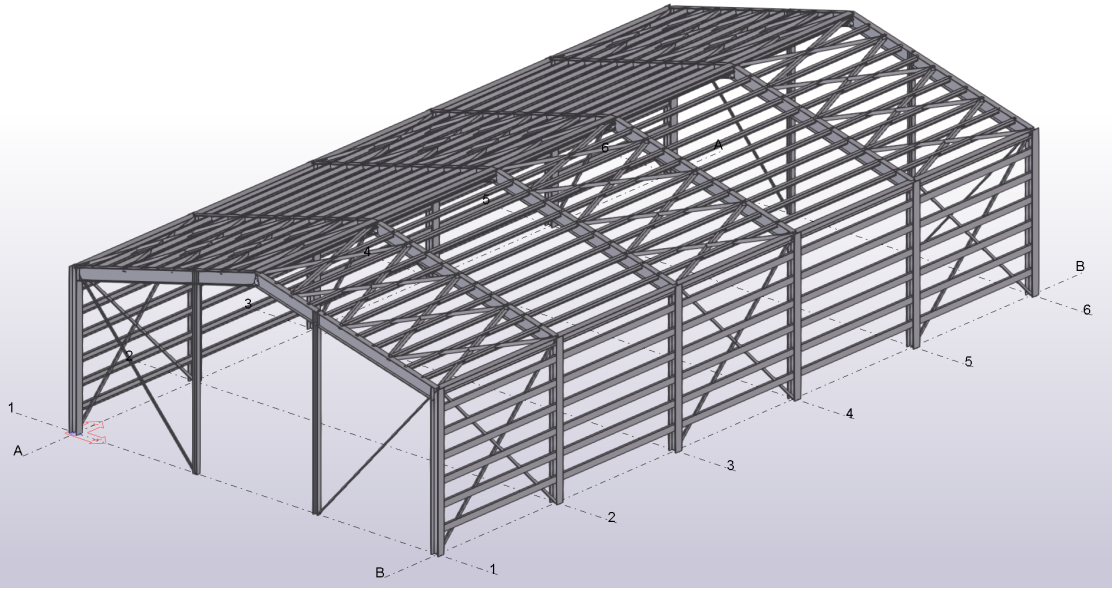
```
// konstruktor třídy FrameModel
public FrameModel(IInputParameters inputParameters)
{
    _logger = Log.ForContext<FrameModel>();
    ...
}

// zalogování úspěšné operace
public bool Build()
{
    ...
    _logger.Information("Frame at Y coordinate {0} was created.", GridDimY);
    return true;
}
```

Obrázek 4.8 Ukázka logování ve třídě *FrameModel*

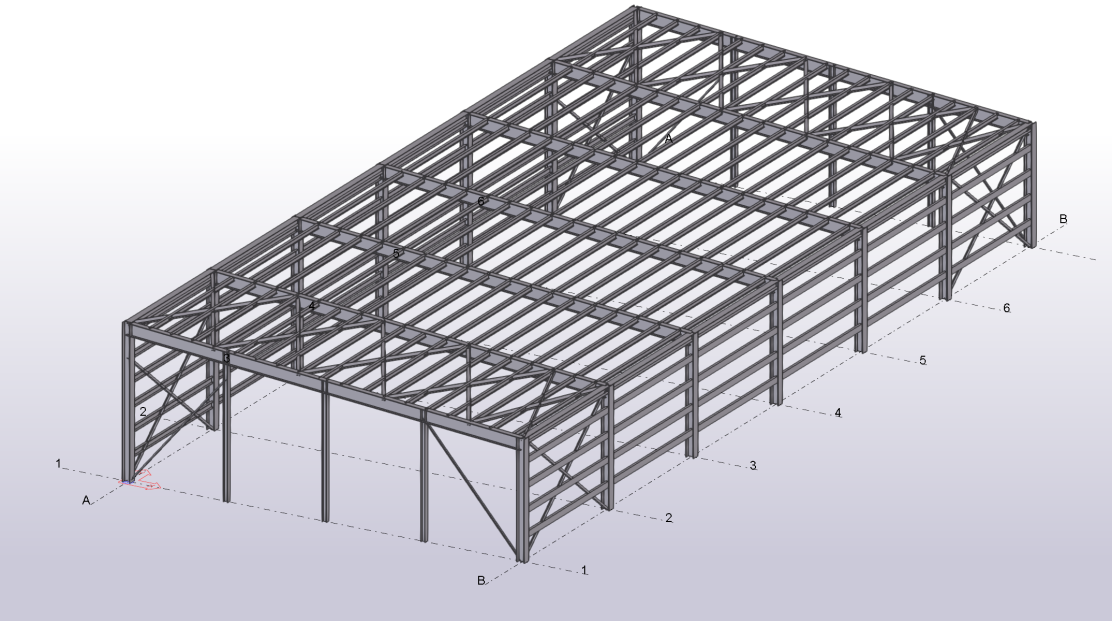
4.7 Ukázka generovaných konstrukcí

Výslednou podobu implementovaných uživatelských obrazovek lze nalézt v uživatelské příručce v příloze B. V následující části se nachází několik snímků, které prezentují výsledné 3D modely konstrukcí v Tekla Structures. Všechny uvedené modely byly vytvořeny pomocí navržené aplikace.

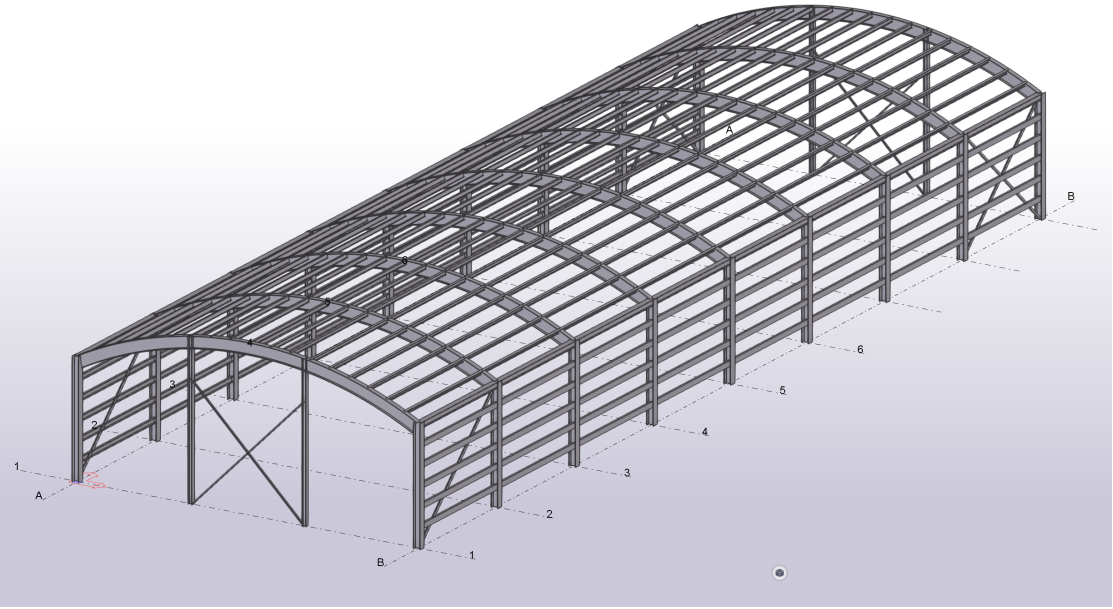
Na obrázku 4.9 je příklad modelu se sedlovou střešní konstrukcí. Hala je tvořena pěti konstrukčními poli. Ve štítových stěnách se nachází ztužidla jednostranného typu. V prvním, prostředním a posledním poli pak můžeme vidět použití křížových ztužidel pro boční stěny i střešní konstrukci. Na obrázku 4.10 je vygenerovaný model konstrukce s pultovým střešním nosníkem. V prvním a posledním poli jsou použita křížová ztužidla. Ztužení ve štítových stěnách je opět jednostranné. Na posledním obrázku 4.11 se pak nachází model s obloukovou střešní konstrukcí a osmi konstrukčními poli. Ztužující konstrukce ve střešní rovině byly tentokrát vynechány. Štítové stěny obsahují křížové ztužení ve svém prostředním poli.



Obrázek 4.9 Vygenerovaný model se sedlovou střešní konstrukcí



Obrázek 4.10 Vygenerovaný model s pultovou střešní konstrukcí



Obrázek 4.11 Vygenerovaný model s obloukovou střešní konstrukcí

Kapitola 5
Testování

Kapitola se zabývá procesem testování vyvíjeného kódu. Aplikace byla podrobena testování na dvou úrovních. První z nich je testování na úrovni tříd a metod. Druhým případem je pak testování na uživatelské úrovni.

5.1 Testování na úrovni tříd

Testování na úrovni tříd a metod (unit testování) slouží k ověření jednotlivých funkcionalit těch nejzákladnějších částí kódu. Jak již z názvu plyne, testovacím subjektem je v tomto případě jedna třída, případně jedna konkrétní metoda. Pokrytí implementovaného kódu unit testy jednak ověřuje správnost dané funkcionality, zároveň ale slouží jako velmi účinný nástroj při refactoringu kódu. V takovém případě dochází totiž k úpravám kódu, které mají zajistit větší přehlednost, lepší zapouzdřenost částí, nebo kód určitým způsobem optimalizovat. Tyto změny však nesmí mít dopad na samotnou funkčnost kódu. Testy na úrovni tříd právě toto dokáží ošetřit a pohlídat, že provedené změny se na funkcionalitě samotné nijak neprojevily.

V práci je toto testování použito pro projekt obsahující byznys logiku, tedy část s modely. Pro implementaci testů byl zvolen NUnit framework. Ten poskytuje několik nástrojů, které usnadňují samotné psaní, ale i správu testů. Jedním z nich jsou atributy, které se zadávají do hranatých závorek na začátek testu a lze díky nim definovat a konfigurovat následný test. V této bakalářské práci je často využíván atribut *TestCase*, kterým lze definovat odlišné vstupní parametry, pro jednotlivé běhy konkrétního testu (obrázek 5.1).

```
[TestCase("HEA300", "S355JR")]
[TestCase("IPE400", "S235")]
[TestCase("HEB240", "S420GP")]
public void ShouldCreateGableFrame(string crossSection, string material)
{ ... }
```

Obrázek 5.1 Ukázka využití atributů NUnit

Další pomocnou knihovnou využitou pro testování je NSubstitute. Ta umožňuje snadné vytváření objektů, které se tváří jako standardní definovaný objekt, ale poskytují možnost přesně specifikovat, jak se mají tyto objekty chovat při jejich použití. Běžně se pro takové objekty používá výraz mock. Ve vytvořených testech je používaná standardní formulace pro vytváření těchto objektů pomocí *Substitute.For<Type>* (obrázek 5.2). Díky jejich využití je samotný kód testu mnohem přehlednější, protože není nutné kompletně instancovat všechny potřebné objekty, stačí

pouze vytvořit jejich mocky a poté nadefinovat, jak se tyto objekty zachovají při zavolání pouze konkrétní metody nebo atributu. Pro správnou funkčnost těchto falešných objektů bylo potřeba důsledně používat v kódu vždy rozhraní místo konkrétní implementace objektu.

```
[Test]
public void ShouldCreateGableWallBracing()
{
    IHallModel hallModel = Substitute.For<IHallModel>();
    IInputParameters inputParams = Substitute.For<IInputParameters>();
    IFrameModel frame1 = Substitute.For<IFrameModel>();
    ...
    inputParams.Span.Returns(12000);
    ...
}
```

Obrázek 5.2 Ukázka použití NSubstitute v testech

Jednotlivé testy byly strukturované do 3 hlavních částí, často označovány jako Arrange-Act-Assert. První část se zabývá přípravou potřebných dat a objektů (například i vytváření mocků). Ve druhé části pak dochází k samotnému vykonání testu, použití objektu, nebo zavolání potřebné metody. Na závěr přichází verifikační část, ve které se ověřuje výsledek. Zde byla pro implementaci použita knihovna FluentAssertions. Ta umožňuje psát validační výroky testů v syntaxi velmi podobné řeči a tím zpřehledňuje celý test. V ukázce na obrázku 5.3 lze vidět použití zmiňované knihovny v jednom z jednotkových testů.

```
[Test]
public void ShouldCreateGableWallBracing()
{
    //Arrange
    IHallModel hallModel = Substitute.For<IHallModel>();
    ...
    //Act
    var sut = new PurlinsModel(inputParams, hallModel);
    var result = sut.Build();

    //Assert
    result.Should().Be(true);
    sut.PurlinsLHSBeams.Count.Should().Be(2);
    ...
}
```

Obrázek 5.3 Ukázka použití FluentAssertions v testech

5.2 Testování na uživatelské úrovni

Druhým typem testů, které byly při vývoji použity jsou testy na uživatelské úrovni. Ty se vyznačují tím, že dochází k testování kompletní aplikace ve finální podobě, podle předepsaného scénáře, který má simulovat jeden ze standardních uživatelských případů užití. Součástí je přesný popis jednotlivých kroků tak, aby test zvládl provést i uživatel, který aplikaci běžně nezná a nepoužívá. Pro tyto kroky jsou definované i jednoznačné výsledky, které se musí po vykonání kroku zkontrolovat.

V praxi jsou tyto testy většinou prováděny v rámci procesu vydání nové verze aplikace. Častá je také jejich integrace do celé vývojové či organizační platformy. Jako příklad lze uvést Azure DevOps od společnosti Microsoft. Na této platformě mohou být uložené jak scénáře testů, tak i jejich průběh a výsledky. Vzhledem k rozsahu bakalářské práce bylo zvoleno jednodušší a méně komplexnější řešení. Testovací scénáře jsou dostupné v příloze A v tabulkovém formátu určeném pro aplikaci Microsoft Excel. Jsou zde připravené formuláře, ve kterých tester nalezne popis jednotlivých kroků a zároveň vyhrazená místa pro vyplnění výsledků, poznámek, apod.

Test Case ID:	Název:	Datum poslední editace:	
TC_01	Import parametrů	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
2	Klikni na tlačítko "Import parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
3	V pravé dolní části dialogu změň typ souboru na "All files". Poté vyber libovolný soubor, který není formátu .xml. Volbu potvrď kliknutím na "Otevřít".	Objevila se notifikační hláška se zprávou "Parameters couldn't be imported, try to check format of the file.".	
4	Zavři hlášku.	Parametry v aplikaci zůstaly beze změny.	
5	Opět klikni na tlačítko "Import parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
6	Vyber soubor "test-input-1.xml" a klikni na "Otevřít".	Objevila se informační hláška, že parametry byly úspěšně naimportovány.	
7	Zavři hlášku.	Parametry v aplikaci nyní odpovídají hodnotám z načteného souboru.	
			PASS

Obrázek 5.4 Ukázka testovacího scénáře

Pro tvořenou aplikaci bylo vytvořeno 9 testovacích scénářů. Jejich zaměření odpovídá jednotlivým funkcionalitám aplikace:

- import parametrů,
- export parametrů,
- import seznamu průřezů,
- import seznamu materiálů,
- vyčištění 3D scény,
- vygenerování haly s obloukovým střešním nosníkem,
- vygenerování haly s jednostranným sklonem,
- vygenerování haly s oboustranným sklonem,
- validace parametrů.

V závěrečné fázi vývoje byly tyto uživatelské testy použity k odladění aplikace. Testy byly vykonány ve 3 iteracích, během kterých bylo nalezeno několik chyb. Všechny nalezené nedostatky byly odstraněny a aplikace byla odevzdána ve stavu, kdy všemi testy úspěšně procházela.

Kapitola 6
Závěr

V rámci této bakalářské práce byl implementován plně funkční prototyp aplikace, která umožňuje na základě zadaných vstupních parametrů vygenerovat uživateli skelet stavebního modelu ocelové halové konstrukce do 3D scény aplikace Tekla Structures. Tím bylo dosaženo hlavního vytyčeného cíle práce. Vytvořená aplikace splňuje všechny požadavky, které byly definovány v zadání práce. Uživatel má možnost použít v aplikaci své vlastní seznamy materiálů a průřezů. Dále je možné uložit aktuálně zadané parametry do samostatného XML souboru a později je opět nahrát zpět do aplikace.

Prvním krokem, a také jedním z dalších cílů práce, byla rešerše současných dostupných řešení. Následovala analýza všech požadavků, výběr vhodných technologií a základní návrh architektury. Během implementace došlo k několika změnám oproti původnímu návrhu. Mezi ty největší lze zařadit změnu struktury aplikace, kdy ve výsledné implementaci je kladen daleko větší důraz na oddělení zodpovědností jednotlivých částí kódu. Veškeré změny jsou popsány a zdůvodněny v implementační části práce. Aplikace byla po své implementaci také otestována na uživatelské úrovni pomocí testovacích scénářů, což byl také jeden z cílů práce.

Výsledná aplikace má značný potenciál pro rozšíření do budoucna. Díky implementaci interního modelu a jeho důslednému oddělení od ostatních částí je možné nahradit Teklu Structures jiným stavebním modelovacím softwarem. Nahrazení by vyžadovalo přidání nové implementace části starající se o konverzi a export. Tvorba interního modelu na základě uživatelských vstupů by zůstala nedotčena. Funkcionalitu programu by bylo možné rozšířit i pomocí přidání více vstupních parametrů. Lze zmínit například umožnění asymetričnosti konstrukce, možnost zadat přesahy prvků tvořících střešní konstrukci nebo umístit počátek modelu do libovolných souřadnic.

Příloha A

Testovací scénáře

V této příloze se nachází jednotlivé testovací scénáře, které byly použity pro testování na uživatelské úrovni.

Test Case ID:	Název:	Datum poslední editace:	
TC_01	Import parametrů	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
2	Klikni na tlačítko "Import parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
3	V pravé dolní části dialogu změň typ souboru na "All files". Poté vyber libovolný soubor, který není formátu .xml. Volbu potvrď kliknutím na "Otevřít".	Objevila se notifikační hláška se zprávou "Parameters couldn't be imported, try to check format of the file.".	
4	Zavři hlášku.	Parametry v aplikaci zůstaly beze změny.	
5	Opět klikni na tlačítko "Import parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
6	Vyber soubor "test-input-1.xml" a klikni na "Otevřít".	Objevila se informační hláška, že parametry byly úspěšně naimportovány.	
7	Zavři hlášku.	Parametry v aplikaci nyní odpovídají hodnotám z načteného souboru.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_02	Export parametrů	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
2	Klikni na tlačítko "Export parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
3	Klikni na tlačítko "Zrušit".	Objevila se informační hláška, že soubor nebyl vybrán a parametery tak nebyly exportovány.	
4	Zavři informační hlášku.	Parametry v aplikaci zůstaly beze změny a žádný soubor nebyl vytvořen.	
5	Zopakuj krok 2.	Otevřel se uživatelský dialog pro výběr souboru.	
6	Zadej název souboru a klikni na "Uložit".	Objevila se informační hláška, že parametry byly úspěšně exportovány do vybraného souboru.	
7	Zavři informační hlášku.	Parametry v aplikaci zůstaly beze změny. Zkontroluj, že požadovaný soubor se vytvořil, a že obsahuje parametry shodné s těmi v aplikaci.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_03	Import seznamu průřezů	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
2	Klikni na tlačítko "Import cross-sections".	Otevřel se uživatelský dialog pro výběr souboru.	
3	V pravé dolní části dialogu změň typ souboru na "All files". Poté vyber libovolný soubor, který není formátu .xml. Volbu potvrď kliknutím na "Otevřít".	Objevila se notifikační hláška se zprávou "Cross-sections couldn't be imported, try to check format of the file.".	
4	Zavři hlášku.	Parametry v aplikaci zůstaly beze změny.	
5	Zopakuj krok 2.	Otevřel se uživatelský dialog pro výběr souboru.	
6	Vyber soubor "test-input-2.xml" a klikni na "Otevřít".	Objevila se informační hláška, že průřezy byly úspěšně naimportovány.	
7	Zavři hlášku.	Seznamy průřezů v aplikaci nyní odpovídají hodnotám v importovaném souboru.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_04	Import seznamu materiálů	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
2	Klikni na tlačítko "Import materials".	Otevřel se uživatelský dialog pro výběr souboru.	
3	V prvé dolní části dialogu změň typ souboru na "All files". Poté vyber libovolný soubor, který není formátu .xml. Volbu potvrď kliknutím na "Otevřít".	Objevila se notifikační hláška se zprávou "Materials couldn't be imported, try to check format of the file.".	
4	Zavři hlášku.	Parametry v aplikaci zůstaly beze změny.	
5	Zopakuj krok 2.	Otevřel se uživatelský dialog pro výběr souboru.	
6	Vyber soubor "test-input-2.xml" a klikni na "Otevřít".	Objevila se informační hláška, že průřezy byly úspěšně naimportovány.	
7	Zavři hlášku.	Seznamy průřezů v aplikaci nyní odpovídají hodnotám v importovaném souboru.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_05	Vyčištění 3D scény	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci Tekla Structures a otevři nový projekt.	Tekla Structures je spuštěna a je v ní otevřen nový projekt.	
2	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
3	Klikni na "Generate structure".	Objevila se informační hláška, že konstrukce byla úspěšně exportována a ve 3D scéně Tekla Structures je vygenerovaný model.	
4	Zavři hlášku a klikni na tlačítko "Clear 3D scene".	Objevila se informační hláška, že 3D scéna byla úspěšně vyčištěna. V Tekla Structures je nyní prázdná 3D scéna.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_06	Vygenerování haly s obloukovým nosníkem	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci Tekla Structures a otevři nový projekt.	Tekla Structures je spuštěna a je v ní otevřen nový projekt.	
2	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
3	Klikni na "Import parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
4	Vyber soubor "test-curved-hall.xml" a klikni na "Otevřít".	Objevila se informační hláška, že parametry byly úspěšně naimportovány.	
5	Zavři hlášku.	Parametry v aplikaci nyní odpovídají hodnotám z načteného souboru. Typ střešní konstrukce je nastaven na "curved".	
6	Klikni na "Generate structure".	Objevila se informační hláška, že konstrukce byla úspěšně exportována a ve 3D scéně Tekla Structures je vygenerovaný model.	
7	Zavři hlášku.	V aplikaci Tekla Structures zkontroluj, že vygenerovaný model odpovídá zadaným parametrům.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_07	Vygenerování haly s jednostranným sklonem	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci Tekla Structures a otevři nový projekt.	Tekla Structures je spuštěna a je v ní otevřen nový projekt.	
2	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
3	Klikni na "Import parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
4	Vyber soubor "test-pitch-hall.xml" a klikni na "Otevřít".	Objevila se informační hláška, že parametry byly úspěšně naimportovány.	
5	Zavři hlášku.	Parametry v aplikaci nyní odpovídají hodnotám z načteného souboru. Typ střešní konstrukce je nastaven na "pitch".	
6	Klikni na "Generate structure".	Objevila se informační hláška, že konstrukce byla úspěšně exportována a ve 3D scéně Tekla Structures je vygenerovaný model.	
7	Zavři hlášku.	V aplikaci Tekla Structures zkontroluj, že vygenerovaný model odpovídá zadaným parametrům.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_08	Vygenerování haly s oboustranným sklonem	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci Tekla Structures a otevři nový projekt.	Tekla Structures je spuštěna a je v ní otevřen nový projekt.	
2	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
3	Klikni na "Import parameters".	Otevřel se uživatelský dialog pro výběr souboru.	
4	Vyber soubor "test-gable-hall.xml" a klikni na "Otevřít".	Objevila se informační hláška, že parametry byly úspěšně naimportovány.	
5	Zavři hlášku.	Parametry v aplikaci nyní odpovídají hodnotám z načteného souboru. Typ střešní konstrukce je nastaven na "gable".	
6	Klikni na "Generate structure".	Objevila se informační hláška, že konstrukce byla úspěšně exportována a ve 3D scéně Tekla Structures je vygenerovaný model.	
7	Zavři hlášku.	V aplikaci Tekla Structures zkontroluj, že vygenerovaný model odpovídá zadaným parametrům.	
			PASS

Test Case ID:	Název:	Datum poslední editace:	
TC_09	Validace parametrů	20.11.2023	
Step ID	**Popis**	**Očekávaný výsledek**	Výsledek testu
1	Spusť aplikaci Tekla Structures a otevři nový projekt.	Tekla Structures je spuštěna a je v ní otevřen nový projekt.	
2	Spusť aplikaci HallGenerator.	Otevřelo se okno s aplikací HallGenerator na obrazovce s nastavením geometrických parametrů.	
3	Do pole "Span" zkus zadat jakýkoliv jiný znak než číslo.	Jiný než číselný znak nebylo možné zadat. Žádné úpravy v hodnotě pole se neprovedly.	
4	Do pole "Count of bays" zadej hodnotu 6.	Tabulka s údaji o ztužidlech se přegenerovala a obsahuje nyní 6 řádků.	
5	Do pole "Spacing between frames" zadej hodnotu 100 a klikni na tlačítko "Generate structure.	Objevila se informační hláška, že konstrukce nemohla být exportována z důvodu nevalidního vstupu. V informační zprávě se také nachází validní rozsah hodnot.	
6	Zavři hlášku a oprav hodnotu na validní (ze zmíněného rozsahu).	Ve 3D scéně Tekla Structures nedošlo k žádným změnám, konstrukce nebyla vygenerována.	
7	Zopakuj krok 5 pro pole "Spacing of purlins".	Objevila se informační hláška, že konstrukce nemohla být exportována z důvodu nevalidního vstupu. V informační zprávě se také nachází validní rozsah hodnot.	
8	Zavři hlášku a opravd hodnotu na validní (ze zmíněného rozsahu).	Ve 3D scéně Tekla Structures nedošlo k žádným změnám, konstrukce nebyla vygenerována.	
			PASS

Příloha B

Uživatelská příručka

B.1 Instalace

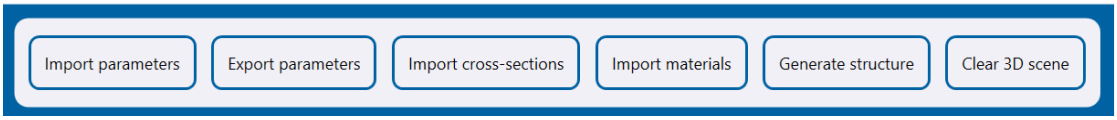
Ke zprovoznění aplikace je potřeba zkopírovat složku s názvem *compiled* do libovolného adresáře v uživatelském počítači. Poté lze program spustit pomocí souboru *HallGenerator.exe*, který se nachází ve zkopírované složce. Pro správný běh aplikace je nutné mít na počítači nainstalovaný také .NET Framework 4.8. Instalační soubory k .NET lze najít na oficiálních webových stránkách Microsoftu zde.

B.2 Použití aplikace

Pro plné využití aplikace je nutné mít nainstalovaný a licencovaný software Tekla Structures 2023 SP2. Aplikaci lze spustit a používat i bez puštěné Tekly Structures, ale její funkcionalita je pak omezena. V následujících částech jsou popsané jednotlivé obrazovky s možnostmi, které lze v aplikaci použít.

B.2.1 Hlavní panel

Na přiloženém obrázku B.1 lze vidět hlavní panel obsahující tlačítka. Panel se vyskytuje na obou obrazovkách programu. Každé tlačítko představuje jednu konkrétní funkcionalitu. Jejich výčet je následující.



Obrázek B.1 Hlavní panel s tlačítky

Import parameters Umožňuje naimportovat veškeré parametry zadání z připraveného souboru ve formátu XML. Při selhání importu zkontrolujte, zda má zvolený soubor správnou koncovku (označení formátu .xml). Soubor lze také otevřít v jakémkoli textovém editoru podporujícím formát XML a v případě nutnosti ho upravit dle libosti. Pro import je vyžadováno, aby soubor byl kompletní, tedy aby obsahoval vyplněné všechny parametry, které se v aplikaci používají. Import pouze vybraných parametrů není podporován.

Export parameters Vyexportuje všechny vstupní parametry do samostatného XML souboru. Ten lze později použít pro import. Vyexportovaný soubor lze také otevřít v jakémkoli textovém editoru podporujícím daný formát a hodnoty ručně upravit.

Import cross-sections Umožňuje naimportovat seznam vlastních průřezů. Pro import je nutné použít soubor ve formátu XML se správným obsahem. Jako šablonu pro přípravu vlastního seznamu lze použít soubor *css-default.xml*, který se nachází ve složce *Libraries* v instalační složce programu. První úroveň je tvořena seznamem tvarů. Uvnitř každé položky se pak nachází seznam s již konkrétními průřezy. Pro správné namapování průřezů do Tekla Structures je nutné použít totožné názvy i ve vlastním seznamu.

Import materials Umožňuje naimportovat seznam vlastních materiálů. Pro import je nutné použít soubor ve formátu XML se správným obsahem. Jako šablonu pro přípravu vlastního seznamu lze použít soubor *materials-default.xml*, který se nachází ve složce *Libraries* v instalační složce programu. Pro správné namapování materiálů do Tekla Structures je nutné použít totožné názvy i ve vlastním seznamu.

Clear 3D scene Použitím tlačítka dojde k vymazání všech objektů (včetně rastru os) z aktuálně otevřeného modelu v Tekla Structures. Toto se doporučuje provést vždy před generováním konstrukce tak, aby bylo zajištěno, že generovaný model nebude ovlivněn stávajícími prvky ve scéně.

Generate structure Na základě zadaných parametrů vygeneruje model do 3D scény Tekla Structures. Pro generování je nutné mít spuštěnou Teklu Structures s aktivním projektem a otevřenou 3D scénou. Po kliknutí na tlačítko dojde nejdříve k ověření platnosti zadaných údajů. Validní rozsahy hodnot jsou popsané v kapitole B.2.2. Před generováním neprobíhá automatické vyčištění scény.

B.2.2 Zadání geometrických parametrů

Tato obrazovka se otevře po spuštění programu. Slouží k zadání geometrických vstupních parametrů, které popisují halovou konstrukci. Zadávané hodnoty mají své platné rozsahy a omezení, která jsou uvedená v následujícím seznamu spolu s popisem jednotlivých hodnot.

Type of roof Typ střešní konstrukce, která může být sedlová (gable), pultová (pitch) nebo zakřivená (curved).

Span [mm] Hodnota definující vzdálenost mezi osami sloupů v hlavní vazbě konstrukce (šířka konstrukce). Validní rozsah hodnot je 1 000–100 000 mm.

Spacing between frames [mm] Osová vzdálenost mezi jednotlivými vazbami. Tato hodnota je aplikována na všechny pole konstrukce. Validní rozsah hodnot je 500–50 000 mm.

Count of bays Počet polí halové konstrukce tvořených hlavními vazbami. Na základě této hodnoty se generuje tabulka se zadáním informací souvisejících s konstrukcí ztužidel. Ta má stejný počet řádků jako počet polí konstrukce. Validní rozsah je 1–50 polí.

Column height [mm] Výška sloupů (měřena mezi hlavou a patou sloupů) v hlavních vazbách. Hodnota je pro obě strany stejná. Výjimkou je, pokud uživatel zvolí střešní konstrukci s jednostranným sklonem. Pro tuto variantu je mu umožněno zadat rozdílnou výšku levých a pravých sloupů. Zadaná hodnota vždy platí pro všechny hlavní vazby. Validní rozsah hodnot je 1 000–50 000 mm.

Roof height [mm] Vzdálenost mezi hlavou sloupů a vrcholem střešní konstrukce. V případě, že uživatel zvolí střechu s jednostranným sklonem je tato volba neaktivní. Validní rozsah hodnot je 100 mm až polovina hodnoty rozpětí konstrukce.

Spacing of purlins [mm] Osová vzdálenost mezi jednotlivými vaznicemi. Hodnota je měřena ve střešní rovině, v případě oblouku se jedná o délku měřenou na zakřiveném nosníku. Vaznice jsou kladeny směrem od krajních rohových nosníků směrem k vrcholu střešní konstrukce. Vzdálenost mezi poslední vaznicí a vrcholovým vazníkem je maximálně rovna této hodnotě. Validní rozsah hodnot je 200 mm až čtvrtina rozpětí konstrukce.

Spacing of side rails [mm] Osová vzdálenost mezi jednotlivými paždíky na bočních stěnách. První paždík je umístěn ve zvolené vzdálenosti od úrovně paty sloupů. Poslední pak maximálně zvolenou vzdálenost od osy rohového nosníku spojujícího jednotlivé vazby. Validní rozsah hodnot je 200 mm až polovina výšky sloupů.

Count of gable mullions Počet doplňkových štítových sloupků. Do počtu se nazapočítávají hlavní sloupy rámu. Validní rozsah hodnot je 1–30 sloupků.

Spacing of gable mullions [mm] Osová vzdálenost jednotlivých štítových sloupků. Stejná vzdálenost je aplikována pro všechna pole. Validní rozsah hodnot je 200 mm až polovina rozpětí konstrukce.

Side bracing Volba, zda chce uživatel v bočních stěnách daného pole použít ztužující prvky.

Side bracing type Typ ztužení bočních stěn. Na výběr je ztužení křížem, levostranné nebo pravostranné.

Side bracing height [mm] Výška, do které zasahují boční ztužidla. Tato hodnota nesmí přesahovat výšku sloupů. Miniální hodnota je 500 mm.

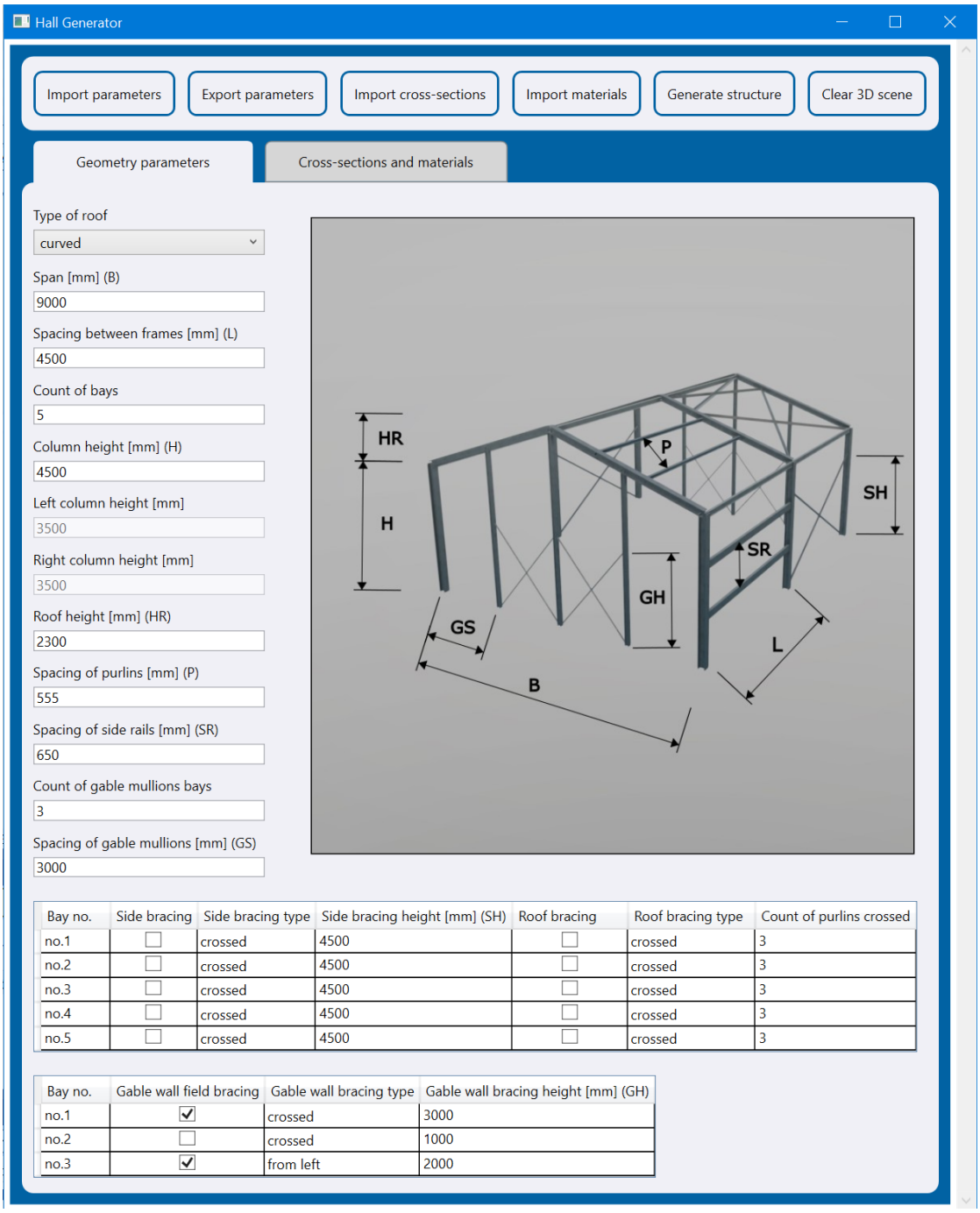
Roof bracing Volba, zda chce uživatel pro dané pole aplikovat ztužení ve střešní rovině.

Roof bracing type Typ ztužení ve střešní rovině. Na výběr je ztužení křížem, levostranné nebo pravostranné.

Count of purlins crossed Počet vaznic, která ztužidla kříží. Krajní vaznice, v jejichž počátečních a koncových bodech jsou ztužidla ukotvena, se do této hodnoty nezapočítávají. Ztužidla jsou kladena směrem od krajních rohových nosníků k vrcholu střešní konstrukce. Minimální hodnota je 1 a maximem je celkový počet vaznic.

Gable wall field bracing Volba, zda chce uživatel pro dané pole ve štítové stěně aplikovat ztužující prvky.

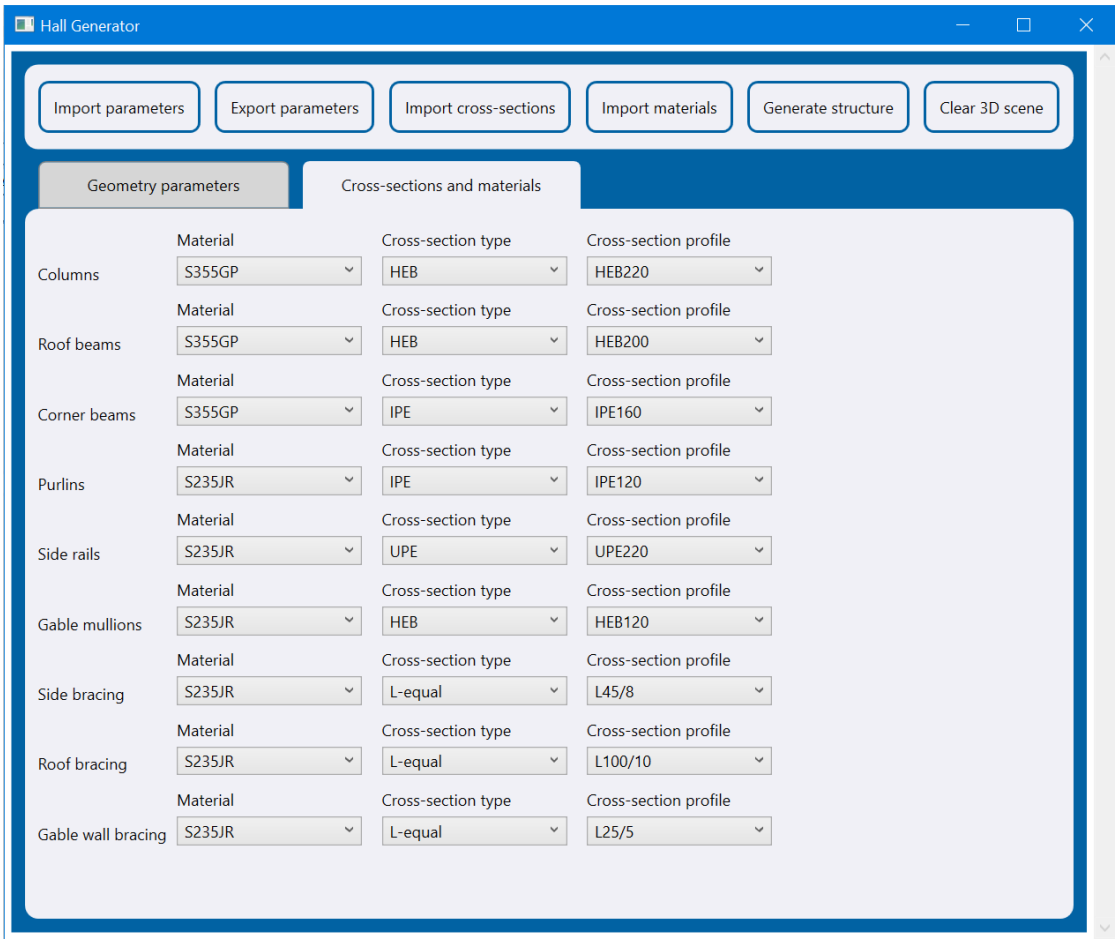
Gable wall bracing height [mm] Výška, do které zasahují ztužující prvky v daném poli štítové stěny. Minimální hodnota je 1 000 mm. Maximální výška je dána výškou sloupů.



Obrázek B.2 Snímek obrazovky se zadáním geometrických údajů

B.2.3 Zadání materiálů a průřezů

Po kliknutí na záložku *Cross-sections and materials* se otevře obrazovka, kde je možné nastavit materiál a průřez pro jednotlivé skupiny prvků. Výchozí seznamy lze nahradit uživatelskými pomocí importu souborů ve formátu XML. Jako šablona pro vytvoření uživatelského seznamu průřezů lze použít soubor *css-default.xml* nacházející se ve složce *Libraries* v instalační složce programu. Podobně pro seznam materiálů lze využít soubor *materials-default.xml* ve stejném umístění.



Obrázek B.3 Snímek obrazovky se zadáním materiálů a průřezů

Bibliografie

1. TRIMBLE. About Trimble Inc. *Trimble.com* [online]. ©2023. [cit. 2023-11-26]. Dostupné z: https://www.trimble.com/en/about.
2. Agenda de Cursos Engenharia. *Webstore.grupohct.com.br* [online]. 2019. [cit. 2023-11-26]. Dostupné z: https://webstore.grupohct.com.br/wp-content/uploads/2019/04/tekla_steel.jpg.
3. EASTMAN, C.M. BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors. Wiley, 2008. ISBN 9780470185285.
4. BUILDINGSMART. OpenBIM. *Buildingsmart.org* [online]. ©2023. [cit. 2023-11-26]. Dostupné z: https://www.buildingsmart.org/about/openbim/.
5. TRIMBLE. *Tekla Structures 2023 SP1* [online]. 2023. [cit. 2023-11-26]. Dostupné z: https://download.tekla.com/.
6. TEKLA. Components. *Support.tekla.com* [online]. ©2023. [cit. 2023-11-26]. Dostupné z: https://support.tekla.com/doc/tekla-structures/2023/det_getting_started_overview.
7. About Grasshopper. *Grasshopper3d.com* [online]. ©2023. [cit. 2023-11-27]. Dostupné z: https://www.grasshopper3d.com/.
8. TEKLA. Grasshopper-Tekla Live Link. *Support.tekla.com* [online]. 2021. [cit. 2023-11-27]. Dostupné z: https://support.tekla.com/help/tekla-structures/not-version-specific/grasshopperteklalink.
9. Grasshopper for Engineers. *Thecomputationalengineer.com* [online]. ©2023. [cit. 2023-11-27]. Dostupné z: https://thecomputationalengineer.com/videos/grasshopper-for-engineers/.
10. DLUBAL. Historie a čísla. *Dlubal.com* [online]. ©2001-2023. [cit. 2023-11-27]. Dostupné z: https://www.dlubal.com/cs/spolecnost/o-spolecnosti-dlubal-software/fakta-a-cisla.
11. DLUBAL. Dlubal RFEM – Generator of 3D Hall. *Dlubal.com* [online]. 2013. [cit. 2023-11-27]. Dostupné z: https://www.dlubal.com/en/support-and-learning/learning/videos/000689.
12. GMBH, Dlubal Software. *Dlubal RFEM 5.33.01* [online]. 2023. [cit. 2023-11-27]. Dostupné z: https://www.dlubal.com/cs/stahovani-a-informace/zkusebni-verze-zdarma/stahnout-zkusebni-verzi.
13. HISTRUCT. About Us. *HiStruct.com* [online]. ©2023. [cit. 2023-11-27]. Dostupné z: https://www.histruct.com/company/about-us.

14. Steel Building Configurator. *HiStruct.com* [online]. ©2023. [cit. 2023-11-27]. Dostupné z: https://app.histruct.com/buildingconfigurator/en/model/editopen.
15. HISTRUCT. What is the HiStruct Building Configurator. *HiStruct.com* [online]. ©2023. [cit. 2023-11-27]. Dostupné z: https://www.histruct.com/products-and-services/building-configurator.
16. MICROSOFT. Overview of .NET Framework. *Microsoft.com* [online]. Březen 2023. [cit. 2023-11-27]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview.
17. HARTINGER, David Čápka. Lekce 1 – Úvod do C# a .NET frameworku. *Itnetwork.net* [online]. Září 2022. [cit. 2023-11-27]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework.
18. AGGARWAL, Anshul. Introduction to .NET Framework. *Geeksforgeeks.org* [online]. Únor 2023. [cit. 2023-11-27]. Dostupné z: www.geeksforgeeks.org/introduction-to-net-framework/.
19. GEORGE, Andy De. WPF overview. *Microsoft.com* [online]. Prosinec 2021. [cit. 2023-11-27]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/introduction-to-wpf?view=netframeworkdesktop-4.8.
20. JECHA, Tomáš. Úvod do Windows Presentation Foundation. *Dotnetportal.cz* [online]. Leden 2012. [cit. 2023-11-27]. Dostupné z: https://www.dotnetportal.cz/clanek/196/Uvod-do-Windows-Presentation-Foundation-WPF-.
21. What is NUnit. *Nunit.org* [online]. ©2023. [cit. 2023-11-27]. Dostupné z: https://nunit.org/.
22. MAZE, Code. Introduction to Unit Testing With NUnit in C#. *Code-maze.com* [online]. Březen 2022. [cit. 2023-11-27]. Dostupné z: https://code-maze.com/csharp-nunit-unit-testing/.
23. DASH, Debendra. Introduction To NUnit Testing Framework. *C-sharpcorner.com* [online]. Listopad 2023. [cit. 2023-11-27]. Dostupné z: https://www.c-sharpcorner.com/article/introduction-to-nunit-testing-framework/.
24. FIAFFÉ, Julien. Easy mocking with NSubstitute. *Codingjourneyman.com* [online]. Září 2015. [cit. 2023-11-27]. Dostupné z: https://codingjourneyman.com/2015/09/07/easy-mocking-with-nsubstitute/.
25. MAZE, Code. Effective Mocking With NSubstitute in .NET. *Code-maze.com* [online]. Srpen 2023. [cit. 2023-11-27]. Dostupné z: https://code-maze.com/csharp-effective-mocking-with-nsubstitute/.
26. DOOMAN, Dennis. About. *Fluentassertions.com* [online]. ©2023. [cit. 2023-11-27]. Dostupné z: https://fluentassertions.com/about/.
27. BLUMHARDT, Nicholas. Configuration Basics. *Serilog.net* [online]. Listopad 2021. [cit. 2023-11-27]. Dostupné z: https://github.com/serilog/serilog/wiki/Configuration-Basics.
28. TOVARYS, Jan. How To Start Logging With Serilog. *Betterstack.com* [online]. Březen 2023. [cit. 2023-11-27]. Dostupné z: https://betterstack.com/community/guides/logging/how-to-start-logging-with-serilog/.
29. SONARSOURCE. SonarLint for Visual Studio 2022. *Marketplace.visualstudio.com* [online]. Duben 2021. [cit. 2023-11-29]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=SonarSource.SonarLintforVisualStudio2022.
30. CADWALLADER, Steve. CodeMaid. *Marketplace.visualstudio.com* [online]. Prosinec 2009. [cit. 2023-11-29]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=SteveCadwallader.CodeMaid.

31. MICROSOFT. Model-View-ViewModel (MVVM). *Microsoft.com* [online]. Duben 2022. [cit. 2023-11-27]. Dostupné z: `https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm`.

32. WATSON, Miles. Creating Better Applications with MVVM. *Dev.to* [online]. Leden 2021. [cit. 2023-11-27]. Dostupné z: `https://dev.to/mileswatson/a-beginners-guide-to-mvvm-using-c-wpf-241b`.