FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Administrační rozhraní pro projekt Věnná města českých královen
Student:	Bc. Michal Martinek
Vedoucí:	Ing. Petr Pauš, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem práce je tvorba webového administračního rozhraní pro projekt Věnná města českých královen (VMČK), který aktuálně běží na fakultě.

1. Seznamte se s projektem VMČK a jeho privátním API.
2. Analyzujte požadavky pro frontend administračního rozhraní. Rozhraní musí minimálně podporovat přihlašování uživatelů, zobrazování 3D modelů a metadat z databáze, editaci metadat.
3. Navrhněte schvalovací proces pro modely z databáze (připomínkování, hlášení chyb, schválení).
4. Analyzujte vhodné knihovny a nástroje pro tvorbu webového frontendu a zobrazování 3D modelů (A-Frame, Three.js, aj.).
5. Navrhněte vhodné uživatelské rozhraní pro zobrazování 3D modelů a interakci s nimi.
6. Dle výsledků analýzy implementujte a proveďte vhodné testování.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 20. listopadu 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Administrační rozhraní pro projekt Věnná města českých královen

Bc. Michal Martinek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petr Pauš, Ph.D

26. května 2020

Poděkování

Rád bych poděkoval vedoucímu mé práce Ing. Petr Paušovi, Ph.D za vedení a cenné rady. Také bych rád poděkoval Ing. Monice Martinkové, L.LM za korektury této práce a dále pak Bc. Michalu Maněnovi za rady týkající se vícekriteriální analýzy variant použité v této práci. Mé díky si zaslouží i ochotní testeři – Mgr. Daniela Borošová, Michael Botur, Bc. Adam Janas a Erik Klemš a rovněž i Václav Procházka, který mi poskytl cenný názor na vzhled a použitelnost uživatelského rozhraní během jeho návrhu.

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 26. května 2020 . . . . . .

Odkaz na tuto práci

Martinek, Michal. *Administrační rozhraní pro projekt Věnná města českých královen.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce je zaměřena na vývoj administračního rozhraní pro projekt Věnných měst českých královen. Součástí této práce je analýza funkčních a nefunkčních požadavků a případů užití. Tato práce se zaměřuje i na analýzu frontendových technologií a výběr jedné z nich pro použití v tomto rozhraní. Dále pak je součástí návrh uživatelského rozhraní, jeho uživatelské testování a implementace prototypu.

Klíčová slova Věnná města českých královen, administrace, frontend, UI

Abstract

This thesis is focused on the development of the administration frontend for Dowry Towns of the Queens of Bohemia project. Parts of this work are analysis of functional and non-functional requirements and use-cases. This work also contains analysis of the frontend technologies and process of selecting one of them. Other parts include user interface design, user testing and prototype implementation.

Keywords Dowry Towns of the Queens of Bohemia, administration, frontend, UI

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Projekt *Věnných měst českých královen* se zabývá českým dějinným fenoménem královských věnných měst. Jeho cílem je primárně prezentovat tuto část dějin široké veřejnosti, přičemž jednou z podob má být průvodce ve formě mobilní aplikace a informačního portálu obsahujících historicky věrné modely budov a dalších objektů v těchto městech. Tyto modely budou využívany i pro použití ve virtuální realitě, čemuž se věnují další studenti na Fakultě informačních technologií ČVUT.

Pro správu těchto modelů je nutné vytvořené administrační rozhraní, aby bylo možné s těmito modely jednoduše manipulovat a schvalovat je, což je činnost, kterou se budou zabývat lidé z několika oborů pro optimální výsledky jak z hlediska grafické, tak i historické správnosti.

Pro tyto účely budu tedy vyvíjet administrační rozhraní, jehož analýzou, jakož i analýzou použitelných technologií se budu zabývat v této práci. Dále pak budu navrhovat samotný schvalovací proces a uživatelské rozhraní pro manipulaci s těmito modely. Toto rozhraní budu následně implementovat a testovat jeho použitelnost.

Kapitola 1

Cíl práce

Cílem této diplomové práce je tvorba webového administračního rozhraní pro projekt Věnná města českých královen. Tento projekt má vytvořit historicky věrný, virtuální model těchto měst napříč časovou osou od 14. století do současnosti. Na tomto úkolu se dohromady podílí více lidí – grafici navrhují modely, historici je revidují a tyto modely pak mají být uloženy centrálně, aby byly dostupné uživatelům. Pro tyto účely je nutné vyvinout administrační rozhraní.

V této práci se tedy budu zabývat analýzou požadavků na toto administrační rozhraní. Na základě analýzy pak bude navrženo uživatelské rozhraní pro manipulaci s modely a jejich metadaty včetně jejich schvalování. Další částí návrhu je i samotný schvalovací proces 3D modelů mezi grafiky a historiky.

Pro implementační část je rovněž vhodné analyzovat použitelné technologie a knihovny pro tvorbu webového frontendu a zobrazování 3D modelů, čímž se v této práci budu rovněž zabývat.

Na základě analýzy a návrhu budu následně implementovat toto administrační rozhraní a provedu uživatelské testování.

Kapitola 2

Analýza

2.1 O projektu a struktuře Věnných měst českých královen

Cílem projektu *Věnných měst českých královen* (dále jako VMCK) je výzkum a experimentální vývoj okolo českého dějinného fenoménu královských věnných měst, které tvoří specifickou kategorii mezi českými historickými městy. Projekt se zaměřuje na prezentaci tohoto kusu historie široké veřejnosti pomocí nástrojů historické geografie a počítačové grafiky. Hlavním výstupem tohoto snažení má být mobilní aplikace a informační portál s historickým průvodcem včetně vyhotovených zrekonstruovaných 3D modelů budov a dalších artefaktů. Mobilní aplikace by měla nabídnout jedinečný zážitek pomocí rozšířené reality, kdy bude možné si prohlédnout podobu města napříč časem přímo na místě. [1]

Tento projekt je víceoborový a proto se na jeho realizaci podílí více subjektů, jedním z nich je i skupina na Českém vysokém učení technickém v Praze na Fakultě informačních technologií, která zodpovídá za technickou část tohoto projektu a má na starosti jak vývoj mobilních aplikací včetně rozšířené reality, systému na ukládání 3D modelů a dalších dat, tak i vývoj dalších pomocných nástrojů. Jedním z nich je administrační rozhraní pro potřeby správy a schvalování, jehož vývojem se budu v této práci zabývat. Zabývám se čistě klientskou částí, toto rozhraní se má opírat o privátní API (Application Programming Interface), jež v bylo v době psaní této práce rovněž ve vývoji. Dále je ve vývoji virtuální realita (dále pod zkratkou VR), aplikace pro pohyb v městě a VR aplikace pro zobrazování menších objektů. Mimo to je ve vývoji i Android aplikace pro smíšenou realitu pro zobrazení budov, či modul pro rozpoznání obrazu a určení polohy.

Pro pochopení dalších částí mé práce si dovolím popsat, co se bude v administračním rozhraní spravovat. Podstatou celého tohoto systému budou 3D modely budov a jiných historických artefaktů jako např. kašen, soch, cest,

mostů atd. Tyto všechny artefakty budou zastřešeny entitou *structure*. V systému ale nemusí být pouze jeden kompletní 3D model tohoto artefaktu. Cílem je totiž nabízet podobu napříč časem, takže budou v systému vyhotoveny 3D modely podob z různých historických období. Pro co největší zážitek se rovněž budou tvořit různé varianty jednotlivých modelů např. zasněžená varianta, varianta v dešti atd. O těchto variantách a historických verzích budeme mluvit jako o jednotlivých entitách *3D object*. Takže entita *structure* bude mít v konečné podobě celou řadu vymodelovaných podob v podobě entit *3D object* a na základě aktuálního počasí a vybraného historického období se koncovým uživatelům zobrazí správný model *3D object*.

2.2 Funkční požadavky na aplikaci

Jako požadavek bereme v potaz cokoliv, co má vliv na návrh aplikace. Těchto požadavků je vícero typů. Existují business požadavky, uživatelské požadavky či např. funkční požadavky, kterými se budu v této sekci zabývat. Funkční požadavky popisují, chování a rozhraní aplikace za definovaných podmínek. Jejich cílem je popsat, co bude nutné naimplementovat, aby rozhraní umožnilo naplnit uživatelské požadavky.[2] Proto v této sekci popíšu nutné funkcionality administračního rozhraní pro projekt VMCK.

F1 Přihlášení

Uživatel se může přihlásit do administračního rozhraní pomocí e-mailu a hesla. Po přihlášení přebírá minimálně jednu ze 4 rolí – *Historik*, *Grafik*, *Modelář* a *Admin*. S rolí *Admin* může uživatel využívat veškeré funkce aplikace nehledě na omezení rolí.

F2 Registrace

Uživatel v roli *Admin* může registrovat další uživatele, editovat a mazat jejich profily a přiřazovat jim uživatelské role.

F3 Zobrazit entity structure a metadata k nim

Přihlášení uživatelé mohou zobrazit všechny entity *structure*, např. reprezentující budovy, včetně jejich metadat. Tato data mohou taky filtrovat podle stavu schválení a řadit dle jejich jména a/nebo data přidání.

F4 Vytváření structure

Uživatelé v roli *Historik* mohou vytvořit nový záznam *structure* včetně jeho metadat.

F5 Editace metadat

Historici mohou editovat metadata k *structure*.

F6 Nahrávání 3D modelů

Modeláři mohou nahrávat nové verze modelů k *structure*, textury a další související soubory.

F7 Schvalování modelů

Modely vytvořené *Modeláři* procházejí schvalováním ze strany *Grafiků* a *Historiků*, kteří zkontrolují správnost modelu a případně model vrátí *Modeláři* s přípomínkou k přepracování. Po zapracování oprav dochází znovu ke kontrole a případně dalším opravám až k finálnímu schválení.

F8 Generování variant

Uživatelé v roli *Grafik* mohou generovat varianty pomocí automatického transformačního systému pomocí definovaných tranformací.

F9 Administrace sekvencí tranformací

V rámci systému jsou definované sekvence tranformací s výchozími hodnotami, toto nastavení mohou editovat *Admini*.

F10 Mazání dat

Uživatelé v roli *Admin* mohou smazat libovolný objekt.

F11 Odhlášení

Uživatel se může odhlásit.

F12 Změna hesla a osobních informací

Uživatel má v administračním rozhraní vlastní profil, který může měnit včetně hesla.

2.3 Nefunkční požadavky na aplikaci

Nefunkční požadavky jsou doplňkem funkčních požadavků, popisují vlastnosti aplikace, podle kterých se má chovat, a definují omezení na spolehlivost, použitelnost, platformu, výkonnost a mnoho dalších. Mohou se týkat i uživatelského rozhraní.[2] Pro administrační rozhraní jsem určil po dohodě se svým vedoucím tyto nefunkční požadavky:

N1 Podpora prohlížečů

Aplikace bude podporovat běh v těchto prohlížečích: Google Chrome 56, Firefox 60, Microsoft Edge 16, Safari 11, iOS Safari 11, Chrome for Android 64.

N2 Responzivita

Aplikace bude podporovat zobrazení i na mobilech, tabletech a jiných médiích, které budou moci používat internetové prohlížeče vypsané v N1.

N3 Bezpečnost

Mimo přihlášení budou veškeré akce vyžadovat autentizaci uživatele. Některé akce budou vyžadovat autorizaci. Komunikace se serverem bude využívat šifrované komunikace a aplikace bude ošetřovat známé bezpečnostní mezery pro zabránění potencionálních útoků např. XXS (Cross-site scripting), CSRF (Cross-Site Request Forgery), Clickjacking a podobně.

N4 Použitelnost

Administrativní rozhraní bude mít jednoduché rozhraní dostupné v českém jazyce.

2.4 Případy užití

Případy užití slouží k lepšímu a přesnějšímu popisu požadavků. Popisují jak uživatelé budou používat systém, co pro ně systém dělá, a umožňují získat přehled, co systém musí umět, aby tyto případy užití naplňoval. Případy užití se často mimo čistě textové formy doplňují diagramy popisujícími případy užití dohromady, tak i ve složitějších případech průběh jednotlivých případů užití.[3]

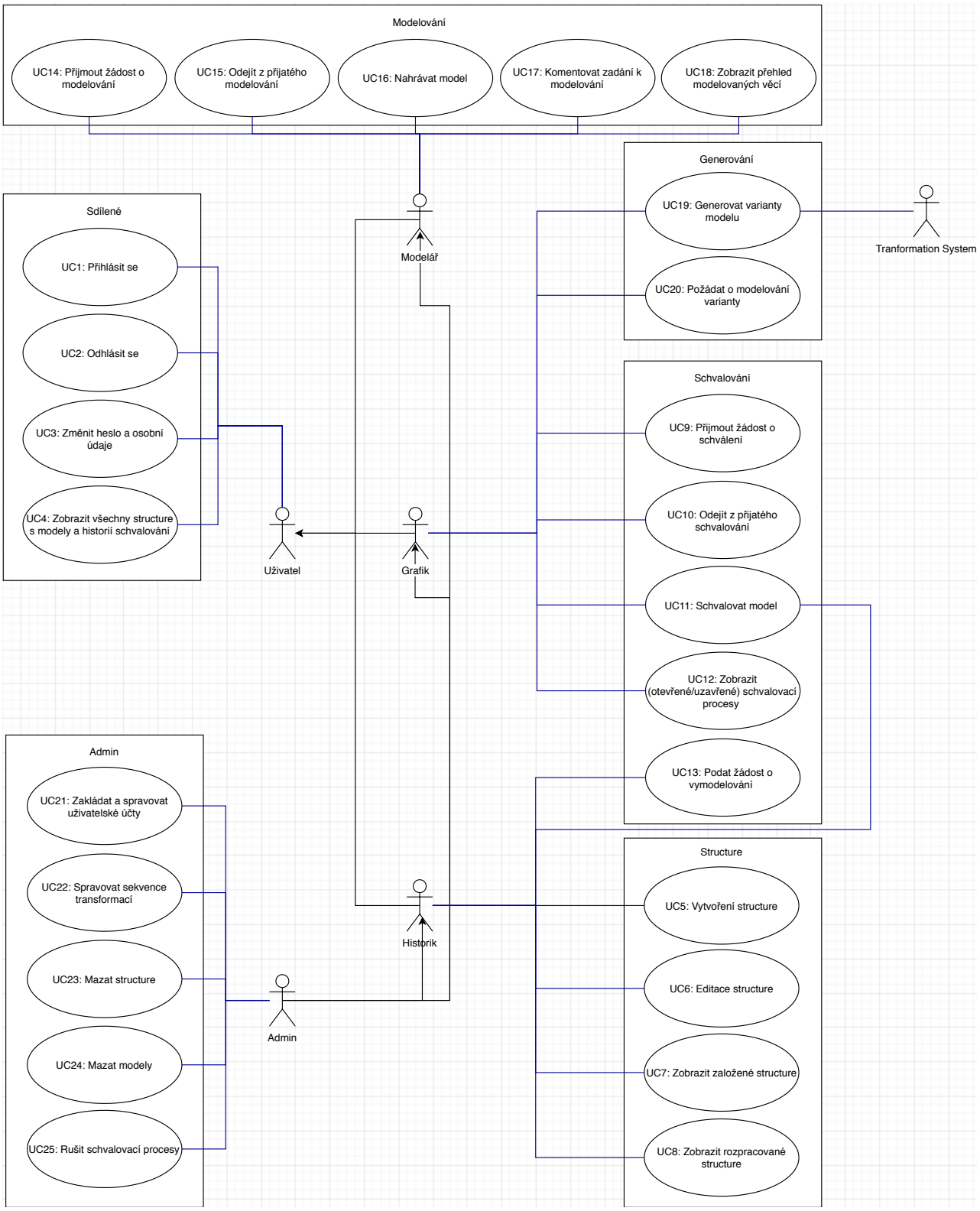
Diagram případů užití je vyobrazen na obrázku 2.1 a skládá se pro přehlednost z několika různých modulů, do kterých jsou vloženy jednotlivé případy užití. V rámci diagramu je vícero aktérů, jeden z nich je aktér *Transformation System*, který znázorňuje službu, jež se bude starat o automatické vytváření variant ze základního modelu. Tyto varianty budou zobrazovat model ku příkladu zasněžený, v dešti, zestárlý a podobně. Zbytek aktérů pak slouží ke znázornění jednotlivých uživatelských rolí. Základní aktér je *Uživatel*, který dokáže provádět základní interakce jako přihlášení, správa vlastního profilu, zobrazení *Structure*, což jsou interakce sdílené se všemi podrolemi. Aktér *Historik* se stará o vytváření záznamů, přípravu a předávku materiálů k modelování a schvalování modelu z hlediska historické správnosti. Aktér *Grafik* se stará o schvalování modelů z hlediska grafické správnosti, generování a schvalování variant základního modelu. Aktér *Modelář* vytváří na základě materiálů

model a má možnost vyjadřovat se k zadání. Aktér *Admin* pak může provádět všechny interkace aktérů *Historik*, *Grafik* a *Modelář* a spravovat systém – spravovat uživatele, sekvence tranformací pro generování variant, manipulovat s daty.

Tyto případy užití se mapují na funkční požadavky popsané v kapitole 2.2. Cílem je propojit a sjednotit obě dvě části, které svým způsobem popisují požadavky na systém, a taky zajistit pokrytí všech funkčních požadavků případy užití. Mapování případů užití je zobrazeno v tabulce 2.1, ve které jsou horizontálně zaneseny funkční požadavky. Vertikálně jsou zanesené případy užití a jejich pokrytí funkčních požadavků, kde znak + značí, že daný případ užití popisuje daný funkční požadavek.

Tabulka 2.1: Pokrytí funkčních požadavků případy užití.

Případy užití	Funkční požadavky											
	1	2	3	4	5	6	7	8	9	10	11	12
UC01	+											
UC02											+	
UC03												+
UC04			+									
UC05				+								
UC06					+							
UC07			+									
UC08			+									
UC09							+					
UC10							+					
UC11							+					
UC12			+									
UC13							+					
UC14							+					
UC15							+					
UC16						+						
UC17							+					
UC18			+									
UC19								+				
UC20							+					
UC21		+										
UC22									+			
UC23										+		
UC24										+		
UC25										+		



Obrázek 2.1: Diagram případů užití.

2.5 Analýza workflow systémů

Podstatná část administračního rozhraní řeší schvalovací proces mezi historiky a grafiky. Což není nic nového a pro tyto účely je vyvinuta řada řešení. Schvalování i jiné procesy lze řešit pomocí workflow systémů. Samotné workflow je schéma/postup nějaké komplexnější činosti/procesu, jako v tomto případě schvalování. Pro jeho definici, správu a řízení jsou vyvinuty worfklow systémy, které umožňují zaznamenávání průběhu provádění úloh, předávání informaci či dokonce samotnou správu digitálního obsahu, jímž se procesy často zabývají. Tak je tomu i v případě projektu VMCK, kde se schvalují vytvořené modely.

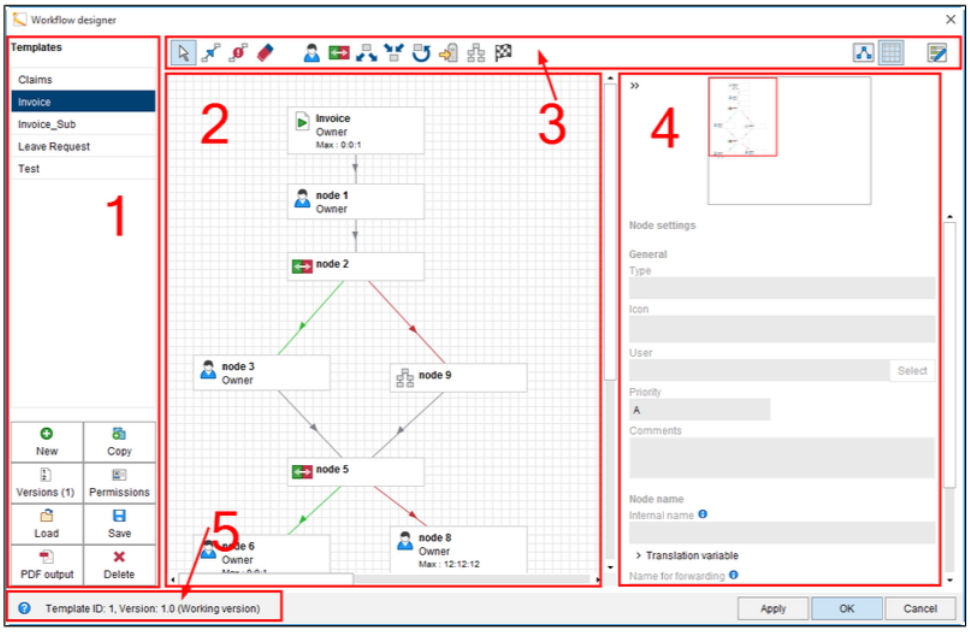
V této kapitole se budu zabývat, jak vypadá typická funkcionalita těchto schvalovacích systémů a zda by schvalovací část administračního rozhraní nemohla být řešena přes nějaký existující schvalovací systém. Do porovnávaných systémů jsem zahrnul ELO a Process Street.

2.5.1 ELO

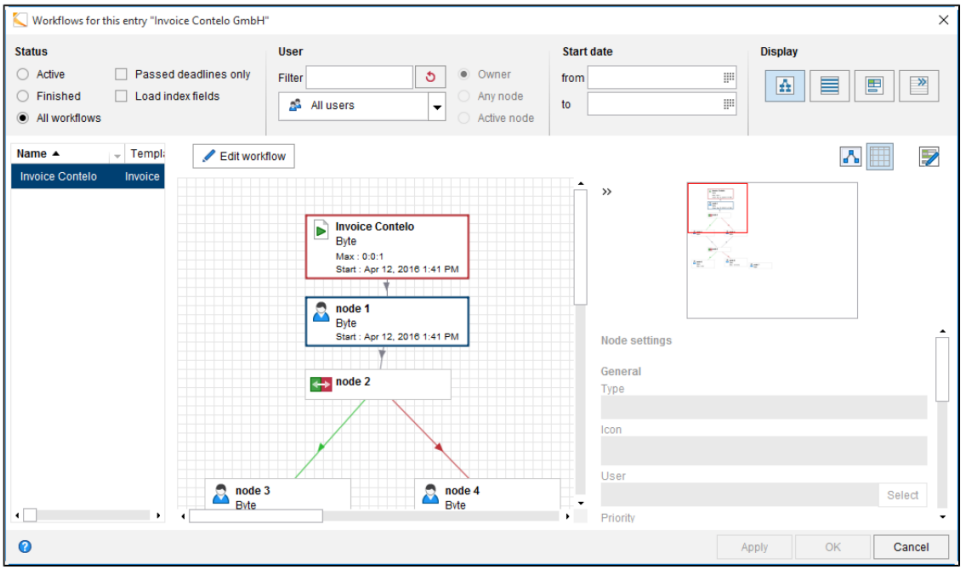
ELO je ECM (Enterprise Content Management) systém, tedy systém primárně určený pro správu digitálního obsahu. Obsahuje mimo jiné i propracovaný workflow systém a všemožné pluginy pro další použití, např. jako HR systém. S tímto softwarem mám osobní zkušenost, protože jsem se o něj staral v obecně prospěšné společnosti Člověk v tísni, kde ho využívali k ukládání všech dokumentů, jako jsou např. smlouvy a faktury, jejichž schvalování a manipulace probíhalo přes workflow. Správa tohoto systému, návrh a doprogramovávání chybějící funkcionality bylo mou úlohou.

V systému ELO běží každé worklow nad evidenční kartou, což je entita, která obaluje v systému soubor či složku a která k nim má přiřazený typ a metadata. Workflow mohou být dvojího typu – jednoduchý *ad hoc*, použitelný pro schválení nebo informování někoho, nebo workflow ze šablon. Tyto worklow ze šablon jsou určeny pro komplexnější procesy. Zmiňované šablony se tvoří přes Workflow Designer, který je vidět na obrázku 2.2. Ten obsahuje v oblasti 1 seznam všech šablon. V oblasti 2 je samotný návrh worklow diagramu. V oblasti 3 je přepínání nástrojů. V oblasti 4 je editor uzlů, ze kterých je worklow sestaven. A v oblasti 5 je status bar. Uzlů může být vícero typů, jsou startovací a ukončovací uzly, pak uživatelské uzly, ve kterých jde označit uživatele nebo celou skupinu uživatelů, kteří mají danou úlohu za úkol, uzly pro odesílání do vzdálených archivů, spouštění dalších workflow a podobně. Z těchto uzlů lze rovněž spouštět skripty, které pak mohou provádět např. manipulaci s metadaty, či editaci workflow uzlů pro aktuálně spuštěné workflow. Systém může nabízet jak čistě sekvenční, tak i paralelní worklow.

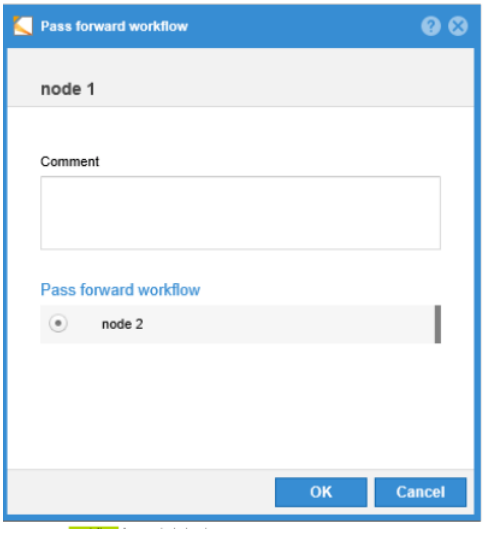
Tyto workflow se spouštějí buď ručně, nebo automaticky navázáním např. na vložení souboru daného typu. Při spuštění vzniká *de facto* kopie šablony, která je pak nadále samostatně měnitelná a do které se zapisují detaily o průběhu. To lze zobrazit, jak je patrné na obrázku 2.3. Jednotlivě uzly jsou



Obrázek 2.2: ELO Worflow Designer.



Obrázek 2.3: ELO Workflow záznam.



Obrázek 2.4: ELO Posun workflow.

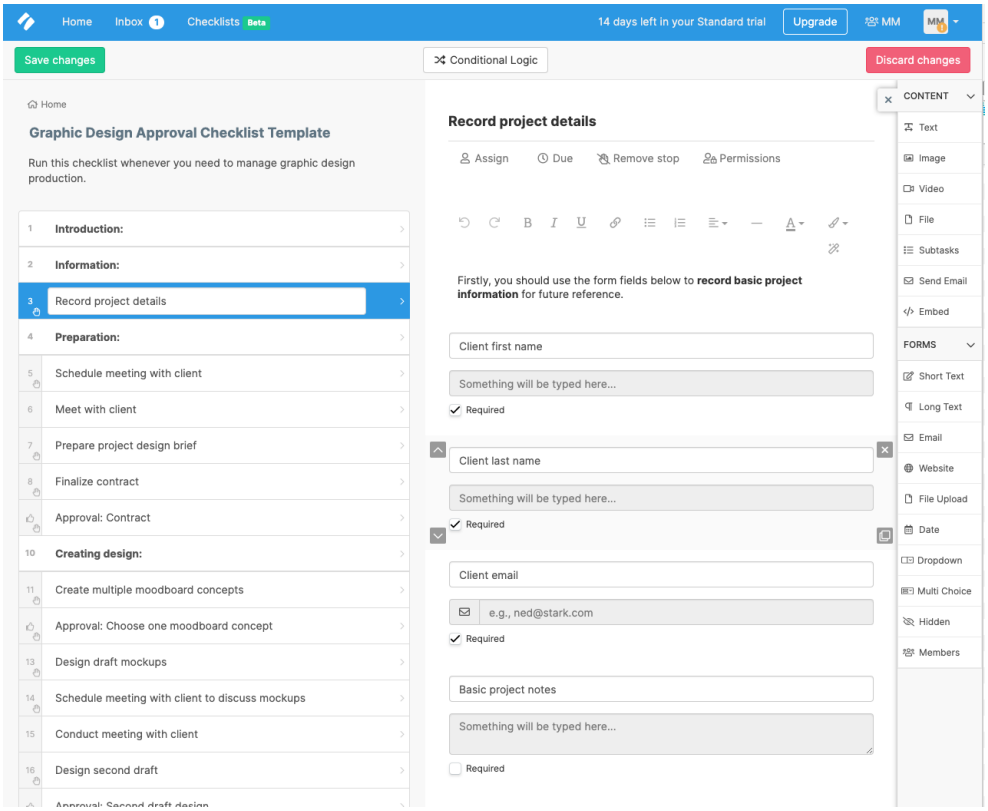
dle typu zpracovávány buď automaticky systémem, nebo je nutná interakce a provedení úkolu člověkem. Buď to dostane za úkol jeden konkrétní člověk, nebo celá skupina uživatelů, třeba jedno oddělení společnosti. Ti ho pak vidí mezi svými úkoly, a před stanovením samotného úkolu ho přiřadí sobě, aby se zamezilo několikanásobnému provádění téhož úkonu. Po provedení úkonu ho přesunou do dalšího uzlu/uzlů pomocí jednoduchého formuláře viditelného na obrázku 2.4. [4]

Worflow systém, který nabízí ELO, je značně bohatý a díky možnostem skriptů psaných v RhinoJS umožňuje velkou variabilitu, manipulaci s daty a podobně. Má v sobě implementovanou již i pokročilejší správu, zastupitelnost konkrétních lidí či předávání workflow dále jiným lidem. Nevýhodou ale je to, že je silně provázaný se zbytkem ELO softwaru, nad kterým je postaven. Jsou sice jisté možnosti integrace – buď externím API, nebo přes databázi. Pro využití GUI workflow systému by bylo ale nutné používat přímo vlastní nástroje ELO, jako je např. Java Client, jelikož integrace přímo do webové aplikace by nebyla možná. Je nutné podotknout, že GUI není úplně intuitivní, jak lze vidět třeba na obrázku 2.4 a pro jeho používání je nutné zaškolení. Dalším úskalím je absence podpory zobrazování a inspekce 3D objektů, které jsou nutnou náležitostí schvalování v projektu VMCK.

2.5.2 Process Street

Process Street je specializovaná online aplikace[1] pro jednoduchou správu procesů a workflow. Nabízí možnost jednoduše spravovat opakující se procesy v

[1] běžící na `https://www.process.st`



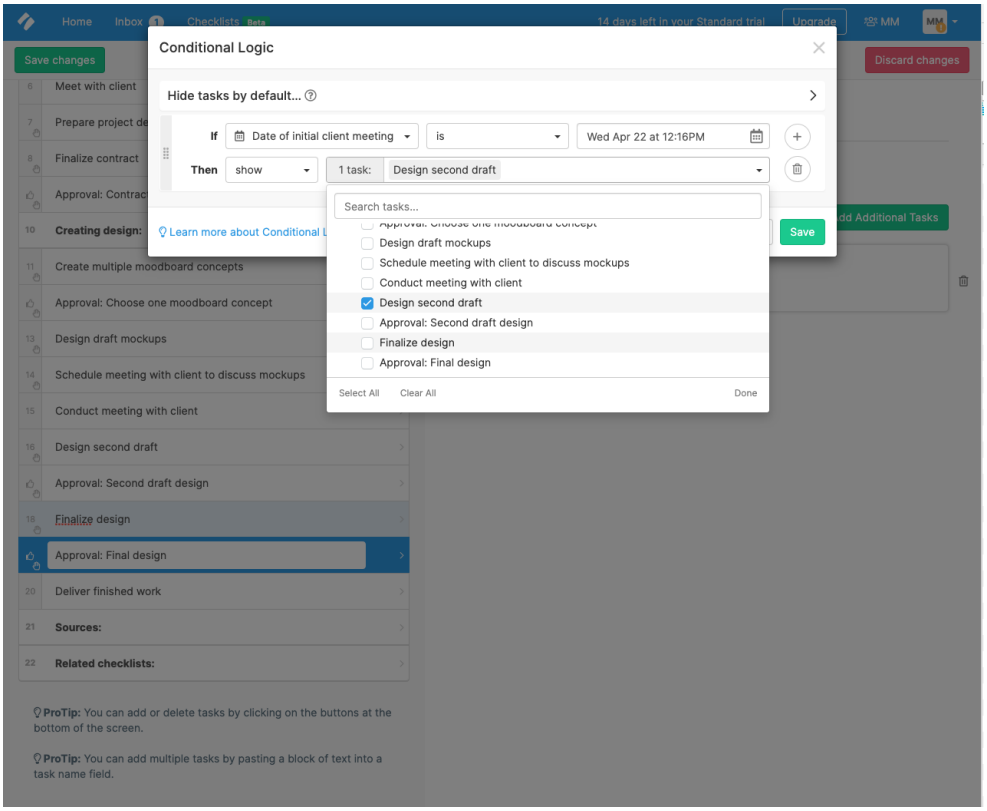
Obrázek 2.5: Process Street – editace formuláře kroků.

rámci týmu, vytvářet postupně dokumenty i je následně schvalovat.

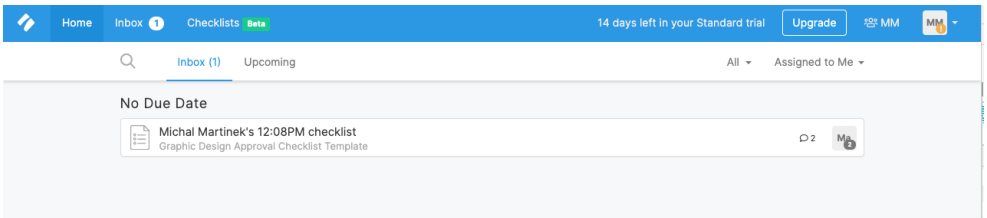
Základem v této aplikaci jsou šablony procesů. Tato šablona obsahuje sled kroků v daném procesu. Každý krok se může skládat z dalších podkroků, součástí tohoto kroku může být formulář, nahrání souboru, či schválení. K tomu je možno přidat libovolné popisky, obrázky či video. To, jak se jednotlivé kroky nastavují, je patrné na obrázku 2.5, kde lze zrovna vidět editaci formuláře v konkrétním kroku.

Připravené šablony se spouštějí – vzniká instance se svým vlastním během, ve které se ukládá postupné plnění jednotlivých kroků i s vyplněnými detaily. Jednotlivé kroky lze úkolovat na týmy či jednotlivé členy. Toto přiřazení uživatelů lze připravit i v šabloně. Ve výchozím nastavení se začíná od prvního kroku, po jeho splnění se postupuje na další krok. Lze nastavit paralelní souběh více kroků zároveň. Zobrazování a postup kroků lze nastavovat ještě složitěji, např. na základě vyplnění formuláře určitými hodnotami je možné zobrazovat určité kroky viz obrázek 2.6. [5]

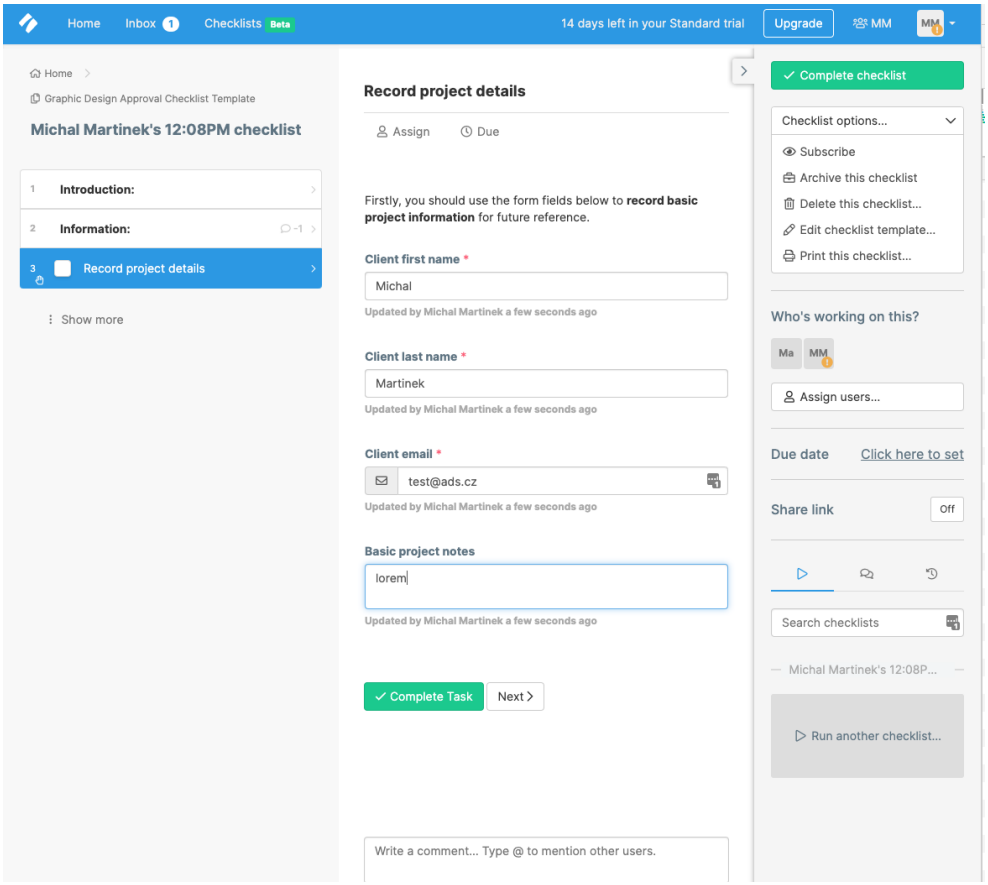
Uživatel ve svém účtu vidí seznam kroků, které se mají aktuálně zpracovat, jak je ukázáno na snímku 2.7. K těmto krokům lze přiřadit datum s časem, do kdy mají být hotové. Jedním klikem se pak může dostat na daný úkol



Obrázek 2.6: Process Street – nastavení toku kroků.



Obrázek 2.7: Process Street – seznam úkolů.

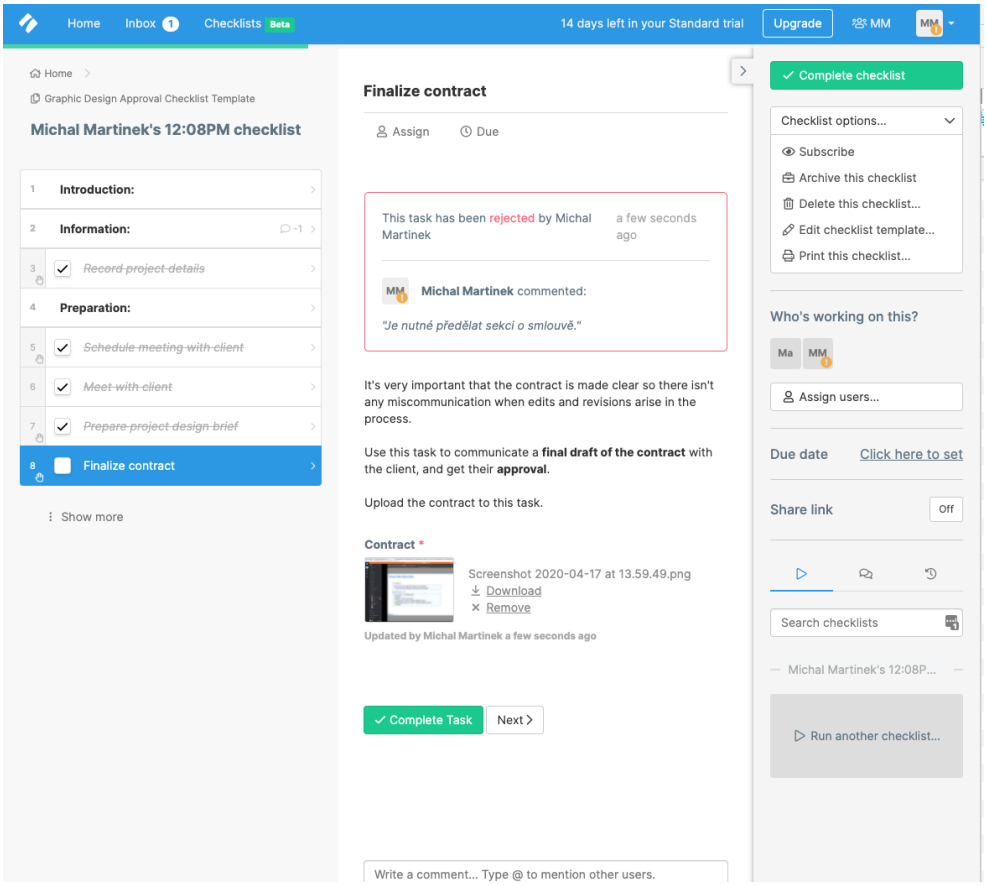


Obrázek 2.8: Process Street – vyplnění formuláře kroků.

(krok, který má za úkol) a může např. vyplnit formulář, jak je zobrazeno na obrázku 2.6, nebo schválit jiný krok, jak je vidět na obrázku 2.10. Pokud je krok zamítnut, viz obrázek 2.9, vrací se zpět i s poznámkou o jeho zamítnutí a je potřeba ho přepracovat a znovu potvrdit jeho provedení. Tímto způsobem je náležitě naimplementovaný schvalovací proces.

Proběhlé procesy pak lze spravovat, archivovat, tisknout a podobně. V nabídce je rovněž napojení na řadu dalších aplikací či export ve formátu CSV. Další praktickou věcí jsou stovky již předpřipravených šablon, které lze použít jako základ a doupravit pro konkrétní použití. [6]

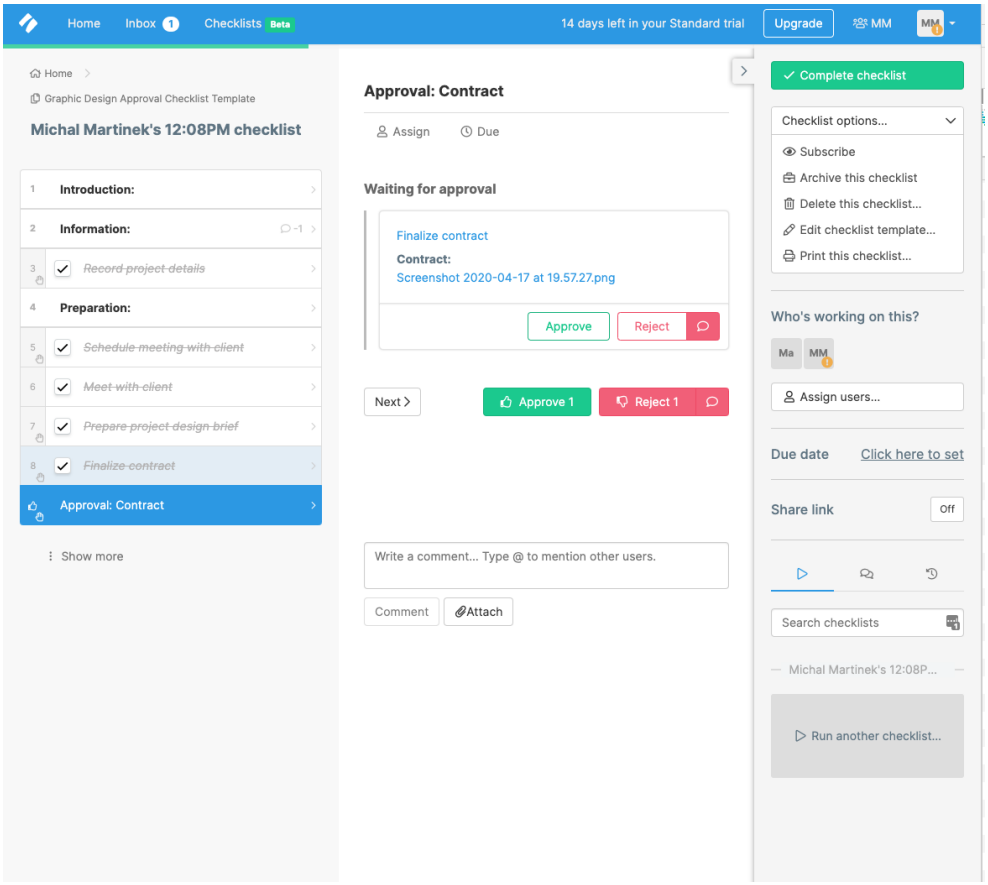
Aplikace Process Street je úzce specializovaná na správu procesů a workflow a tuto činnost dle mého mínění dělá velmi dobře. Pro jednodušší případy nebo skromnější použití nabízí vše nutné s uživatelsky příjemným a jasným UI. Problematická je větší customizace – v případě projektu VMCK je to chybějící podpora zobrazování 3D modelů, složitější schvalovací proces, jeho efektivita a napojení do vlastní aplikace.



Obrázek 2.9: Process Street – zamítnutý krok.

2.5.3 Srovnání

Systémy ELO a Proccess Street nabízejí oba velmi bohatý systém pro správu workflow, každý svým vlastním způsobem. ELO je zaměřené více na správu digitálního obsahu, což je bližší potřebám administračního rozhraní VMCK, ale kvůli problémům s integrací a velmi specifickým potřebám pro manipulaci s 3D modely je tento systém neuchopitelný. V těchto ohledech není vhodná ani aplikace Process Street, která je zaměřená na správu procesů a vytváření jednoduchých dokumentů, svým proměnlivým definováním workflow by ale rovněž schvalovací proces administračního systému byla schopna obsáhnout mimo kontrol 3D modelů.



Obrázek 2.10: Process Street – schvalování.

2.6 Analýza webových technologií a přístupů k vývoji frontendu

V diplomové práci se budu zabývat pouze frontendem této administrace, jelikož backendem se aktuálně zabývá Dan Vančura ve své bakalařské práci, přičemž samotný základ již postavil Jindřich Máca. Jako kontrakt pro komunikaci slouží definice v technologii OpenAPI v3 (nová verze specifikace pro Swagger), která slouží k popisu rozhraní API. K tomuto kontraktu se dostaneme později v kapitole 3.1. Tedy je nutné se zabývat pouze analýzou technologií použitých pro tvorbu frontendu.

Samotná administrace bude typu Single-page application (SPA), která funguje na tom principu, že po prvotním stáhnutí kostry stránky v HTML, příslušných kódů v JavaScriptu a CSS (Cascading Style Sheets) a dalších souborů se stránka již dynamicky přepisuje místo výchozího načítání celých nových stránek. Tento přístup funguje na základě přepisování DOM (Document Object Model) pomocí JavaScriptu. Ten taky obstarává komunikaci

s backendem pro stahování potřebných dat a případnou manipulaci s nimi. Tento přístup má ty výhody, že aplikace pak nabízí možnost lepší odezvy pro uživatele, chování blížící se nativním aplikacím a menší datový přenos.

Tento typ aplikací má ale rovněž několik nevýhod, většina z nich je spojena se SEO (Search Engine Optimalization), kvůli tomu, že jednotlivé stránky aplikace jsou složitěji indexovatelné pro vyhledávače, neboť kostra HTML neobsahuje při prvotním načtení žádné podstatné informace, do DOMu se standardně zapíše obsah až díky JavaScriptu. Toto se typicky řeší pomocí Server-Side Renderingu/Prerenderingu, kdy se provede JavaScript již na serveru, takže v poslaném HTML je již obsah a na tyto prvky v DOMU se pak naváže JavaScript aplikace. Indexace obsahu není pro naše administrativní rozhraní důležitá, protože veškerý obsah bude stejně zobrazen až po přihlášení. Tedy tato nevýhoda není pro naši administraci podstatná. Další nevýhodou SPA je velikost všech souborů, které se musí stáhnout při prvotním načtení. Toto se negativně projevuje i v SEO a tento obsah většinou bývá větší než načtení normální webové stránky, jelikož JavaScripty obsahují veškerou logiku a další zobrazitelné prvky pro aplikaci, které třeba nejsou ani aktuálně nutné. Toto se řeší např. postupným načítáním modulů s danou funkcionalitou na vyžádání, tzn. až je v aplikaci reálně potřeba. Jelikož se ale předpokládá, že uživatelé budou do aplikace chodit přes počítače, a nikoliv přes mobilní zařízení jako telefony a tablety, kde se dá očekávat lepší internetové připojení. A taky je zcela očekávatelné, že budou v rozhraní trávit více času, pak je trochu delší první načtení akceptovatelné, zvlášť když se zrychlí další práce s touto administrací.

Po srovnání výhod a nevýhod SPA lze vidět, že v případě tohoto administrativního rozhraní převážily výhody nad nevýhodami. Z obdobných důvodů se čím dál častěji vyvíjejí nejen podobné systémy jako aplikace typu SPA. Proto existuje i spousta knihoven a frameworků pro jejich tvorbu, na jejichž výběr se dále zaměřím.

Jak již bylo zmíněno, existuje řada knihoven, frameworků a rozdílných přístupů pro tvorbu webových aplikací, myšleno těch, které se starají o manipulaci s DOMem a samotné zobrazování dat. Mimo to existuje celá řada dalších, které se zabývají např. state managmentem, práci s 3D modely a podobně, jimiž se budu zabývat později. Do úvahy jsem vzal přístup frameworkless[7], tedy psaní *de facto* v čistém JavaScriptu s použitím malých knihoven a pak nejrozšířenější frontend frameworky – Angular, React, Vue.js a Ember.

2.6.1 Frameworkless

Frameworkless razí přístup vývoje webových aplikací bez použití frameworku. Je to z toho důvodu, že vývoj pomocí frameworku má signifikatní vliv na výsledný kód. Rozdíl mezi frameworkem a knihovnou je v tom, že knihovny zaobalují část funkcionality, kterou zprostředkovávají skrz funkce. Tyto funkce lze jednoduše používat v kódu a v naprosté většině případů nemá na jejich použití vliv použití dalších knihoven či frameworků. Takže kód, který využívá

dané knihovny, může vypadat prakticky jakkoliv a nahrazení jedné knihovny za jinou nebývá až tak problematické. Zatímco frameworky fungují tak, že je kód volán funkcionalitou frameworku. To ovlivňuje samotný kód a přepsání z jednoho frameworku do druhého často v podstatě znamená přepsání velké části aplikace. Rovněž některé části frameworku jsou zadrátované hluboko do aplikace, takže není např. pro daný kus funkcionality možné použít jiné řešení třeba v podobě jiné knihovny.[7]

I když se to nezdá, tak použití frameworku nebývá zadarmo. Pro řešení určitých problémů není nutné znovu implementovat stejné kusy funkcionality, nýbrž používá se k tomu cizí kód, který však nikdy nebude zcela optimální a optimalizovaný pro řešení konkrétních problémů v té konkrétní aplikaci. Každý framework obsahuje rovněž svůj technický dluh, který si částečně zavádíme dál do aplikací. Další cenou je závislost kódu na daném frameworku, a tedy možný zdroj budoucích problémů. Dá se očekávat, že obecně bude aplikace použitelná roky. V proměnlivém světě JavaScriptových technologií to ale nutně neznamená, že framework bude za roky ještě aktivně vyvíjen, nebo se aspoň budou opravovat chyby a bezpečnostní mezery. Ukázkovým příkladem je framework AngularJS z dílny Googlu, který vznikl v roce 2010. Jednu dobu byl prakticky standardem ve webových frameworcích, byl součástí např. MEAN stacku technologií. Google ale přišel v roce 2016 s novou verzí tohoto frameworku – Angular 2, který se v mnoha aspektech diametrálně liší. Jistou dobu probíhal paralelní vývoj obou verzí tohoto frameworku, ale v roce 2021 končí oficiální podpora původního AngularJS. Přejít na novější verzi Angularu je netriviální problém a prakticky znamená přepsat podstatnou část aplikace, což vyžaduje nemalé zdroje.

Proto je možné obejít se bez frameworků a vybudovat webovou aplikaci v čistém JavaScriptu či jiném jazyku, který se do JavaScriptu transpiluje, jako např. TypeScript, který se stává čím dál oblíbenějším. TypeScript přidává do světa JavaScriptu typování a typovou kontrolu v průběhu transpilace. K tomu lze samozřejmě použít i již existující knihovny. Podstatnou částí funkcionality, kterou nabízí frameworky typu Angular, Vue.js či React, je použití komponent jako stavebních bloků, na základě kterých je sestavena celá aplikace. Pro toto již ale existuje nativní řešení v podobě *Web components*. Ty mají již docela dobrou podporu v prohlížečích, ke dni 13. 3. 2020 tuto technologii podporuje 91,15% prohlížečů, s polyfilly ještě více. [8]

Web components se skládají ze tří hlavních technologií:

- **HTML templates** přináší podporu *template* tagu. Pomocí tohoto tagu lze zapisovat znovupoužitelné kusy HTML kódu, které ale nejsou přímo renderované, je však možné je použít v DOMu.

- **Custom elements** umožňují vytvářet vlastní DOM elementy s rozličnou funkcionalitou.

- **Shadow DOM** izoluje komponentu od vnějšího DOMu a zjednodušuje znovupoužitelnost.

I díky této technologii je možné vytvářet modulární aplikace nativně bez frameworků[9]. Na stránkách `https://www.webcomponents.org` je k dispozici celá řada již vytvořených komponent.

2.6.2 React

React je JavaScriptová knihovna pro deklarativní vytváření a používání komponent a jejich následnou manipulaci v DOMu. Na počátku byl vyvinut Facebookem pro interní účely, ale nakonec byl uvolněn v roce 2013 pod opensourcovou licencí a vznikla kolem něho velká komunita vývojářů a společností. Byl také inspirací pro řadu frameworků a knihoven jako např. Vue.js, Preact.[10]

React sice sám sebe představuje jako knihovnu, ale způsob psaní komponent a celých aplikací je natolik určující pro samotný zápis a fungování kódu, že je diskutabilní, jestli se ještě pořád dá považovat za knihovnu a skutečně bývá často zařazován jako framework. Proč tomu tak je, bylo bylo zmiňováno v sekci 2.6.1.

Základ Reactu tkví v komponentách, které zaobalují samostatnou ucelenou část aplikaci. Ty se pak skládají v celé stránky a aplikace. Komponenty si navzájem předávají data, jejichž změny fungují pouze jednosměrně (one-way binding). Tato data bývají většinou standardní JavaScriptové entity – typicky objekty. To umožňuje držet stav mimo DOM. Ke každému DOM elementu existuje v Reactu odpovídající VirtualDOM entita, která je reprezentací skutečného DOM elementu a která obstarává jednosměrný tok dat přes komponenty a aktualizaci skutečného DOM elementu pouze v případě, že se změnila data, která se vykreslují. To zaručuje minimální změny v DOMu a zabraňuje přerenderování celé stránky. Je to z toho důvodu, že zápisy do DOMu jsou relativně pomalé, a tímto se zrychluje aktualizace obsahu.

Komponenty se skládají ze standardních DOM elementů jako je např. *div, p, h1* atd. Ty jsou ale zaobaleny ve speciálních funkcích kvůli VirtualDOMu a podobně. Pro to, aby jejich použití bylo pro webový vývoj přirozené, vyvinul React tým syntax sugar v podobě zápisu JSX (JavaScript XML), což je rozšíření JavaScriptové notace o použití XML značek přímo v kódu, pro zaznačení výše zmíněných DOM elementů.[11]

Ekosystém okolo Reactu je dost poznamený funkcionálním paradigmatem a inspiruje se v jiných technologiích a jazycích jako Haskell, Elm či OCaml. Proto se v poslední době oprošťuje od psaní komponent jako statefull tříd a směřuje se k psaní čistých funkcí. Lze najít spoustu dalších příkladů – řada state managementů, které se vyvinuly původně v React komunitě, je inspirována myšlenkami funkcionálního programování – neměnitelnými datovými strukturami, čistými funkcemi, funkcemi výšších řádů.

2.6.3 Angular

Angular je open-source framework vyvíjený Googlem od roku 2010. Tento framework obsahuje celý ekosystém nástrojů a funkcionalit, pomocí kterých lze vyvíjet webové aplikace, bez nutnosti hledání dalších knihoven. Angular nabízí vytváření a manipulaci s komponentami včetně obousměrného datového navázání, routingu, funkcionality modulů (též jejich postupného zavádění (lazy loading)), Dependency Injection.

Mimo těchto funcionalit obsahuje nástroje pro urychlení vývoje. Jedním z nich je vlastní CLI (Command Line Interface), pomocí kterého lze generovat kostry jakýkoliv nových konstruktů jako např. komponenty, moduly, služby a podobně. Tento generátor je postaven na technologii zvané Schematics. Je to koncept generátoru kódu, jenž podporuje komplexní logiku pro složité transformace projektu přes virtuální souborový systém a ve kterém je možné pracovat i na úrovni AST (Abstract Syntax Tree). Je možné psát vlastní Schematics ke knihovnám, které je možné použít třeba pro automatickou migraci na novější verzi. Dalším častým příkladem použití je automatické zavedení knihovny do projektu nebo generování vlastních typů komponent.[12]

Zvláštně první verze frameworku – AngularJS byla dost oblíbená. Po vydání druhé verze, která nebyla zpětně kompatibilní a přinesla prakticky jiný framework, hodně vývojářů odešlo jinam, hlavně k Reactu s Fluxem či Reduxem. I přesto má širokou uživatelskou základnu a množství knihoven. Nabízí velmi dobrou podporu Material UI, Boostrapu či oficiální generátor kódu z Swaggeru. Je celý naspaný v TypeScriptu, který je výchozí volbou pro nové projekty, ale obsahuje podporu i pro čistý JavaScript či Dart, což je další projekt Googlu. V poslední době tento framework láká na optimalizovanou kompilaci a malý a efektivní výsledný kód. Další zajímavou věcí, kterou Angular nabízí, je vytváření Web Components pomocí Angular Elements.

Angular aplikace se skládá z modulů, které zaobalují část funkcionality. Tento modul je definován pomocí obsažených komponent, služeb či direktiv, importů dalších modulů. Rovněž lze nastavit, které části jsou pro interní použití a které věci z tohoto modulu budou přístupné dalším modulům. Pak můžeme do jednoho modulu naimportovat druhý, ze kterého lze používat exportované komponenty atd. Angular aplikace obsahuje minimálně jeden modul, a to kořenový.

Komponenty obsahují kostru psanou v šablonovacím systému Angularu a mimo to dodatečný kód v TypeScriptu. Ten slouží hlavně k předávání dat do šablony, přepočtům dat, zpracovávání interakce a podobně.

Mimo samotných kompent lze, jak již bylo naznačeno, definovat služby a direktivy. Služby jsou inspirované Command vzorem. Jsou užitečné třeba pro stahování či odesílání dat přes API, nebo jakékoliv jiné získávání či předávání dat. Angular se hodně opírá o knihovnu pro reaktivní programování Rx.js, která se používá dost často ve službách, např. i při použití Angular knihovních funkcí pro HTTP komunikaci. Služby slouží rovněž k odstínění komponent od

samotných modelů. Direktivy umožňují měnění vzhledu nebo chování jiných DOM elementů.

Každý objekt v Angularu se definuje jako třída, která se obaluje do dekorátů, ty pak určují o jaký typ objektu, třeba komponenta, se jedná a dodefinovává jeho části. V těchto třídách pak lze injektovat další části jako např. služby pomocí DI. To zmenšuje závislosti a jejich předávání. Rovněž kód je jednodušeji testovatelný.

V jistých aspektech se Angular podobá návrhovým vzorům jako MVVM či MVP ve vztahu zobrazovací vrstvy a šablony a odstínění od modelů. Mimo to má Angular velmi dobrou podporu pro reaktivní programování, ke kterému inklinuje i díky zabudované knihovně Rx.js.

2.6.4 Vue.js

Vue.js je další JavaScriptový framework pro tvorbu uživatelských rozhraní a webových aplikací. Vytvořil ho Evan You, který pracoval pro Google s Angularem, a rozhodl vzít z Angularu to, co se mu líbilo, a vytvořit vlastní odlehčený framework, který by umožňoval tvorbu komponent a jejich vzájemnou kompozici. S prvním vydáním přišel v roce 2014. Vue.js se samo označuje jako Progressive JavaScript Framework, což má znamenat, že je postaveno jako dodatečné značení ke HTML, které umožňuje dělat šablony k modelům, jež reagují na změny v modelu.[13]

Tento framework má za poslední roky velký nárůst oblíbenosti[14], a to i díky tomu, že je relativně jednoduchý na naučení. Samotný základ Vue.js obsahuje jen zobrazovací vrstvu, tedy technologie pro tvorbu, komunikaci a skládání komponent. Tento základ je poměrně malý, okolo 20 KB. Pokročilejší věci jako state management, routing, server-rendering či tooling pak řeší další oficiální samotné balíčky.[11]

Základ Vue.js je rovněž v komponentách. Přístup k nim je velice podobný jako v případě Reactu. Tyto komponenty se zapisují rovněž do jednoho souboru, a buď se může kostra komponenty psát do samostatného template elementu, nebo dokonce umí používat JSX notaci. Za Vue stojí rovněž Virtual-DOM, který se snaží o optimální překreslování a minimální změny v DOMu kvůli rychlosti.

2.6.5 Ember

Ember je webový framework orientující se na webové aplikace, který byl vydán již v roce 2011. Ember obsahuje zajímavý ekosystém, který se skládá z CLI, EmberData a samotného Ember.js. Stejně jako u Angularu platí, že pokud vyhovuje tento framework a jeho přístup k aplikaci, může být produktivita vývoje velmi vysoko, třeba kvůli generování kódu a předpřipravaných funkcionalitám.

Ember se nejvíce hodí pro aplikace ve kterých je tenký klient a server, na kterém je řešena většina logiky. Ember je postaven na MVVM architektuře.

Vše začíná u routeru, kde jsou zavedené jednotlivé cesty, podle kterých se zvolí příslušný pohled a kontroler. V cestě se předává model vykreslené šabloně, jelikož v Emberu má přístup k modelu pouze cesta a kontroler. Model obsahuje samotná data, kontroler slouží jako jeho rozšíření v podobě Singletonu a obsahuje aplikační logiku spjatou s modely. V šabloně se pak vykreslují data z modelu. Je silné oddělení zájmů mezi modelem a pohledem, jak tomu v MVVM architektuře bývá. Samotné modely jsou pak uloženy ve Storu, který obstarává i jejich stahování a aktualizaci přes API. Store slouží jako jeden zdroj pravdy a obsahuje veškeré modely, které jsou pak vkládány cestám, pakliže si o ně řeknou.[15]

2.6.6 Srovnání

Z popsaných přístupů k vývoji webových aplikací je nutné vybrat pouze jeden. Zde nastává ten problém, že jednotlivé přístupy nejsou zcela porovnavatelné, jelikož jde o různé kategorie technologií.

Pro samotné porovnání nakonec byla použita vícekriteriálni analýza variant, a to bodovací metoda s vahami. U každého kritéria budeme podle daných metrik bodovat každý přístup na stupnici od 1 do 10. Těmto metrikám následně přidělíme váhy a na základě toho získáme konečný počet bodů. Ne u každého kritéria lze získat data pro všechny varianty, proto porovnávání na konci rozdělíme na vyhodnocení těch kritérií, pro něž jsou dostupná data u všech variant a celkově, aby to bylo adekvátní a žádná varianta nebyla „penalizována“ za nedostatek dat.

2.6.6.1 Dokumentace

Důležitou součástí každého softwarového projektu je dokumentace. Ta podstatně ovlivňuje jak samotný proces učení, tak i dohledávání dalších užitečných informací. Hodnotil jsem u všech přístupů oficiální dokumentaci jako zdroj zaručených a aktuálních informací v různých kategoriích. Vyhodnocení tohoto kritéria je uvedeno v tabulce 2.2.

Frameworkless Přístup Frameworkless má ze své podstaty mnoho variant a kombinací, jak využít dostupné technologie a knihovny. Z tohoto důvodu neexistuje žádná komplexní dokumentace pokrývající toto téma. Je však o tom vydáno několik knih a rovněž k jednotlivým knihovnám a technologiím, jako např. Web Components, je k dispozici jednak bohatá literatura, jednak online zdroje. Získávání informací je kvůli roztříštěnosti složitější a náročnější. Samozřejmě je na každé považované téma hodnocení řada článků či knih, ale tak je tomu i u ostatních technologií, proto jsem se rozhodl považovanou do-

kumentaci omezit na komplexní knihu věnovanou tomuto tématu[16] a rovněž na MDN web docs[17].

React React nabízí jednodušší dokumentaci, která však pokrývá všechny aspekty této knihovny a je plná živých ukázek a odkazů na další zdroje. Existuje jistý prostor k vylepšení sekce přemýšlení v Reactu a osvědčené postupy. Obsahuje ale všechna důležitá témata mimo Server-Side Renderingu.

Angular Angular obsahuje komplexní dokumentaci popisující důkladně všechny jeho součástí. Zahrnuty jsou i návody pro korektní provedení, styleguide, bezpečnost či důkladně probraná technika testování, kterou má Angular implementovanou i díky vlastnímu kusu technologie na vysoké úrovni. Zdá se mi, že co se týká propojení frameworku s knihovnou Rx.js, mohla by na tom být dokumentace trochu lépe a zabývat se tím důkladněji.

Vue.js Vue.js nabízí velice pěknou dokumentaci, která pokrývá jak samotné základy, vývoj a osvědčené postupy, tak i složitejší aspekty tohoto frameworku, jako je bezpečnost, nasazování atd. Jediné, co chybí, je sekce o přístupnosti. Perfektně je popsaný i samotný tooling.

Ember Součástí Emberu je jasně strukturovaná dokumentace pokrývající všechny důležité součásti mimo pokročilejších technik jako Server-Side Rendering a ladění výkonnosti. Tutorial je velice obsáhlý a nabízí hluboký vhled do fungování tohoto frameworku. Komplexně probraná je rovněž sekce testování.

Tabulka 2.2: Porovnání dokumentace s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Tutorial	1	2	2	2	2
Základy	2	2	2	2	2
Testování	0	2	2	2	2
Přístupnost (A11y)	2	2	2	0	2
Server-Side Rendering	0	0	1	1	0
Výkonnost a nasazení	1	1	1	1	0
Úroveň dokumentace	6	9	10	8	8

2.6.6.2 Složitost a křivka učení

Dalším aspektem je složitost jednotlivých technologií. Ta určuje, jak dlouho se bude člověk tuto technologii učit a za jak dlouho se vývojář neznalý projektu zorientuje. Hodnocení složitosti a křivky učení vychází hlavně z článku *Angular vs React vs Ember vs Vue – JS framework comparison* [18] a dále pak

z článku *React vs Angular vs Ember vs Vue: Which Is the Best JavaScript Framework?*[19]. Výsledky těchto článků jsem validoval i se svým vlastním názorem, jelikož mám do jisté míry zkušenosti s každým tímto přístupem mimo framework Ember. Výsledek porovnání je v tabulce 2.3.

Frameworkless Přístup Frameworkless má ze své podstaty mnoho variant a kombinací, jak využít dostupné technologie a knihovny. Z tohoto důvodu neexistuje žádná komplexní dokumentace pokrývající toto téma. Je však o tom vydáno několik knih a k jednotlivým knihovnám a technologiím, jako např. Web Components, je k dispozici bohatá literatura i online zdroje. Získávání informací je kvůli roztříštěnosti složitější. Na druhou stranu člověk musí znát dobře jen čistý JavaScript, což by měla být nutná predispozice pro učení frameworků.

React React sám o sobě je *de facto* čistý JavaScript a zápis ve formátu JSX je relativně přirozený a snadný. Je to jen zobrazovací knihovna, tedy neobsahuje v sobě tolik různých funkcionalit, a proto je možné se ho naučit relativně rychle.

Angular Angular je komplexní framework s vlastní formou šablonovacího jazyka s určitými vlastními koncepty. Proto je jeho pochopení složitejší a je nutné se toho hodně naučit. O to více, pokud člověk není obeznámen s konceptem reaktivního programování a knihovny Rx.js.

Vue.js Vue.js je sice, co se týká obsažnosti, něco mezi knihovnou a frameworkem, ale je relativně jednoduchý na pochopení, díky čemuž se i proslavil a díky čemuž roste jeho obliba.

Ember Ember je jako Angular komplexní framework, takže je logicky nutné pochopit více věcí. Ale jelikož funguje na principu *convention over configuration*, může naučení všech konvencí zabrat velké množství času. I proto neexistuje až tolik lidí znalých Emberu do hloubky.

Tabulka 2.3: Porovnání křivky učení s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Křivka učení	5	7	4	6	2

2.6.6.3 Velikost komunity a používanost

Jedním z faktorů, které budu prozkoumávat, je i velikost komunity a používanost frameworků. Na tom záleží i to, kolik informací je dostupných, kolik nových

balíčků vzniká, jak moc je technologie otestovaná na reálných projektech či jestli v případě opuštění aktuálního spravovatele bude schopen projekt žít.

Toto vyhodnotím na základě několika metrik. Jednou z nich je počet uživatelů ze statistiky *State of JavaScript 2019* [14]. Ta poukazuje na počet lidí, kteří tuto technologii aspoň vyzkoušeli. Proto je tu i další metrika, která poukazuje na reálné řešení problému v té dané technologii, a to počet otázek na portálu StackOverflow (SO). Vyšší počet otázek sice může být způsoben stářím i nekvalitní dokumentací, to ale můžeme na základě předchozího bodu vyloučit. Taky můžeme předpokládat, že čím více otázek je, tím spíše nalezneme i tu řešící jeden konkrétní problém, proto toto budeme považovat jako sekundární faktor.

K přístupu Frameworkless nejsou dostupná žádná spolehlivá čísla, proto jej z bodování tohoto faktoru vyloučíme. Pro ostatní jsou data uvedená v tabulce 2.4.

Tabulka 2.4: Porovnání komunity s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Počet uživatel	-	16 551	11 582	9 320	4 532
Počet otázek na SO	-	195 454	206 610	51 800	23 407
Velikost komunity	-	10	9	6	2

2.6.6.4 Spokojenost uživatelů

Důležité mimo samotného počtu uživatelů je i skutečnost, jak moc jsou s danou technologií spokojení. Tato data jsem opět získal ze statistiky *State of JavaScript 2019* [14] z metrik udávajících, kolik uživatel by znovu použilo danou technologii. Bohužel nejsou žádná dostupná data pro přístup Frameworkless. Data jsou k nahlédnutí v tabulce 2.5.

Tabulka 2.5: Porovnání spokojenosti s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
% spokojených uživatel	-	87	38	87	37
Spokojenost	-	10	2	10	2

2.6.6.5 Oblíbenost

Mimo spokojenosti je další metrikou oblíbenost Git repozitářů na portálu Github, kde mají všechny frameworky vystaveny zdrojové kódy. Oblíbenost

budeme měřit pomocí počtu hvězdiček a sledujících[2], více v tabulce 2.6.

Tabulka 2.6: Porovnání oblíbenosti s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Počet hvězdiček	-	145 255	58 873	159 301	21 423
Počet sledujících	-	6 623	3 184	6 101	1 044
Oblíbenost	-	9	5	10	3

2.6.6.6 Počet knihoven

Další z faktorů, který ovlivňuje používání jedné z variant, je počet dostupných knihoven navržených pro spolupráci s daným frameworkem. Ve všech včetně přístupu frameworkless lze použít univerzální JavaScriptové knihovny, často bývá ale jednodušší zvolit upravenou knihovnu, jejíž použití zapadne více do daného frameworku. Počet těchto speciálních knihoven kromě přístupu Frameworkless jsem vyčetl[3] z oficiálního npm (Node Package Manager) registru, kde se nacházejí prakticky všechny veřejné balíčky pro Javascript. Tato data jsem pak ohodnotil v tabulce 2.7.

Tabulka 2.7: Porovnání počtu knihoven s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Počet npm knihoven	-	116 823	39 249	32 958	7 134
Počet knihoven	-	10	6	5	2

2.6.6.7 Výkonnost

Jedním z důležitých faktorů prakticky každé aplikace je rychlost načtení a rychlost při používání. Proto porovnám zabundlené velikosti frameworků a knihoven[19], stejně jako rychlost přerenderování[18]. Srovnání je uvedeno v tabulce 2.8.

2.6.6.8 Nástroje usnadňující vývoj

Jedná se o nástroje, které mají za cíl usnadnit a urychlit vývoj. Do této kategorie patří jak bohaté CLI (Command Line Interface), použitelné např. pro generování kódu, tak i podpora v IDE a editorech. Já se zaměřím pouze na nejpoužívanější nástroje v mé sociální skupině pro webový vývoj – VSCode,

[2]Ke dni 15. 3. 2020
[3]Ke dni 15. 3. 2020

Tabulka 2.8: Porovnání výkonnosti s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Velikost	0	100KB	500KB	80KB	111K
Přerenderování	dobré	výborné	dobré	výborné	pomalé
Výkonnost	8	9	4	10	4

Atom, Sublime Text 3 a WebStrom/IDEA Ultimate. Přístup Frameworkless využívá čistého JavaScriptu, a proto má nadprůměrnou podporu ve všech těchto i jiných IDE/editorech. Dalším aspektem je existence rozšíření do prohlížeče usnadňující proces prohlížení a debugování komponent. V nabídce jej mají všechny frameworky, jejich hodnocení na Chrome storu bylo tudíž vzato jako metrika. Sumarizace je k nahlédnutí v tabulce 2.9.

Tabulka 2.9: Porovnání nástrojů s údajem o bodování.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Podpora v IDE/editorech	4/4	4/4	4/4	4/4	4/4
CLI	-	ne	ano	ano	ano
Doplněk do prohlížeče	-	4,1/5	3,8/5	4,6/5	4,8/5
Nástroje usnadňující vývoj	6	5	6	7	8

2.6.7 Určení vah

Pro každou metriku je nutné určit váhu, jakou má pro konkrétní projekt. Z tohoto důvodu jsem se rozhodl použít Saatyho metodu pro určení těchto vah, přičemž jsem postupoval dle knihy *Manažerské rozhodování: postupy, metody a nástroje* [20]. Jelikož váhy mohou být velmi specifické pro konkrétní projekty a z velké míry záleží na preferencích vedoucích projektu, poprosil jsem o diskuzi kolegy – vedoucího projektu Ing. Jiřího Chludila a vedoucího mé diplomové práce Ing. Petra Pauše, Ph.D. V této diskuzi jsme se bavili o pořadí kritérií a jejich důležitosti. Určené priority pro dané metriky jsou zaznamenané v tabulce 2.10. Na základě této diskuze jsem pak následně vypracoval určení vah pomocí Saatyho metody, které následně bylo revidováno.

Saatyho metoda se skládá ze dvou kroků, prvním je zjištění preferenčních hlasů pro každou dvojici a na základě toho pak určení výsledných vah. U preference se určuje její velikost počtem bodů ze zvolené bodové stupnice. K ní se doporučuje i užití deskriptorů, které lze vidět v tabulce 2.11. Preferenční hlasy se zavádějí do matice velikosti preferencí (též Saatyho matice), která je dostupná v tabulce 2.12. Prvky s_{i_i} Saatyho matice jsou odhadem podílů (hledaných neznámých) vah kritérií v_i a v_j, takže platí $s_{i_j} = \frac{v_i}{v_j}$. Důležité je

Tabulka 2.10: Priority jednotlivých kritérií.

Metrika	Priorita
Dokumentace	1
Křivka učení	2
Velikost komunity	3
Spokojenost	4
Oblíbenost	5
Počet knihoven	6
Výkonnost	7
Nástroje usnadňující vývoj	8

Tabulka 2.11: Popis deskriptorů bodové stupnice preference.

Body	Deskriptor
1	Kritéria jsou stejně vyznamná
3	První kritérium je méně významné
5	První kritérium je dosti významné
7	První kritérium je prokazatelně významnější
9	První kritérium je absolutně významnější

vyplnit pravou horní trojúhelníkovou část matice, zbytek prvků je odvoditelný následovně:

- prvky na diagonále – $s_{i_i} = 1$ pro všechna i,
- prvky v levé dolní trojúhelníkové oblasti – $s_{j_i} = \frac{1}{s_{i_j}}$ pro všechna i a j.

Váhy kritérií lze na základě této matice určit exaktními nebo aproximativními způsoby. Já jsem se rozhodl pro aproximativní způsob, který poskytuje dobré odhady, a to geometrický průměr řádků. Saatyho matici a aproximaci vah jsem uvedl v tabulce 2.12. Přepočet vah na procenta jsem zanesl do tabulky 2.13.

2.6.7.1 Celkové shrnutí

Do srovnání nebyly zařazeny další faktory, které sice mají vliv na volbu technologie a frameworku, nejsou však důležité pro tento konkrétní projekt administračního rozhraní Věnných měst českých královen. Je to ku příkladu podpora PWA (Progressive Web Apps).

Jak bylo zmíněno na začátku, pro porovnání je použita vícekriteriální analýza variant, a to bodovací metoda s vahami. Na základě vah (viz 2.13) a předchozího ohodnocení jednotlivých metrik jsem vynásobil vahami bodový zisk z jednotlivých metrik. Přístup frameworkless neměl u poloviny metrik dostatečná data, proto je vyhodnocení rozděleno na dvě části. První součet

porovnává všechny tyto přístupy na podmnožině metrik s dostupnými daty (v tabulce označen jako *Součet kritérií označených **). Druhý součet všech kritérií je pak směrodatný pro porovnání variant kromě přístupu frameworkless. Data jsou zanesena v tabulce 2.14.

Z výsledků vícekriteriální analýzy variant vychází po srovnání všech přístupů v metrikách, v nichž byla data pro všechny varianty, nejlépe React s bodovým ziskem 4,54. Těsně za ním následuje Vue.js se 4,14 bodu. Dalším v pořadí je Angular se 3,93 bodu a následně přístup Frameworkless se 3,27 bodu. Poslední skončil framework Ember s 2,99 bodu. Dle těchto výsledků vychází, že podle stanovených kritérií bychom měli vybírat optimálně z prvních dvou – React a Vue.js, jejichž výsledek je relativně srovnatelný. Tento výsledek koresponduje s tím, že administrační rozhraní by mělo být napsané relativně čitelně, v technologii, do které není složité proniknout, jelikož se předpokládá další rozvoj tohoto systému v budoucnu. Přístup Frameworkless narazil na přílišnou variabilitu a nutnost hlubších znalostí. Uchopení projektu v jednom ze zmíněných kandidátů je podle mé osobní zkušenosti daleko jednodušší než pochopení vlastního řešení.

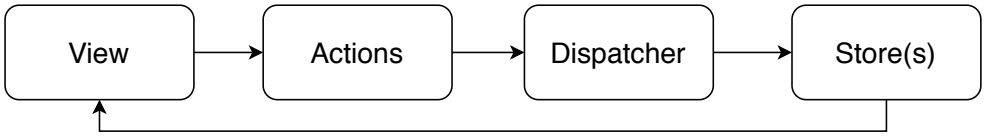
Pro výběr z kandidátů – React a Vue.js použijeme srovnání ve všech kritériích, která byla dostupná pro všechny srovnávané frameworky. V tomto celkovém hodnocení dopadl nejlépe React s 8,80 bodu, Vue nakonec skončil s 7,49 bodu. Pro jistotu posoudíme i ostatní přístupy, další v pořadí skončil Angular 6,11 bodu a poslední Ember s 3,90 bodu. Je tedy jisté, že budeme volit z předvybraných kandidátů z prvního srovnání. Na základě celkových výsledků vychází nejlépe React, proto bude použit pro implementaci administračního rozhraní VMCK. Další drobnou výhodou je, že s ním mám praktické zkušenosti mimo jiné i z bakalařské práce.

2.7 Analýza podpůrných knihoven

Z předchozí analýzy je jasné, že se vývoj bude odehrávat v Reactu. Jak už ale bylo zmíněno, React se zabývá primárně zobrazovací částí. Obsahuje vnitřní mechanismy pro efektivní předávání dat, ale u větší aplikací již tento přístup bývá neudržitelný. Podobným nedostatkem trpí rovněž i Vue.js a Angular. Proto vznikly knihovny pro state managment, které řeší problém úložiště stavu, manipulaci s daty a rovněž i efektivní napojení na komponentový systém každého z těchto frameworků pro minimální překreslování. Dalším problém, který bude nutné řešit, je zobrazování 3D modelů v prohlížeči. I pro to již existují podpůrné knihovny. V této části se budu věnovat jejich analýze a výběru.

2.7.1 State management

Problém state managmentu je obecný a řeší se i v přístupu frameworkless. I proto platí, že jakoukoliv z níže projednávaných knihoven je možné použít i v



Obrázek 2.11: Flux vzor.

tomto přístupu a rovněž ve všech diskutovaných frameworcích. Z dostupných možností jsem vybral ty nejpoužívanější, a to z důvodů podpory, dohledatelných dalších informací a otestovanosti v praxi. Jsou to knihovny Flux, Redux a MobX.

2.7.1.1 Flux

S tímto způsobem správy stavu přišel tým Facebook jako s řešením vyvinutým pro interní účely po 2 letech trápení s MVC pro použití s Reactem, který je rovněž z jejich dílny. Jedná se o návrhový vzor zakládající se na jednosměrném toku dat a podporující skládatelnost pohledů z React komponent.[21]

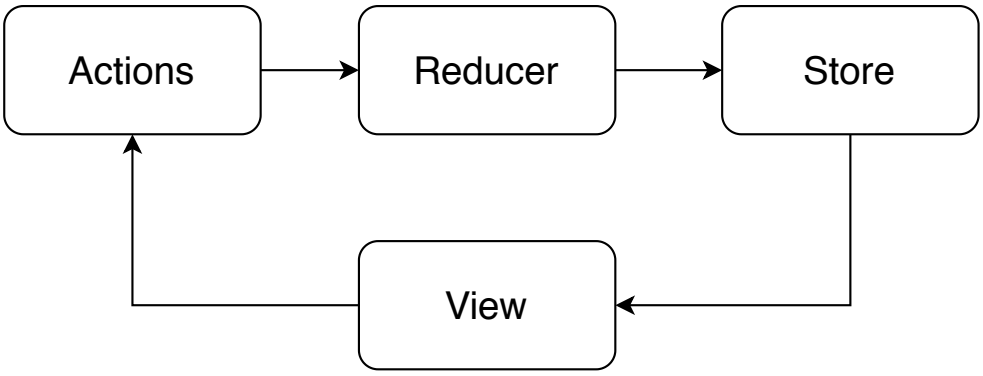
Na obrázku 2.11 převzatého lze vidět fungování tohoto vzoru. Na počátku stojí samotná zobrazovací vrstva např. React komponenta. Interakce s uživatelem a jiné akce jsou převedeny pomocí jednoduchých funkcí action creatoru do typu action, což je jednoduchý JavaScript objekt obsahující označení typu akce a případně přibalená data (payload). Tato akce je převedena do dispatcheru, který obstarává koordinaci akcí a předávání do úložišť (stores) a akci pošle všem úložištím. Tento dispatcher je pouze jeden - dle vzoru singleton. Úložištěm je pak kontejner pro data s logikou. Reaguje na příchozí akce a mění případně data. Změny v datech pak tečou zpět do zobrazovací vrstvy.[22]

Zveřejnění tohoto navrhového vzoru a knihovny Flux v roce 2014 podnítilo vznik dalších variací na tento vzor, jedna z nich je i knihovna Redux.

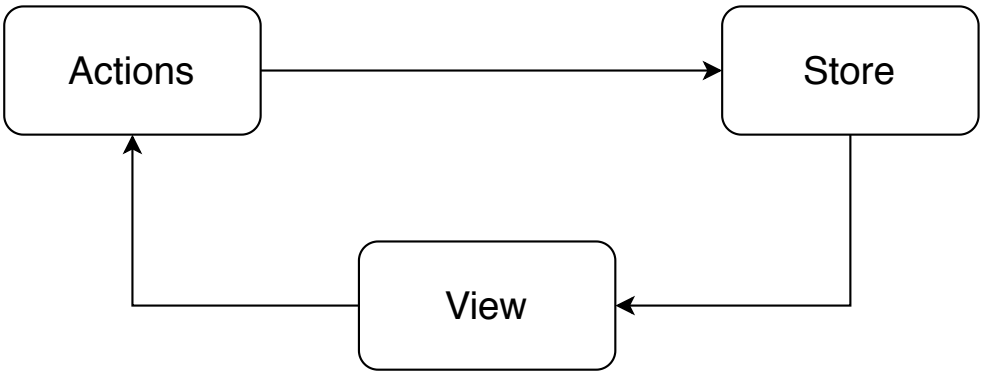
2.7.1.2 Redux

Tuto knihovnu inspirovanou Fluxem vyvinul Dan Abramov, jelikož si myslel, že by state management mohl fungovat jednodušeji. Na obrázku 2.12 je znázorněno jeho fungování. Na rozdíl od Fluxu má Redux pouze jeden centrální store. Podstatná změna je rovněž v tom, že neexistuje dispatcher, přes který by se akce dostávaly do storu, existují však reducery, které jsou čisté funkce, a ty na základě akce a starého neměnitelného stavu vytvoří nový neměnitelný stav. Tento nový stav se pak propíše do úložiště (store) a zobrazovací vrstvě se pošle informace o změnách.

Samotný Redux je nezávislý na použitém frameworku atd. Originálně vznikl stejně jako Flux pro React, který stojí rovněž na principech funkcionálního programování a se kterým skvěle funguje. Na jeho základě vznikly



Obrázek 2.12: Redux vzor.



Obrázek 2.13: MobX vzor.

další odnože, které mají za cíl zakomponovat tento state managment do stylu daných frameworků, je to např. Vuex pro Vue.js či NgRx pro Angular.[23]

2.7.1.3 MobX

MobX je další knihovna, která řeší state managment. Na obrázku 2.13 je znázorněno její fungování. Ta na rozdíl od Reduxu může obsahovat více úložišť (store), jejichž data jsou měnitelná. Pro samotné ukládání dat se používají observables, díky čemuž je možné se napojit a pak reagovat na změny v datové struktuře. I díky tomu je možné do MobX ukládat denormalizovaná data, která automaticky reagují na změny prostřednictvím observables. Taky pro to je možné napojit komponenty jednoduše na tyto data a propisovat do nich efektivně nová data. Do storu se data dostávají rovněž pomocí akcí, což je ale ve světě MobX jakýkoliv kód, který mění store, a není přesně dáno, jakým způsobem to má činit – lze jej měnit napřímo.[24]

2.7.1.4 Vybrání knihovny

Na základě analýzy i osobní zkušenosti se všemi třemi knihovnami jsem toho názoru, že žádná z nich není špatná volba. MobX je orientovaný spíše na menší aplikace, zatímco Redux s Fluxem spíše na větší. Implementace v Reduxu či Fluxu je lépe udržitelná než v MobX, ale na druhou stranu je toho kódu o dost více, jelikož MobX řeší část věcí skrytě pomocí dekorátorů. Redux a Flux je v tomto směru dost přímočarý a je jasné, co se děje. Redux je založen na funkcionálních principech stejně jako React, díky čemuž je jeho chování více predikovatelné a jednodušeji se debuguje. Bohužel kvůli tomu je však složitější se jej naučit. Dalším faktorem je používanost, ta ovlivňuje množství dostupných materiálů, znalost mezi lidmi či otestovanost v praxi. Do tabulky 2.15 jsem proto zanesl data o používanosti těchto knihoven ze statistiky *State of JavaScript 2019* [14], ze které jasně vyplývá, že Redux je nejpoužívanější z těchto knihoven, i když MobX v posledních letech docela vyrostl.

Z kvůli široké používanosti, robustnosti a čistotě řešení a skvělé dokumentaci jsem se rozhodl použít pro projekt administračního rozhraní VMCK knihovnu Redux. Dalším faktorem je i má osobní zkušenost, Redux je mi koncepčně nejbližší a jsem s ním nejvíce obeznámen.

2.7.2 Práce s 3D modely

Cílem projektu Věnná města českých královen je tvorba 3D modelů. Pro jejich inspekci a schvalovací proces je nutné je zobrazovat v prohlížeči a rovněž nabídnout sadu nástrojů napomáhajících zkoumání modelu. Pro tyto účely se mi zdá adekvátní použít knihovnu, která by ulehčovala práci s 3D modely. Přirozeně se proto nabízí použití nativní technologie WebGL. Dalšími možnostmi, které jsem vybral k analýze, jsou knihovny Tree.js a A-Frame, jehož použití a možnosti již byly na projektu VMCK zkoumány v bakalářské práci Tomáše Biláka[25].

2.7.2.1 WebGL

WebGL je webový standard pro manipulaci s 3D grafikou, vystavený jako JavaScriptové API. Je relativně nízkoúrovňový a vychází z OpenGL. Pro svůj běh využívá GPU (Graphics processing unit) a vykresluje se pomocí canvas elementu.

Podpora WebGL je vestavěná v prohlížečích a zástupci většiny moderních desktopových prohlížečů jako Chrome, Firefox či Safari jsou dokonce ve WebGL Working Group.[26]

2.7.2.2 Three.js

Three.js je knihovna, která obaluje WebGL a canvas pro práci s 3D modely. Má za cíl zjednodušit práci a nabídnout vývojářům vysokoúrovňový přístup.

Obsahuje širokou nabídku efektů, animací, manipulaci se scénami během runtimu, načítání dat z různých formátů či světel a pohledů (kamer). Má širokou komunitu a v dokumentaci přes 150 ukázek použití.[27]

2.7.2.3 A-Frame

A-Frame je knihovna, která vznikla původně v Mozille, za účelem tvorby VR pro webové prostředí jednoduchým způsobem. Interně používá knihovnu Tree.js. Nabízí práci deklarativním způsobem pomocí ECS architektury, která funguje na základě komponent. Její zaměření je primárně na tvorbu VR a AR.[28]

2.7.2.4 Volba knihovny

Z těchto možností se mi pro použití na projektu administračního rozhraní VMCK jeví jako nejvhodnější knihovna Three.js. Řeší problémy načítání 3D modelů z různých datových formátů jako FBX, Blender či OBJ. Nabízí efektivní práci s 3D objekty a jejich manipulaci a má bohatou dokumentaci a další zdroje.

Tabulka 2.12: Saatyho matice.

Kritérium	Doku.	Křivka uč.	Spokoj.	Knihovny	Komunita	Oblíbenost	Výkonnost	Nástroje	Geom. prům.
Doku.	1	1	3	3	3	5	5	5	2,76
Křivka uč.	1	1	3	1	3	5	5	5	2,40
Spokoj.	$0,\overline{3}$	$0,\overline{3}$	1	3	3	5	3	5	1,71
Knihovny	$0,\overline{3}$	1	$0,\overline{3}$	1	1	3	3	3	1,15
Komunita	$0,\overline{3}$	$0,\overline{3}$	$0,\overline{3}$	1	1	3	5	3	1,07
Oblíbenost	0,2	0,2	0,2	$0,\overline{3}$	$0,\overline{3}$	1	3	3	0,55
Výkonnost	0,2	0,2	$0,\overline{3}$	$0,\overline{3}$	0,2	$0,\overline{3}$	1	5	0,44
Nástroje	0,2	0,2	0,2	$0,\overline{3}$	$0,\overline{3}$	$0,\overline{3}$	0,2	1	0,30

Tabulka 2.13: Váhy jednotlivých kritérií.

Metrika	Váha (v %)
Dokumentace	26,59
Křivka učení	23,18
Velikost komunity	10,27
Spokojenost	16,52
Oblíbenost	5,27
Počet knihoven	11,05
Výkonnost	4,27
Nástroje usnadňující vývoj	2,85

Tabulka 2.14: Výsledné srovnání přístupů.

Metrika	Varianty				
	Frameworkless	React	Angular	Vue.js	Ember
Dokumentace *	1,60	2,39	2,66	2,13	2,13
Křivka učení *	1,16	1,62	0,93	1,39	0,46
Velikost komunity	-	1,03	0,92	0,62	0,21
Spokojenost	-	1,65	0,33	1,65	0,33
Oblíbenost	-	0,47	0,26	0,53	0,16
Počet knihoven	-	1,10	0,66	0,55	0,22
Výkonnost *	0,34	0,38	0,17	0,43	0,17
Nástroje usnadňující vývoj *	0,17	0,14	0,17	0,20	0,23
Součet kritérií označených *	3,27	4,54	3,93	4,14	2,99
Součet všech kritérií	3,27	8,80	6,11	7,49	3,90

Tabulka 2.15: Používanost state managment knihoven.

Knihovna	Počet používajících uživatel
Flux	17
Redux	13 441
MobX	2 378

Kapitola 3

Návrh

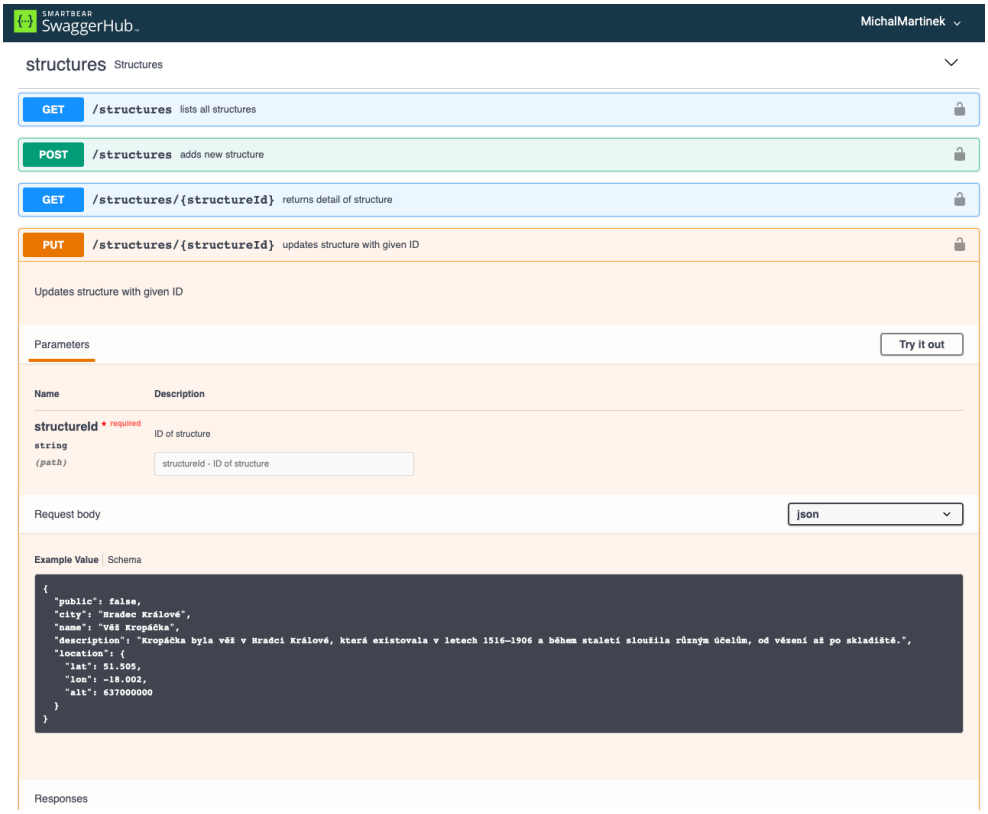
3.1 Privátní API

Jak již bylo zmíněno na samém začátku, v kapitole 2.1, tak administrační rozhraní se opírá o privátní API, které se stará o veškeré ukládání dat, přístupová oprávnění, případnou business logiku atd. Toto API ale v době psaní této práce bylo rozdělané a nebylo jasné ani samotné rozhraní. Podstatný kus funkcionality byl již vytvořen Jindřichem Mácou, který naimplementoval nejpodstatnější část, a to ukládání 3D modelů a spjatých souborů jako textur, včetně jejich verzování. Velkou část ale bylo potřeba dodělat – samotná správou entit *structure*, schvalovací systém, správu uživatelů a další drobnosti.

Na dokončení této části funkcionality se již nějaký čas pracovalo a diskutovalo se o ní. Navrhnout dobře API není triviální úkol a je ideální, když se na tom podílí lidé ze serverové (backend) i klientské (frontend) strany, aby bylo navrženo optimální rozhraní. Taková je i moje zkušenost, co se týká ideálního vývoje, kterou jsem nabyl při práci pro Komerční banku. Z těchto důvodů jsem byl přizván do diskuzí o tomto návrhu s vedoucím projektu za fakultu Ing. Jiřím Chludilem, Danem Vančurou, vypracovávajícím bakalářskou práci, která se zabývá právě vývojem tohoto API, a Martinem Půčalou, tvořícím bakalářskou práci na téma mobilních aplikací. Cílem bylo navrhnout společný kontrakt – rozhraní, v tomto projektu ve formě OpenAPI 3 definice, jelikož již hotová část od Jindřicha Mácy byla v tomto formátu.

OpenAPI 3 je specifikace pro dokumentaci rozhraní API. V této specifikaci se definují jednotlivé endpointy, u kterých se uvádí popis, formát data na vstupu a výstupu, možné stavové kódy a podobně. Tato definice se ukládá ve strukturovaném formátu YAML (YAML Ain't Markup Language) nebo JSON, díky čemuž je strojově čitelná. Kolem OpenAPI je vybudovaná sada nástrojů Swagger, pomocí které je možné tuto definici jednoduše vytvářet, generovat dokumentaci pro toto API, posílat testovací dotazy či např. generovat kód obsluhující toto API jak z klientské, tak i serverové strany.[29]

V rámci zmiňovaných diskuzí a po několika iteracích jsem doplnil původní



Obrázek 3.1: Dokumentace API.

definici o tyto části:

- Entita *structure* s metadaty, vazbami na entity *3D object* s endpointy, pro jejich získávání, vytváření, editování a mazání – CRUD (Create, read, update, delete).
- Doplnění entity *3D object* o další nutná pole – dynamické štítkování, 3D transformace atd.
- Doplnění endpointů pro manipulaci se samotnými 3D modely, texturami a dalšími soubory.

Tato varianta privátního API není určitě definitivní, ale je dostačující pro nynější fungování a bude v budoucnu rozšiřována dle aktuálních potřeb. Původní verze návrhu API je k dispozici na přiloženém médiu v souboru *api/before.yaml* a na adrese[4], aktualizovaná verze je rovněž přiložena na médiu v souboru *api/after.yaml* a je k nahlédnutí na adrese[5]. Pro ukázku, jak vlastně

[4] https://app.swaggerhub.com/apis-docs/VMCK/private-api/3.0.0
[5] https://app.swaggerhub.com/apis-docs/mmlab7/draftVMCK/3.0.0

vypadá dokumentace vygenerovaná z této definice, jsem přidal snímek kusu této dokumentace, a to na obrázku 3.1.

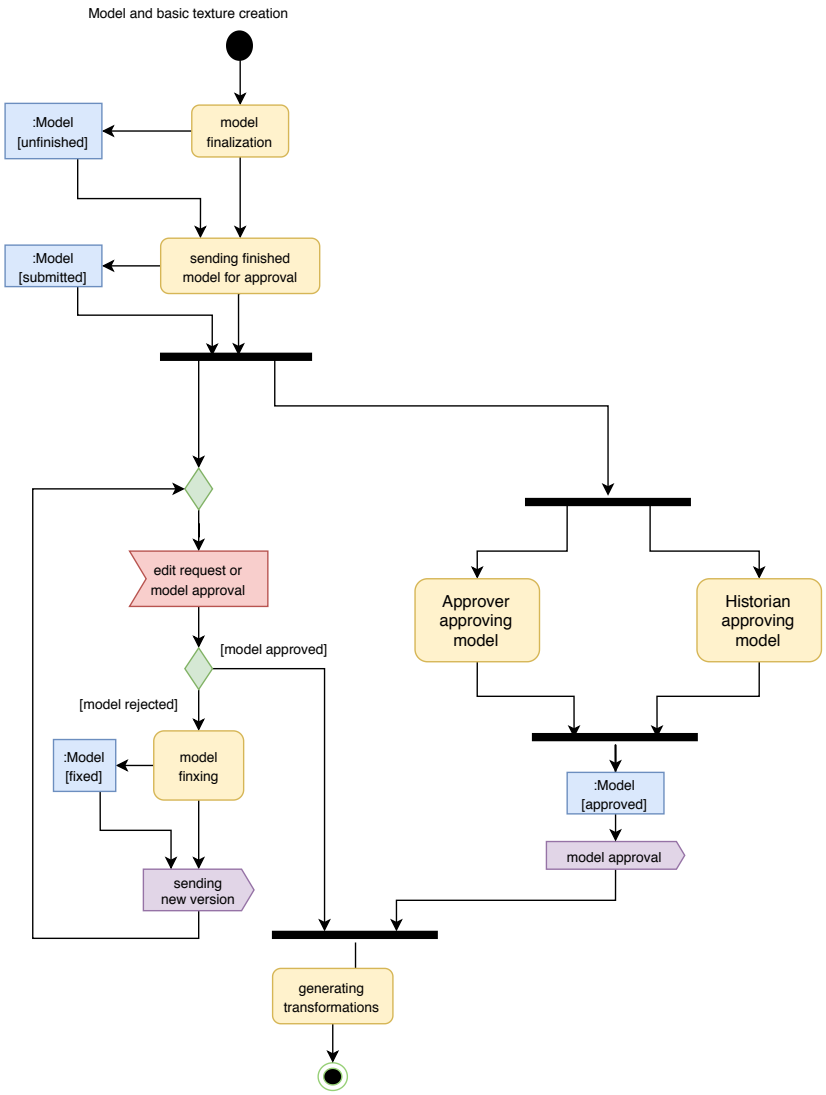
3.2 Schvalovací proces

Schvalovací proces tvoří podstatnou část administračního rozhraní a je to důvod, kvůli němuž bude do systému chodit podstatná část uživatelů. Schvalovat se musí každý 3D model předtím, než může jít do produkce a zobrazovat se uživatelům.

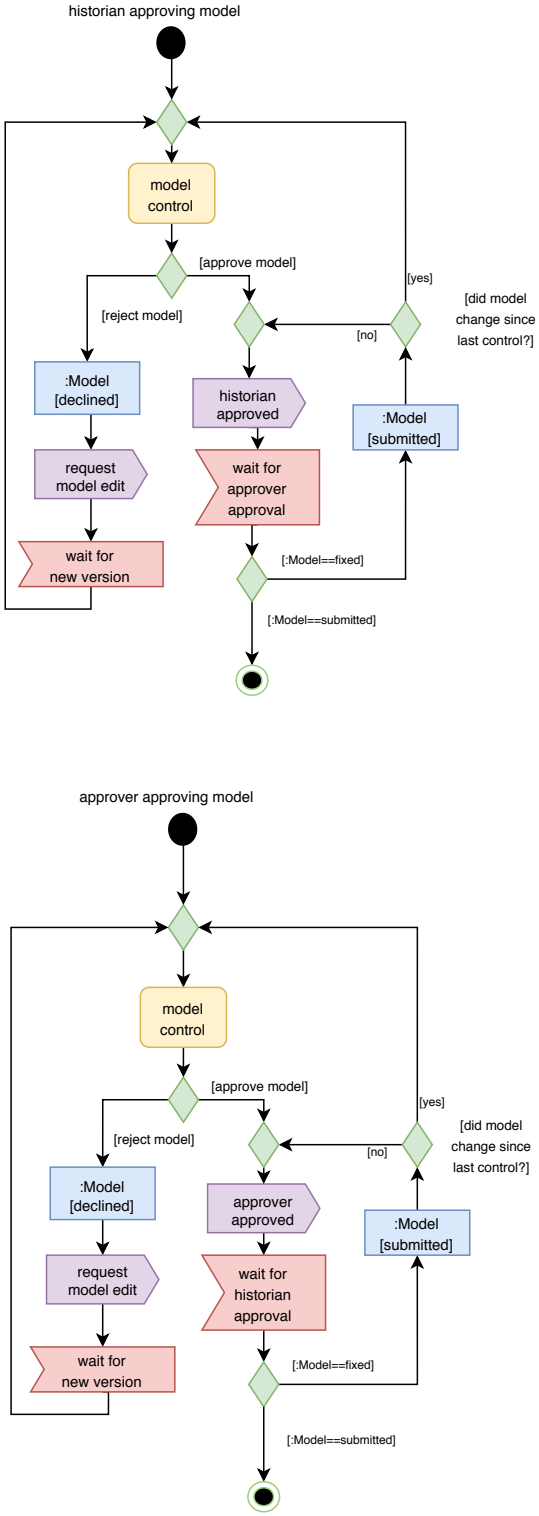
Na schvalovacím procesu jsem spolupracoval s Danielem Vančurou a diskutovali jsme to i s Ing. Jiřím Chludilem, který projekt zastřešuje za naši fakultu. Samotné schvalování lze rozdělit na několik částí. Nejdříve *Historik* vytvoří entitu *structure* se všemi nutnými informacemi pro vymodelování modelů. Přidá různá metadata, např. kde se artefakt nachází, dodá všechny nutné relavantní podklady pro modelování. Jakmile je se zadáním spokojen, spustí proces modelování a schvalování. K tomuto procesu se připojí *Grafik*, který má za úkol zkontrolovat model po stránce grafické správnosti, a *Modelář*, který má za úkol model dle zadání připravit. Po tomto přiřazení se spouští proces modelování základní verze modelu, tzn. čistý model dle zadání, který lze vidět na diagramu 3.2 a 3.3. Diagramy pro tuto část schvalovacího procesu vytvářel kolega Daniel Vančura v rámci své bakalářské práce.

Modelář vytváří model a může do systému ukládat průběžné verze. Když je s modelem spokojený, odešle ho ke schválení, v tuto chvíli se dostane k oběma schvalovatelům – *Historikovi*, který kontroluje, zda je model historicky věrný, a *Grafikovi*, který kontroluje technickou správnost. V tento okamžik se mohou oba dva k modelu vyjádřit a označit ho jako schválený, nebo ho vrátit s připomínkami. Pokud ho oba dva rovnou schválí, tak se přesune do produkční databáze, a spustí se další fáze, o které bude řeč později. Pokud ho oba dva neschválí, dostanou oba poté, co *Modelář* nahraje opravenou verzi a odešle ji znovu ke schválení, upomínku k opětovnému schválení. Nastane-li situace, kdy je jeden s výsledkem spokojen a druhý ne, tak ten, co je spokojen, nemusí provést opětovné schválení, a čeká se, až se spraví daná chyba mezi tím, co je nespokojen, a *Modelářem*, a až po opravě daných chyb je ten druhý vyzván k finálnímu schvalování. Toto má za úkol redukovat pracnost schvalovacího procesu.

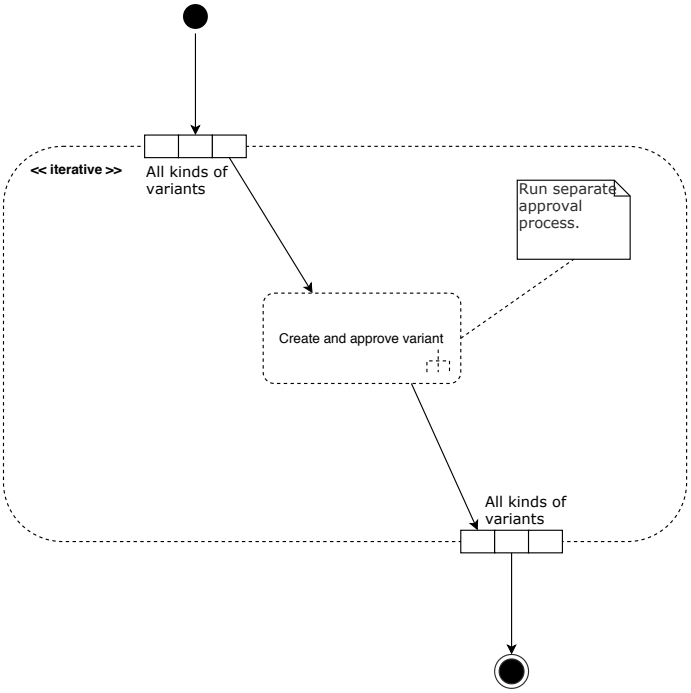
Po schválení tohoto základního modelu je tento model zveřejněn a spouští se druhá fáze, a to vytváření jednotlivých variant. Samotné spouštění lze vidět na diagramu na obrázku 3.4, schvalovací proces pro jednotlivou variantu pak na diagramu 3.5. Diagramy pro tuto část schvalovacího procesu jsem vytvářel já. Jak již bylo psáno, tak pro maximální věrnost a přesnost se budou vytvářet jednotlivé varianty pro různé typy počasí, zestárnutí modelu v průběhu času a podobně. Tvorba těchto variant bude poloautomatická pomocí tranformačního systému využívajícího vyhotovené pluginy do Blenderu,



Obrázek 3.2: Základní schvalovací proces I.

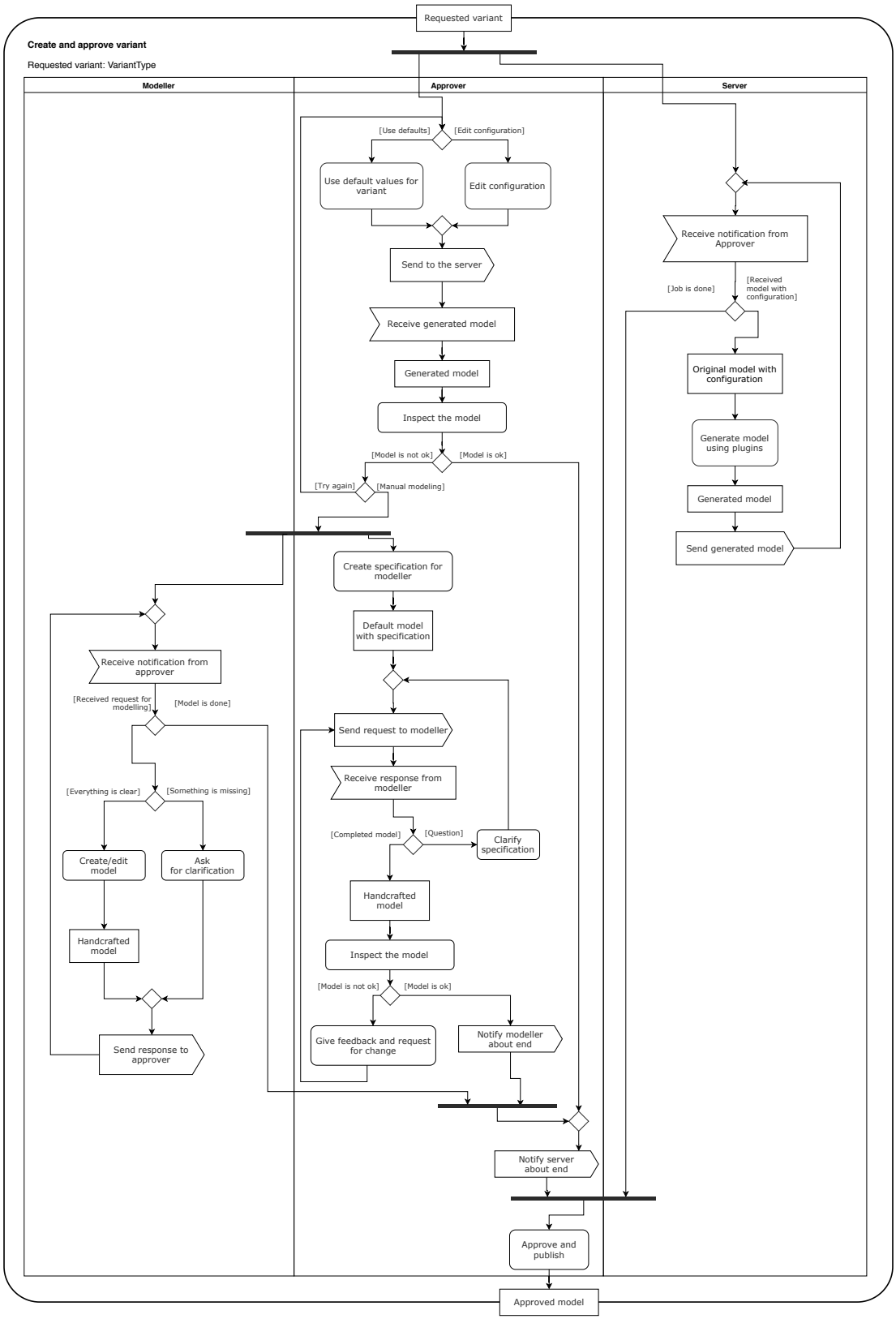


Obrázek 3.3: Základní schvalovací proces II.



Obrázek 3.4: Spouštění schvalování variant.

kterými se zabývala v rámci projektu VMCK Denisa Sůvová ve své bakalářské práci[30] či Michal Zajíc ve své bakalářské práce[31]. V administraci budou uložené sekvence tranformací pomocí tohoto systému včetně výchozích hodnot parametrů, kterými se tyto pluginy nastavují. *Grafik*, který nemusí být nutně stejný jako v předchozí fázi, nejprve zkusí vygenerovat varianty pomocí tohoto automatického generování. Po dogenerování, které může trvat i jednotky až desítky hodin, zkontroluje výsledek a v případě spokojenosti jednotlivé varianty může schválit. Není-li spokojen, zkusí obměnit zmiňované parametry. Takto je generuje až do momentu, kdy je spokojen, nebo když usoudí, že pluginy nejsou natolik mocné a bude nutné tuto variantu doupravit ručně, v tomto okamžiku se osloví znovu *Modelář*, kterému se předá výchozí model a detail k vytvoření konkrétní varianty. Pak se odehrává znovu schvalovací a odevzdávací proces, ale ve zjednodušené podobě. Jakmile je *Modelář* spokojen, odešle model ke schválení, *Grafik* může poprosit o přepracování, nebo ho schválí. Jednotlivé varianty se po schválení posílají do produkce a nečeká se na schválení všech variant.



Obrázek 3.5: Schvalovací proces jendotlivé varianty.

3.3 Uživatelské rozhraní

V rámci návrhu uživatelského rozhraní budu postupovat podle metodiky *Task-based UI design*, se kterou jsem se seznámil v rámci studia na ČVUT FIT v předmětu MI-NUR.

3.3.1 Task List

Před samotným návrhem uživatelského rozhraní jsem prvně vypracoval *Task List*, který popisuje uživatelské rozhraní z pohledu uživatele a jeho cílem je pomoci se základním návrhem UI. Do značné míry vychází z případů užití, které jsem vypracoval a popsal v kapitole 2.4. Z tohoto seznamu by měly být odvoditelné veškeré další atomické podúlohy.

Úlohy z tohoto seznamu jsem pak sloučil do skupin. Cílem tohoto sloučení je jejich kategorizace, která má pomoci při návrhu uživatelského rozhraní. Je více přístupů, kterými lze jednotlivé úlohy roztřídit, jedním z nich je např. to, jestli se jedná o zobrazovací úlohy, nebo zda se vyžaduje interakce s uživatelem. Dalším krokem, kterým se budu zabývat i já, je rozdělení dle velikosti plochy na obrazovce. Kromě těch vyčlením i skupinu pro ovládací prvky, které mají být vidět na každé obrazovce pro přihlášeného uživatele.

3.3.1.1 Pro všechny uživatele

Pro všechny uživatele jsou dostupné tyto úkoly:

Ovládací prvky, důležité info

- Odhlásit se
- Navigace
  - Dashboard
  - Profil
  - Seznam entit *structure*
- Informace, kde se uživatel nachází
- Zobrazení oznámení (chybové, informativní ...)
- Získat kontakt na správce systému

Obsahové prvky

- Velké (hlavní obrazovky)
  - Přihlásit se pomocí e-mailu a hesla

  - Zobrazit svůj profil
  - Editovat svůj profil včetně změny hesla
  - Zobrazit dashboard
  - Zobrazit seznam entit *structure*
  - Filtrovat entity *structure*
  - Zobrazit detail entity *structure* včetně schvalování a verzí
  - Zobrazit/prozkoumat 3D model entity *structure*

3.3.1.2 Role Historik

Pro uživatele s rolí *Historik* a *Admin* jsou dostupné tyto úkoly:

Ovládací prvky, důležité info

- Navigace (k prvkům definovaným pro všechny uživatele přibývají další)
  - Zobrazit založené entity *structure*

Obsahové prvky

- Velké (hlavní obrazovky)
  - Vytvoření entity *structure*
  - Editovat entitu *structure*
  - Zobrazit založené entity *structure*
  - Zobrazit rozpracované (neodeslané) entity *structure*
  - Zobrazit schvalované entity *structure*
  - Zobrazit schválené entity *structure*
- Malé (tlačítka s případnými dialogy, informace)
  - Nahrát doplňující soubory k entitě *structure*
  - Odeslat žádost o vymodelování entity *structure*

3.3.1.3 Role Grafik

Pro uživatele s rolí *Grafik* a *Admin* jsou dostupné tyto úkoly:

Ovládací prvky, důležité info

- Navigace (k prvkům definovaným pro všechny uživatele přibývají další)
  - Schvalované entity *structure*

Obsahové prvky

- Velké (hlavní obrazovky)
  - Zobrazit entity *structure* ke schválení bez schvalovatele
  - Zobrazit všechny schvalovací procesy daného uživatele
  - Zobrazit všechny otevřené procesy daného uživatele
  - Zobrazit všechny uzavřené procesy daného uživatele
  - Prozkoumat a případně schválit generované varianty schváleného 3D modelu entity
- Malé (tlačítka s případnými dialogy, informace)
  - Přijmout žádost o schvalování entity *structure*
  - Odejít ze schvalování entity *structure*
  - Schválit 3D model entity *structure* odeslaný ke schválení
  - Vrátit 3D model entity *structure* odeslaný ke schválení k přepracování s komentářem
  - Odeslat žádost o ruční vypracování varianty 3D modelu entity *structure*
  - Schválit variantu 3D modelu entity *structure* odeslanou ke schválení
  - Generovat varianty schváleného 3D modelu entity *structure* pomocí transfromačního systému včetně úpravy paramterů *structure*

3.3.1.4 Role Modelář

Pro uživatele s rolí *Modelář* a *Admin* jsou dostupné tyto úkoly:

Ovládací prvky, důležité info

- Navigace (k prvkům definovaným pro všechny uživatele přibývají další)
  - Schvalované entity *structure*

Obsahové prvky

- Velké (hlavní obrazovky)
  - Zobrazit entity *structure* ke schválení bez modeláře
  - Zobrazit všechny všechny modelovací procesy daného uživatele
  - Zobrazit všechny otevřené modelovací procesy daného uživatele
  - Zobrazit všechny uzavřené uzavřené procesy daného uživatele

- Malé (tlačítka s případnými dialogy, informace)
  - Přijmout žádost o modelování entity *structure* nebo její varianty
  - Odejít od modelování entity *structure*
  - Požádat o vyjasnění zadání k modelování entity *structure* nebo její varianty
  - Nahrát model entity

3.3.1.5 Role Admin

Pro uživatele s rolí *Admin* jsou dostupné tyto úkoly:

3.3.1.6 Ovládací prvky, důležité info

- Navigace (k prvkům definovaným pro všechny uživatele přibývají další)
  - Uživatelé
  - Sekvence tranformací

3.3.1.7 Obsahové prvky

- Velké (Hlavní obrazovky)
  - Zobrazit všechny uživatele
  - Zobrazit profil uživatele
  - Editovat profil uživatele včetně změny hesla a uživatelských rolí
  - Vytvářet nové uživatele
  - Zobrazit definované sekvence tranformací
  - Editovat definované sekvence tranformací včetně výchozích hodnot
  - Vytvořit sekvenci tranformací včetně výchozích hodnot
- Malé (Tlačítka s případnými dialogy, informace)
  - Mazat entity *structure*
  - Mazat uživatele
  - Mazat jednotlivé modely entity *structure*
  - Mazat sekvence tranformací
  - Uzavřít schvalovací proces k entitě *structure* bez schválení

Základem pro spokojenost uživatele je zprostředkování dobré přístupnosti nejčastěji používaných funkcionalit systému. Díky tomu mohou uživatelé provádět činnosti výkonněji. Jelikož není možné vyjít z reálných dat, protože toto administrační rozhraní se bude teprve spouštět, rozhodl jsem se nad touto problematikou aspoň zamyslet. Předpokládám, že většina lidí, co budou navštěvovat administrační systém pravidelně, tam bude chodit za účelem otevřených schvalovacích procesů, proto bude možné se dostat na aktuální schvalovací procesy jednoduše z menu a jejich přehled bude i na dashboardu, aby bylo rychle zjistitelné, které schvalovací procesy se posunuly, a zejména kde je vyžadována akce ze strany aktuálního uživatele. Lidé, co budou chodit do administračního systému občas, budou dle mého názoru primárně využívat přehledy entit *structure* a dá se očekávat i zapojení do schvalování. Proto bude zajištěna snadná přístupnost těchto primárních úkolů. Zároveň jsou tyto primární úkoly těmi nejdůležitějšími úkony, kvůli kterým tento systém vůbec vzniká. Neočekávám příliš časté vytváření, editaci či mazání entit *structure*, manipulaci s uživatelskými účty či sekvencemi transformací a podobně, což jsou úkony mimo těžiště tohoto systému.

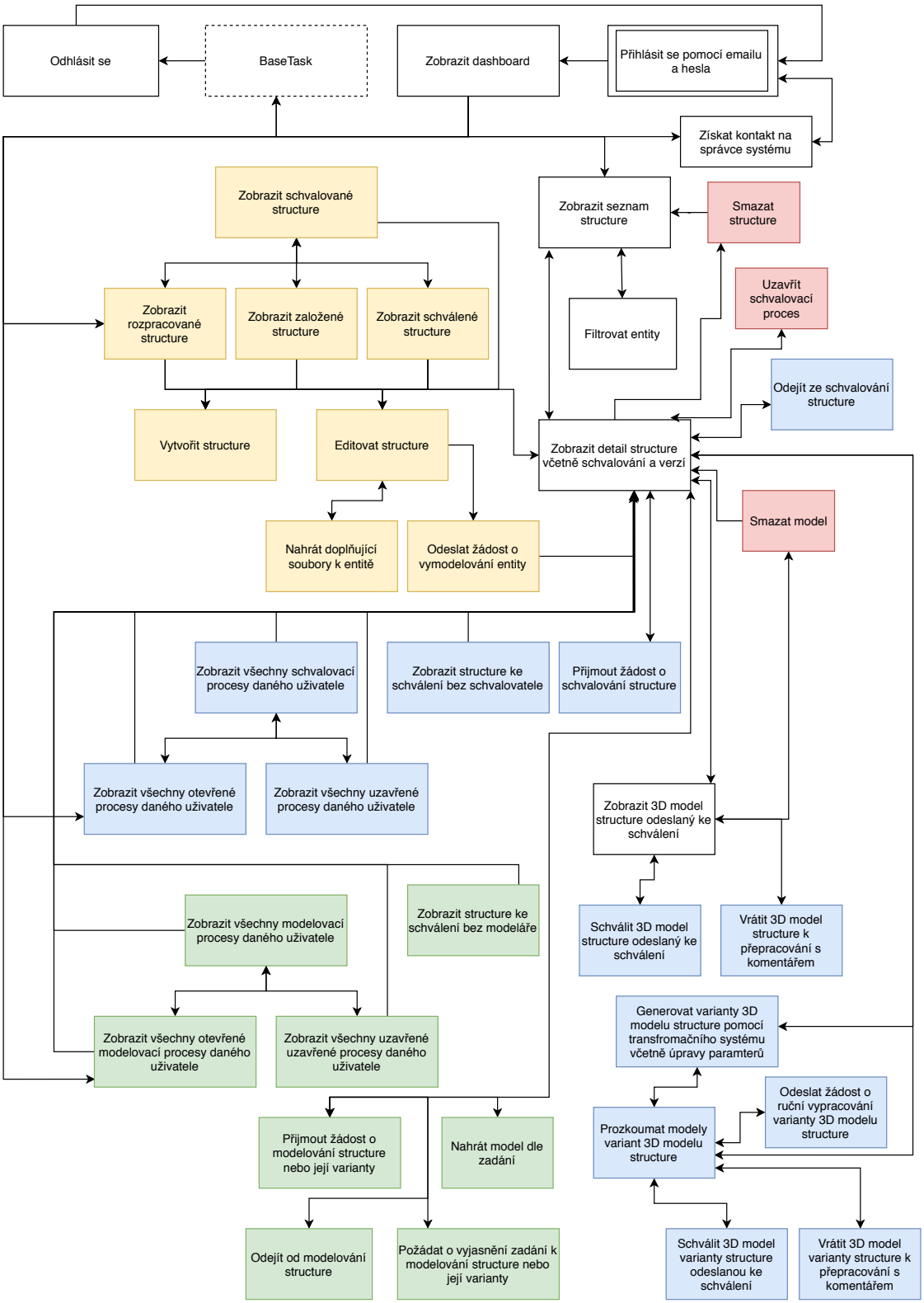
3.3.2 Task Model

Jako doplnění Task Listu uvedeného v kapitole 3.3.1 jsem vytvořil Task Model. Task Model je abstraktní struktura, např. ve formě stromu, grafu, vývojového diagramu, v níž je načrtnutá množina hlavních tasků a jejich vzájemné propojení. Hlavní úkoly pak mohou být dále rozloženy na menší podúlohy. Cílem je pomoci se zorientovat ve vzájemných závislostech a vazbách jednotlivých úkolů. Dále je možné i načtrnout tok dat v případě grafového vyobrazení.

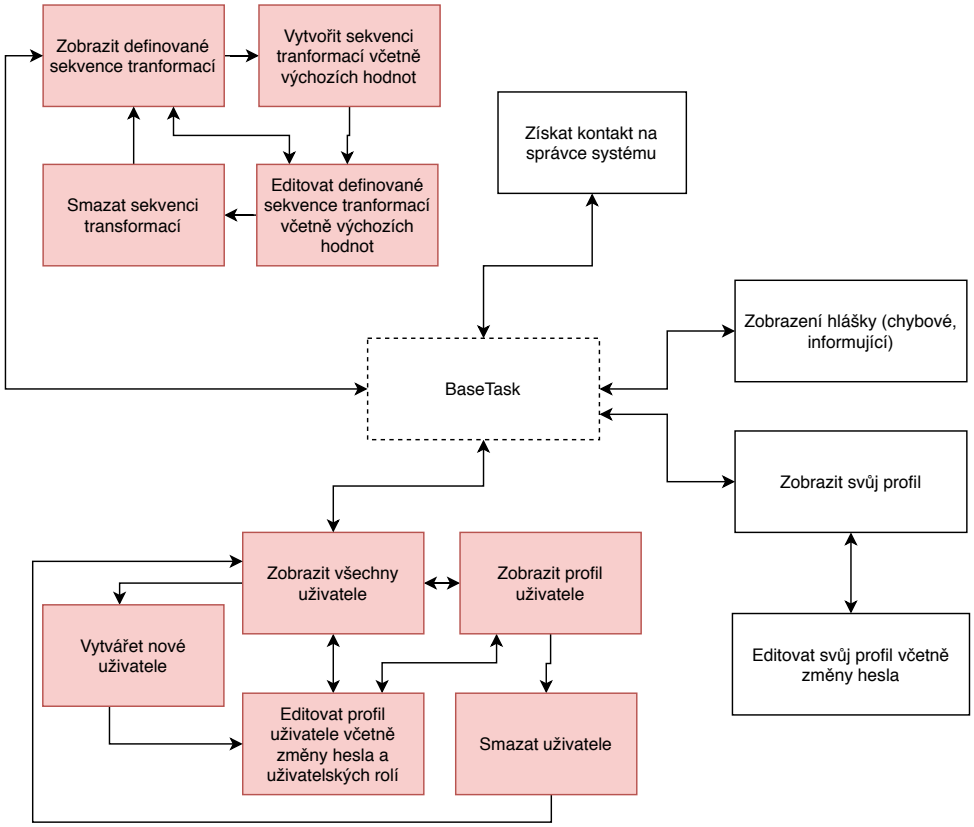
Já jsem se rozhodl pro vyobrazení ve formě grafu. Ten jsem z důvodu přehlednosti rozdělil na dvě části. Vytvořil jsem speciální základní úkol *BaseTask*, na který jsem připojit veškeré úkoly spojené s ovládacími prvky a důležitými informacemi. Tyto jsou dostupné ze všech dalších tasků pro přihlášené uživatele (tedy kromě tasku *Přihlásit se pomocí e-mailu a hesla* a *Získat kontakt na správce systému*). Tento task je taky spojujícím prvkem obou částí Task Graphu. První část, která je na obrázku 3.6, je zaměřená na manipulaci se samotnými entitami *structure* a jejich schvalováním. Druhá část, která je na obrázku 3.7, tvoří administrativní část systému. Jednotlivé úkoly jsou označeny barvou dle toho, která role je nutná pro jejich vykonání. Legendu k použitým barvám lze vidět na obrázku 3.8 (Pro barvoslepé jsou role odvoditelné z rozdělení Task Listu v kapitole 3.3.1).

3.3.3 Wireframy (Lo-Fi prototyp)

Na základě Task Listu a Task Modelu jsem přešel k prototypování UI. V prvotní fázi jsem vytvářel wireframy jako Lo-Fi (Low-fidelity) prototyp pomocí papíru a tužky. Důvodem je co nejkratší a nejpohodlnější způsob výroby



Obrázek 3.6: Task Graph – první část.

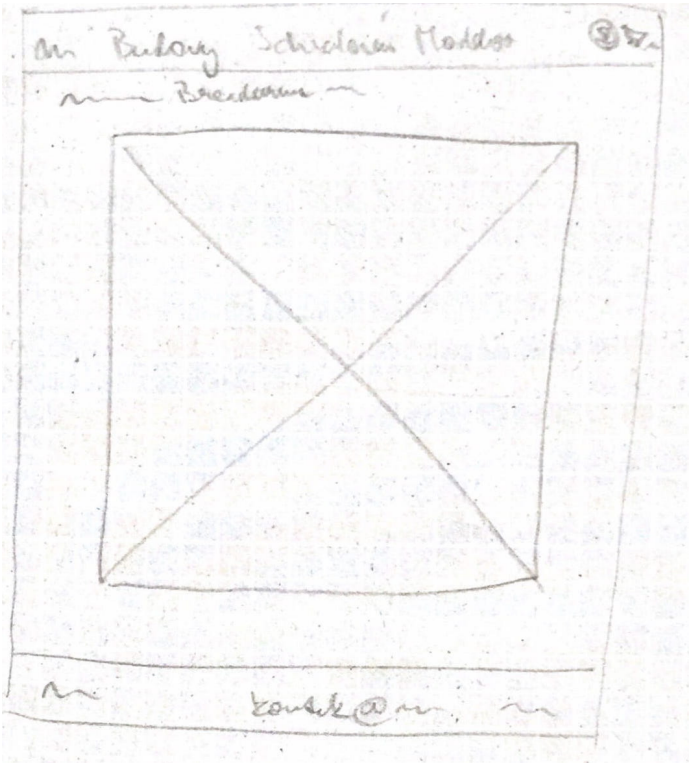


Obrázek 3.7: Task Graph – druhá část.

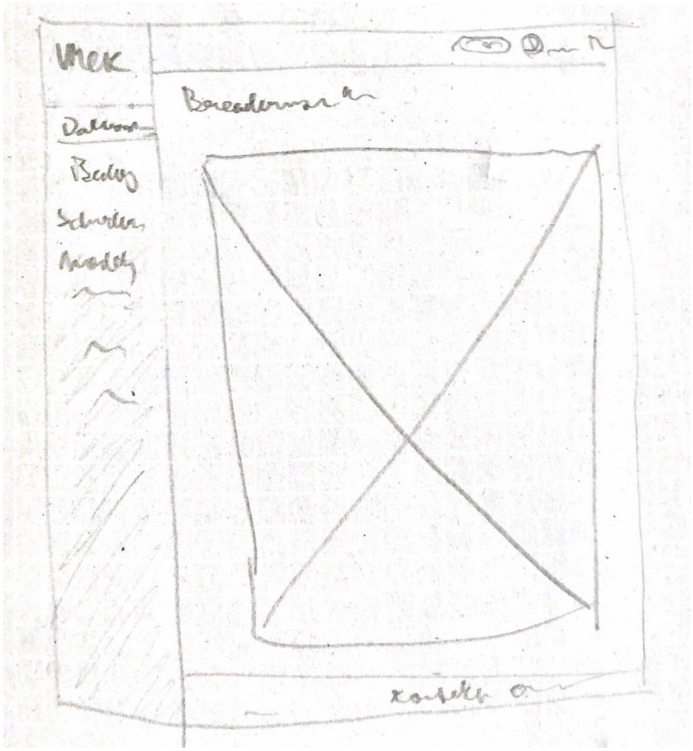
těchto prototypů, jelikož se často provádí úprava a rovněž se v této fázi jedná hlavně z velké části o rozmístění jednotlivých prvků a základu layoutu. Jsou dvě možnosti: *From top*, ve které se prvně vytvoří základní layout a do něho se vpasovávají jednotlivé části. Další způsob je *From bottom*, ve kterém se prvně naprototypují jednotlivé části a až na základě nich se k nim vytvoří uzpůsobený layout. Dle doporučení z předmětu MI-NUR jsem použil kombinací obou a nejprve jsem uvažoval nad velikostí jednotlivých částí a nad tím, kolik místa budou potřebovat, a až pak následně jsem vytvářel obecný layout.

Mým cílem bylo hlavně rozvrhnout průběh jednotlivých úkolů a rozložení prvků, a ne dbát na detaily. Pokud máte grafické cítění, tak mé wireframy prosím omluvte. Základní wireframy layoutů lze vidět na obrázcích 3.9 a 3.10, jedná se o dvě varianty, mezi kterými jsem se nebyl schopen rozhodnout, takže jsem to odložil na další fázi HiFi prototypu (viz kapitola 3.3.4) a pro obrazovky jsem počítal s variantou I. Na ukázku jsem přidal wireframe pro výchozí obrazovku na obrázku 3.11, obrazovky pro schvalování 3.12 . Veškeré další návrhy po iteracích lze najít v příloze B.

Obrázek 3.8: Task Graph – legenda.



Obrázek 3.9: Základní layout I.

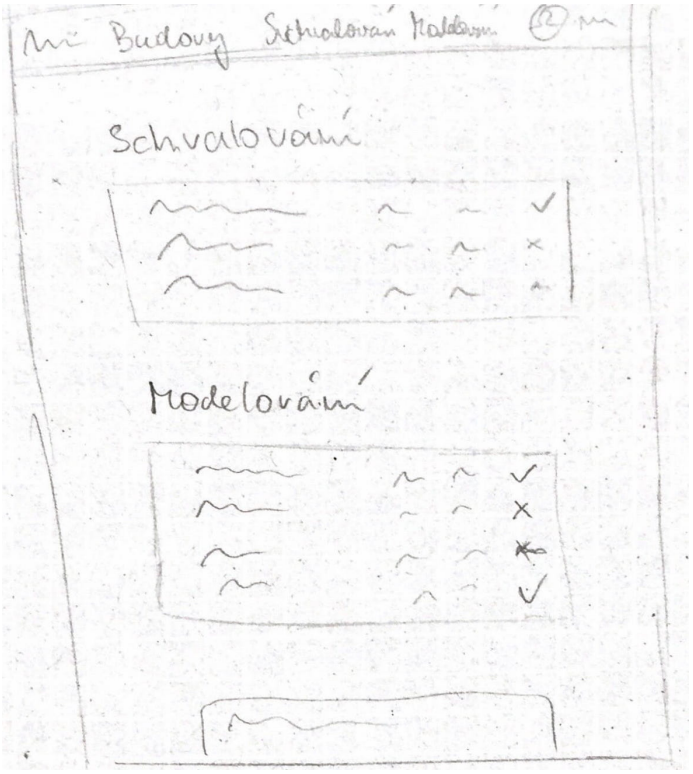


Obrázek 3.10: Základní layout II.

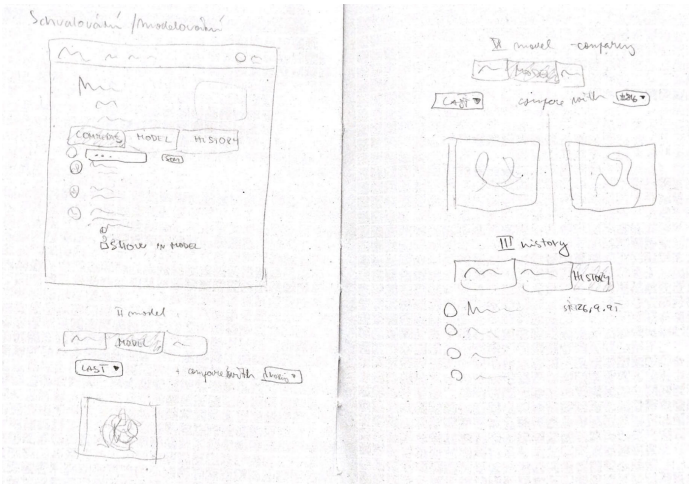
3.3.4 Hi-Fi prototyp

Díky Lo-Fi prototypu jsem si ujasnil rozvržení prvků, průchody administračním rozhraním, není problém si v tomto prototypu rychle vyzkoušet více variant. U této fáze není nutné uživatelské testování a stačí vlastní hodnocení. Na základě tohoto prototypu jsem přistoupil k tvorbě Hi-Fi (High-Fidelity) prototypu, který v optimální podobě má tvořit iluzi výsledné aplikace. Cílem je co nejvíce se přiblížit věrnosti ve vizuálu a interaktivitě. Tento prototyp by měl běžet na cílové platformě, funkční aplikační logika není nutná. Je důležité naprototypovat hlavní části uživatelského rozhraní a ty na tomto prototypu odladit.[32]

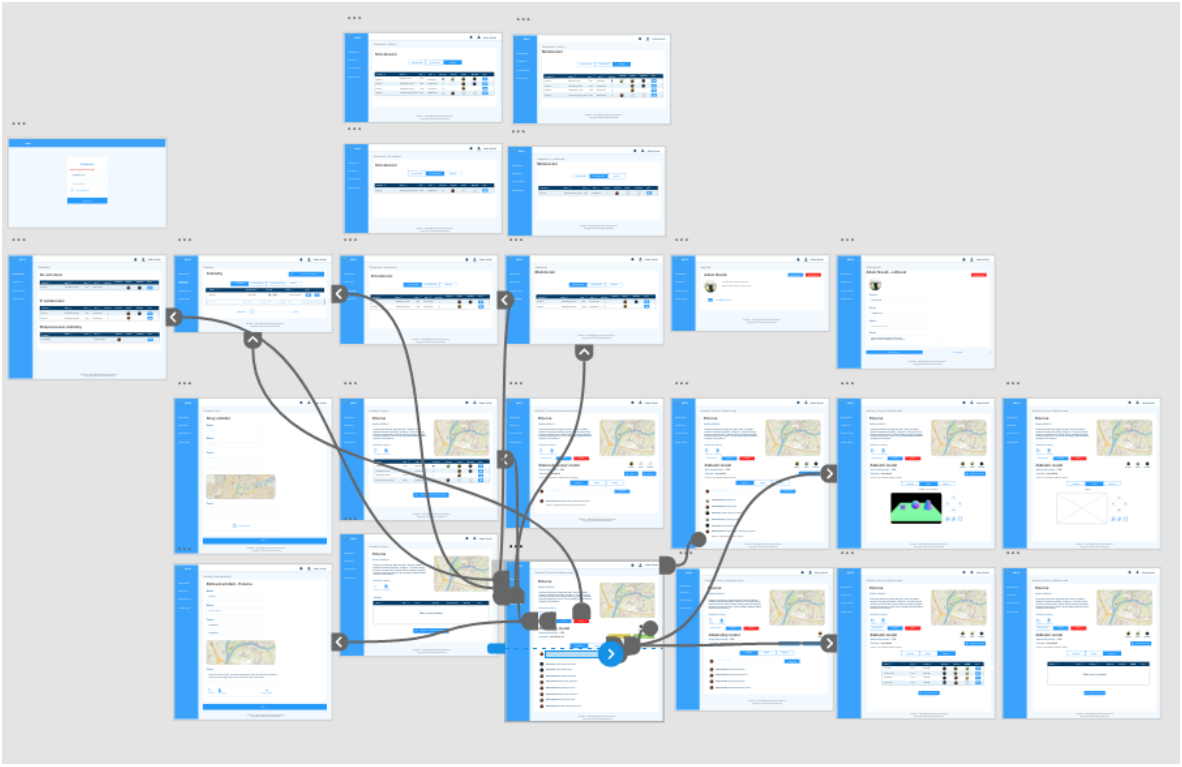
Já jsem Hi-Fi prototyp vytvořil v programu Adobe XD, který mi z programů, s nimiž jsem měl tu čest – Axure, Balsamix a Adobe XD, vyhovoval nejvíce. Program je zdarma dostupný na `https://www.adobe.com/cz/products/xd.html`. Zaměřil jsem se na návrh webové aplikace ve variantě pro desktop. Naprototypoval jsem nejdůležitější a nejsložitější část uživatelského rozhraní, a to manipulaci s entitami *strucutre*, jejich vytváření, schvalování a připomínkování, generování variant. Část pro administraci tohoto administračního systému – správa uživatelů, sekvencí transformací atd., což jsou úkoly, které bude používat malá skupina *Adminů*, u nichž se předpokládá vy-



Obrázek 3.11: Dashboard.



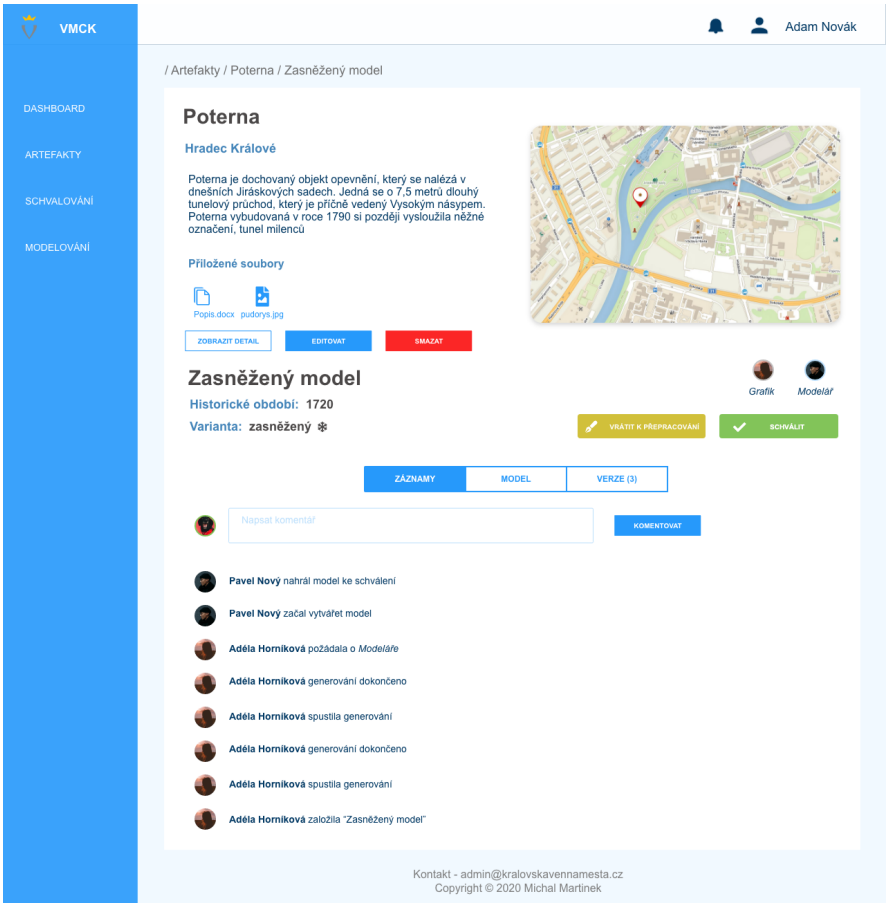
Obrázek 3.12: Schvalování/modelování.



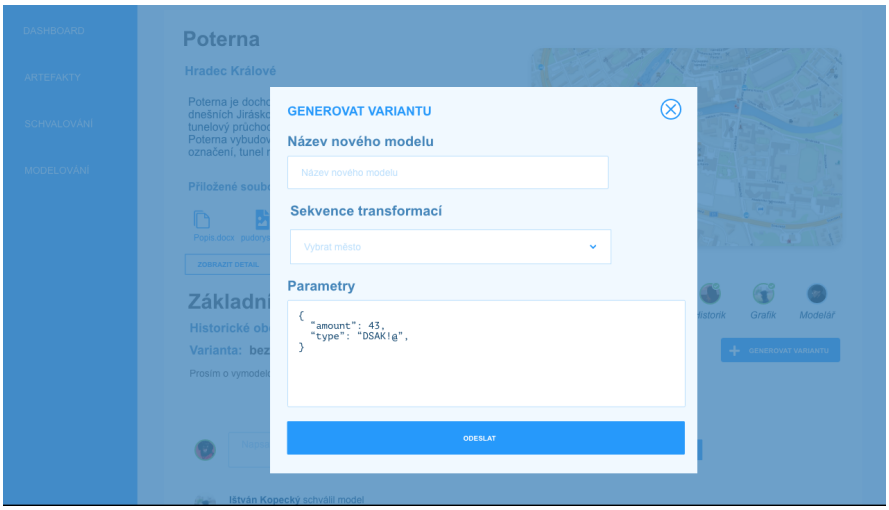
Obrázek 3.13: Prototypování modelu.

soká znalost tohoto systému –, jsem neprototypoval. Tato část rovněž nebude komplikovaná.

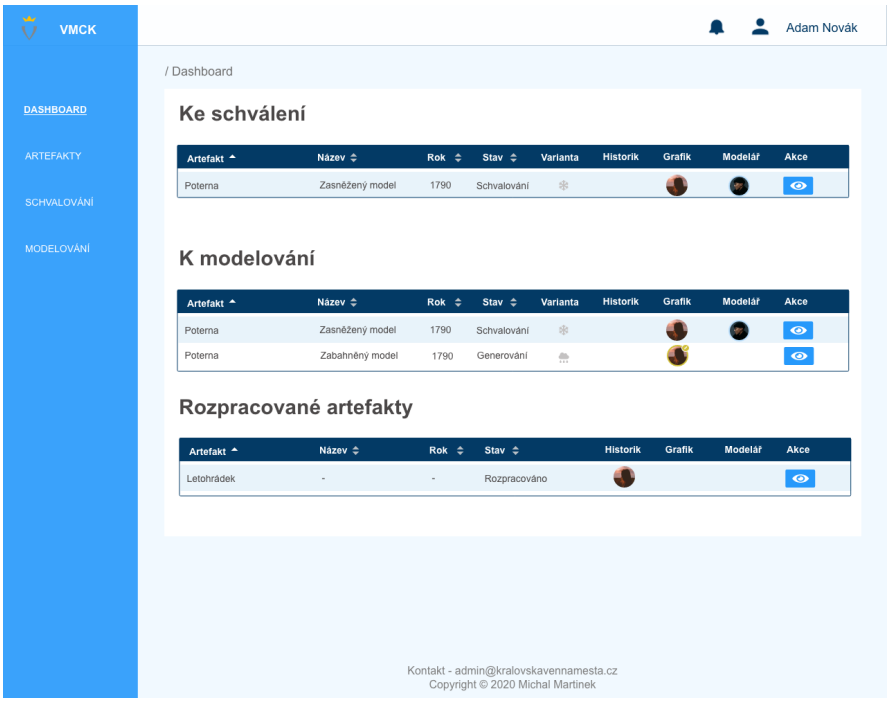
V rámci prototypu jsem navrhnul celou řadu obrazovek, pro interaktivní průchod prototypu je nutné otevřít soubor *ui/hifi.xd* z přiloženého média v programu Adobe XD, jako alternativa je přibalený i soubor *ui/hifi.pdf*, ve kterém ale nejde vidět spoustu variant stavů. Pro ukázku zde přikládám několik snímků z Wi-Fi prototypu. Na obrázku 3.13 lze vidět propojení jedné obrazovky s dalšími v tomto prototypovacím nástroji. Na obrázku 3.14 je snímek schvalování jednotlivého modelu, na 3.15 lze vidět modální okno pro spuštění generování varianty modelu a na obrázku 3.16 je k nahlédnutí úvodní dashboard. Řadu dalších obrazovek lze najít i v příloze C.



Obrázek 3.14: Schvalování modelu.



Obrázek 3.15: Generování varianty.



Obrázek 3.16: Dashboard.

Kapitola 4

Uživatelské testování použitelnosti

Navržený prototyp administračního rozhraní jsem se rozhodl před samotnou implementací otestovat na použitelnost, jelikož změny v tomto protypu jsou daleko jednodušší na provedení. Cílem tohoto testování bylo otestování použitelnosti, funkčnosti a srozumitelnosti tohoto rozhraní a nalezení problematických míst a bariér, které by uživatele mátly nebo práci se systémem znepříjemňovaly.

Problémem ale byla aktuální situace – karanténní opatření kvůli koronaviru[6]. Proto jsem se rozhodl provést aspoň guerilla testování na lidech ve svém okolí. Proč guerilla? Protože jsem ho uskutečnil v domácích podmínkách a s menším počtem testerů. Rovněž výstupy a metody záznamu nejsou takové, jako by byly třeba ze školního SAGElabu, ale i přesto tato forma testování dokáže objevit mnohé nedostatky. „Testování s tak málo lidmi nemůže vést ke statisticky platným výsledkům. Vzorky jsou příliš malé na to, aby se jimi statistika vůbec zabývala. Ale účelem tohoto druhu testování není cokoliv prokázat, cílem je identifikovat hlavní problémy a napravit je. Funguje to prostě, protože většina typů problémů, které je třeba napravit, je tak zřejmá. že zde není třeba nic ‚prokazovat'."[33] V této knize se rovněž píše, že nejlepší je začít testovat co nejdříve. Nejhorší je začít testovat, když už je hotová prakticky celá implementace. Tento autor rovněž nabádá k použití tohoto typu testování, který jsem prováděl i já, k jednoduchým neformálním testům s malým vzorkem, které dokáží mnohé problémy odhalit a jsou časově a finančně nenáročné.

[6] viz https://cs.wikipedia.org/wiki/Pandemie_covidu-19_v_%C4%8Cesku

4.1 Nastavení testování

Kvalitativní uživatelské testování se 4 participanty. Testování proběhlo v domácnostech testerů v prototypovacím nástroji Adobe XD na souboru VMCK.xd, který je k dispozici v příloze.

4.2 Cílová skupina

Ideální je vybírat testery z cílové skupiny, takže v případě tohoto administračního rozhraní to budou primárně osoby mezi 20 – 60 roky, minimálně se středoškolským, dost často vysokoškolským vzděláním zvládající aspoň základy práce na PC. U části uživatelů, skupiny správců a grafiků, se dá očekávat velmi dobrá znalost práce s PC. Část z nich bude mít vzdělání v oblasti historii a geografii.

Proto jsem vybral ze svého okolí blízké, kteří jsou sice v mladší části spektra této cilové skupiny, jedná se však o vysokoškolsky nebo středoškolsky vzdělané lidi, kteří jsou zvyklí pracovat s počítačem, většinou však nikoliv na mistrovské úrovni.

4.3 Průběh testování

Testování probíhalo v domácnostech testerů na laptopu Macbook Pro, dle jejich možnosti volby jej ovládali pomocí touchpadu nebo myši. Veškeré testování probíhalo samostatně v klidně místnosti a byl jsem přítomen jen já, coby moderátor, a tester. Z průběhu testu byl pořízen videozáznam dění na obrazovce a zvukový záznam. O průběhu testu a tomto záznamu byli účastníci testování informování a s jeho pořízením a zveřejněním pro potřeby této diplomové práce souhlasili. Na záčátku byli krátce obeznámi s projektem a průběhem testování, dále se spustilo nahrávání a proběhly jednotlivé fáze testování – pretest, testovací úkoly a dojem testera.

4.4 Fáze uživatelského testování

4.4.1 Pretest

- Jak se jmenuješ?
- Jak se dnes cítíš?
- Pohybuješ se v IT, případně v návrhu UI/vývoji aplikací?
- Je toto první uživatelské testování, kterého se účastníš?

4.4.2 Testovací úkoly

1. Přihlásit se do administračního rozhraní.
    - Víš, kde se nacházíš po přihlášení?
2. Jsi *Historik* a potřebuješ založit nový záznam budovy.
    - Nahrát soubor.
    - Zajistit vymodelování budovy dle materiálů.
3. Chceš změnit e-mail na *Novy-email@mail.com*.
4. Potřebuješ zobrazit veškeré schvalovací procesy.
5. Jsi *Grafik* a měl bys schválit zasněženou verzi Poterny.
6. Vzpomněl sis však, že jsi před schválením zapomněl prozkoumat 3D model, proto to proveď teď.
7. Chceš vytvořit pomocí automatického systému verzi aktuálního modelu v mlze.
8. Práci v systému jsi dokončil, proto se odhlas.

4.4.3 Dojem testera

- Popíšeš prosím svůj celkový dojem?
- Co byl nejsložitější úkol?
- Je něco, co bys v aplikaci uvítal, co by ti zlepšilo práci v něm?

4.5 Report uživatelského testování

Daniela Borošová

1. Pretest

- Vyskoškolské vzdělání v IT
- Programátor SW
- Cítila se báječně.
- Není to první uživatelské testování.

2. Testovací úkoly

2.1 Přihlásit se

- Tester se bez problémů přihlásil.
- Tester pochopil rámcově, kde se nachází, jen byl zaražen přehršlem věcí na dashboardu.

2.2. Vytvořit nový záznam budovy

- Tester našel bez problémů cestu k vytvoření.
- Nahrávání pomocí drag and drop není dostatečně jasné.
- Chybějící zpětná vazba o vytvoření.
- Požádání o vytvoření výše na obrazovce, když není žádný dostupný model.

2.3. Změnit e-mail

- Tester bez zaváhání změnil e-mail pomocí Editace profilu

2.4. Zobrazit všechna schvalování

- Tester si pamatoval i cestu přes dashboard, ale zvolil rychlejší přes menu.

2.5. Schválit zasněženou variantu Poterny

- Tester si pamatoval i cestu přes Dashboard, ale zvolil rychlejší přes menu.

2.6. Prozkoumat 3D model

- Tester bez problémů našel detail modelu a schválení.
- Bylo jasné i použití filtru v tabulce.

2.7. Generovat variantu v mlze

- Tester bez problému provedl úkol.

2.8. Odhlásit se

- Tester bez problému provedl úkol.

3. Dojem testera

- Rozhraní by mělo být hezčí a více lákající ke kliknutí. Moc výrazné záhlaví tabulek. Důležité prvky by měly být více viditelné. Chybějící zpětná vazba o aktivovatelných prvcích. Např. na hover efekt přidat k uživatelům jméno.
- Nejsložitější bylo se zorientovat v detailu modelu. Oddělit artefakt.
- Timeline pro schvalování, kde jsou jednotlivé úkony ve schvalovacím artefaktu. Klidně navázat akce na body, nebo zobrazení více informací o provedeném bodu (čas, kdo, verze).

Erik Klemš

1. Pretest

- Student právnické fakulty
- Cítil se trochu unaveně, ale jinak v pohodě.
- Je to první uživatelské testování.

2. Testovací úkoly

2.1. Přihlásit se

- Tester neměl s přihlášením problém.
- Tester pochopil, že jsou na dashboardu aktuální rozpracované artefakty.

2.2. Vytvořit nový záznam budovy

- Tester našel bez problémů cestu k vytvoření.
- Chápal vytvoření modelu jako další fázi, s jejímž vytvoření neměl problém.

2.3. Změnit e-mail

- Tester měl problém jen se scrollováním na touchpadu, se zbytkem úkolu neměl problém.

2.4. Zobrazit všechna schvalování

- Tester bez problému provedl úkol.

2.5. Schválit zasněženou variantu Poterny

- Tester bez problému provedl úkol. Ale neměl úplně jasno o úspěšnosti schválení akce ani o stavu v rámci schvalovacího procesu.

2.6. Prozkoumat 3D model

- Chtělo by to lepší rozlišení mezi modely (3D objekty) a 3D modely v systému, což bylo pro testera matoucí, ale neměl s nalezením větší problémy.

2.7. Generovat variantu v mlze

- Tester bez problému provedl úkol. Jen by to chtělo lepší zpětnou vazbu o úspěšném vygenerování.

2.8. Odhlásit se

- Tester bez problému provedl úkol.

3. Dojem testera

- Rozhraní je dle něho relativně pořádku.
- Největší potíže měl s používáním daného počítače.
- Nic podstatného mu nechybělo.

Adam Janas

1. Pretest

- Vysokoškolé vzdělání v oblasti geografie.
- Kancelářská činnost, zkušenost s administrací článků na jedné webové stránce.
- Cítil se dobře.
- Je to první uživatelské testování.

2. Testovací úkoly

2.1. Přihlásit se

- Tester neměl s přihlášením problém.
- Tester byl zvyklý na dashboard.

2.2. Vytvořit nový záznam budovy

- Tester měl problém s vyhledáním formuláře, dáno asi označením a malou znalostí domény.
- Další fáze – požádání o modelování – proběhla bez problému.

2.3. Změnit e-mail

- Tester měl problém s pochopením vstupu. Samotné nalezení editace profilu bylo okamžité.

2.4. Zobrazit všechna schvalování

- Tester bez problému provedl úkol.

2.5. Schválit zasněženou variantu Poterny

- Tester bez problému provedl úkol. Ikonka sněhu velmi pomohla.

2.6. Prozkoumat 3D model

- Uživatel měl větší problém s nalezením modelu. Pomohlo by lepší odlišení v názvech.

2.7. Generovat variantu v mlze

- Tester bez problému provedl úkol.

2.8. Odhlásit se

- Tester bez problému provedl úkol.

3. Dojem testera

- Přijemný pocit, byl nadšen z modrého rozhraní. Nebolí z toho oči.
- Nejsložitější bylo zobrazit model, chtělo se to zamyslet a hledat.

Michael Botur

1. Pretest

- Student fakulty tělesné výchovy a sportu.
- Cítil se v pohodě, jen unavený po fyzické stránce.
- Je to první uživatelské testování.

2. Testovací úkoly

2.1. Přihlásit se

- Tester neměl s přihlášením problém.
- Tester nebyl zvyklý na dashboard, ale rámcově pochopil, co mu tato obrazovka nabízí.

2.2. Vytvořit nový záznam budovy

- Tester chvíli zkoumal, ale po pár klicích formulář nalezl a úspěšně vyplnil.
- Testerovi chyběla zpětná vazba o požádání modelování.

2.3. Změnit e-mail

- Tester bez problémů změnil e-mail, ale chyběla mu zpětná vazba o uložení.

2.4. Zobrazit všechna schvalování

- Tester bez problému provedl úkol.

2.5. Schválit zasněženou variantu Poterny

- Ikonka sněhu pomohla v nalezení. Tester bez problému provedl úkol.

2.6. Prozkoumat 3D model

- Tester bez problému provedl úkol.

2.7. Generovat variantu v mlze

- Tester bez problému provedl úkol. Překvapilo ho jen zmizení modalního okna bez zpětné vazby.

2.8. Odhlásit se

- Tester bez problému provedl úkol.

3. Dojem testera

- Jednoduché rozhraní, po vizuální stránce příjemné.
- Jednoduše ovladatelné.
- Úkoly bez problémů.

4.6 Výsledky uživatelského testování

V průběhu testování byly zjištěny jisté problémy v uživatelském rozhraní. Tyto problémy byly zaznamenány do tabulky 4.1 i s jejich četností a prioritou, jejíž číselné hodnocení je stanoveno takto:

1. zanedbatelné,
2. viditelný problém,
3. problém lehce znepříjemní uživateli použití,
4. problém, který uživatele bude velmi frustrovat,
5. těžký problém, který uživatel nepřekoná.

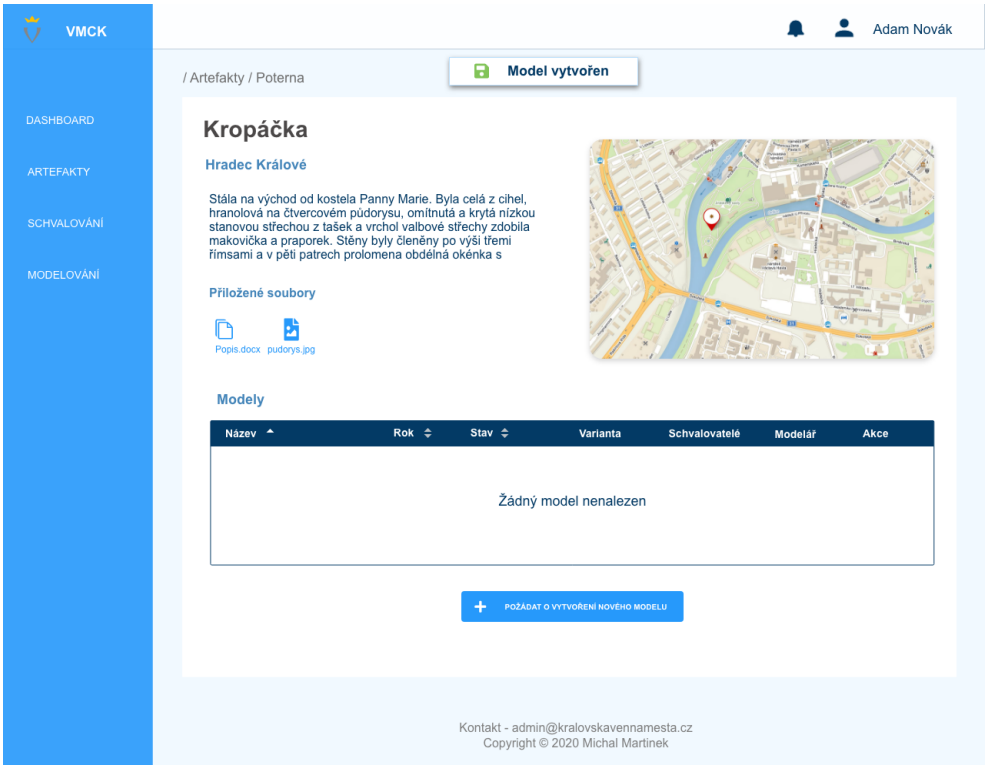
Tabulka 4.1: Nalezené problémy UI na základě uživatelského testování.

Obrazovka	Problém	Četnost	Priorita
Všechny formuláře	Chybějící oznámení po odeslání formulářů	4	3
Detail modelu	Lépe znázornit fázi schvalovacího procesu	4	3
Detail modelu	Málo oddělený artefakt od modelu	3	2
Detail modelu	Lepší odlišení modelu a samotného 3D modelu	3	2
Vytvoření artefaktu	Zmínit drag and drop pro nahrávání souborů	1	1
Detail artefaktu	Požádání o vytvoření modelu když žádný není	1	1
Všechny tabulky	Moc výrazné záhlaví	1	1
Vytvoření artefaktu	Snazší cesta k vytvoření artefaktu	1	1

4.7 Závěr

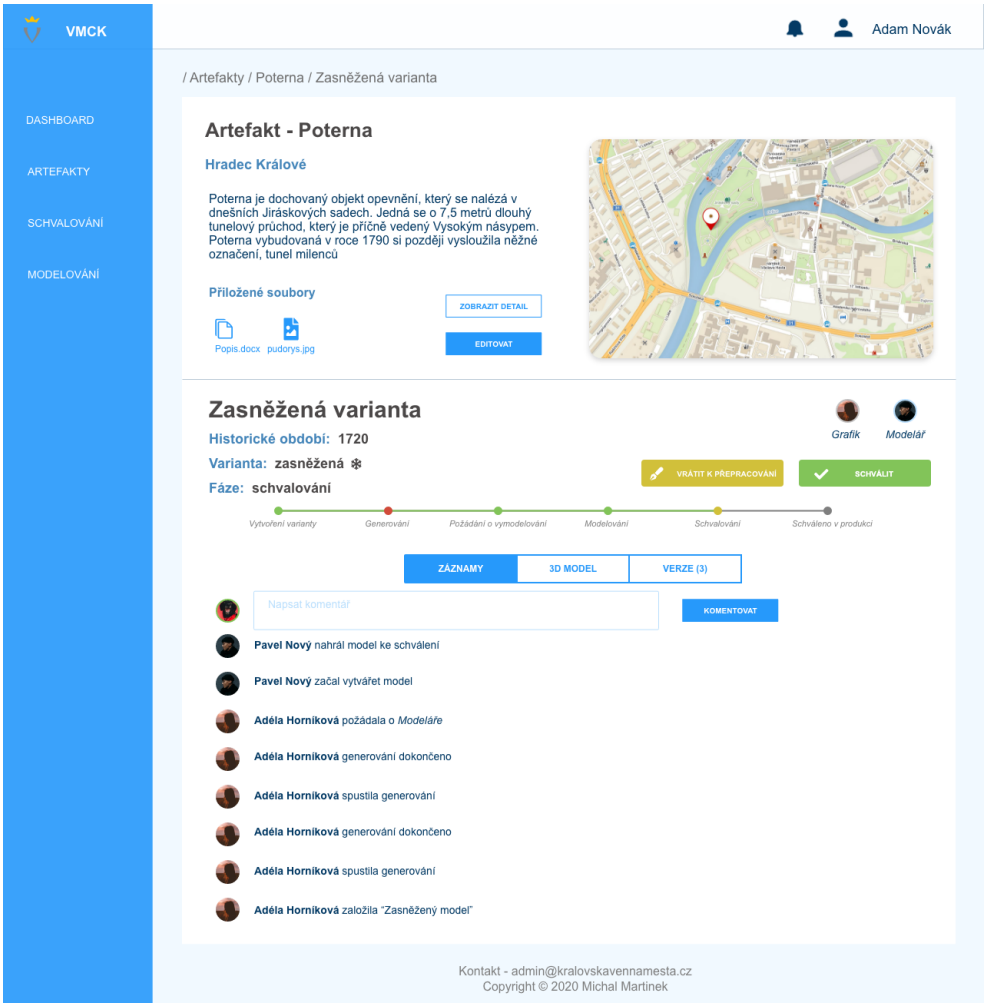
Uživatelské testování dopadlo lépe, než jsem původně očekával, a potěšila mě pozitivní zpětná vazba a to, že je systém použitelný bez větších překážek. Samozřejmě se objevily jisté problémy, které je záhodno vyřešit. Všechny problémy s prioritou větší než 1, tzn. ty, které nejsou zanedbatelné, jsem se rozhodl řešit. Upravil jsem tyto části UI v HiFi modelu. Na obrázku 4.1 lze vidět oznámení o úspěšném vytvoření. Tímto stylem bude řešen problém zpětné vazby i v jiných formulářích. Další problémy, jako lepší oddělení informací o artefaktu od varianty, málo jasný stav v rámci procesu schvalování a lepší odlišení modelu a 3D modelu, jsem vyřešil, jak lze vidět na obrázku 4.2 se schvalovací osou, na které po najetí se objevují dodatečné informace, jak je patrné na obrázku 4.3.

Testování v prototypovacím nástroji Adobe XD mělo ale i své stinné stránky, prostředí není 100% věrné výslednému. Formulářové prvky se nechovají interaktivně, komponenty na najetí nereagují stejně jako ve výsledku.

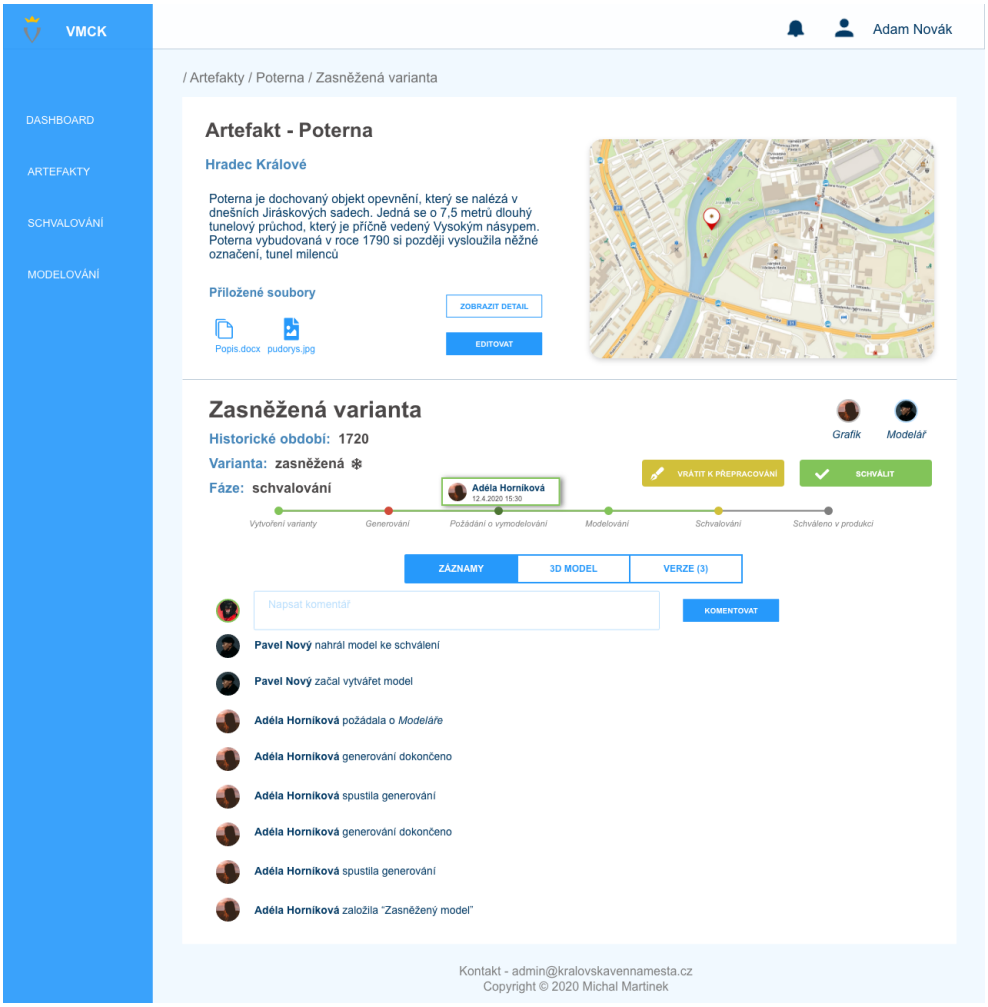


Obrázek 4.1: Ukázka zpětné vazby po odeslání formuláře.

Po přechodu na jiný snímek člověk zůstává v rámci obrazovky na bývalé pozici, takže není vidět záhlaví stránky. Je složité nasimulovat v tomto nástroji komplexnější systém, aby byly pokryty veškeré varianty obrazovek a testery nemátla jiná data. Proto jsem usoudil, že tento prototyp splnil svůj účel prototypovacího nástroje a ověřil základní funkcionalitu a průchody přes obrazovky. Další uživatelské testování má větší význam na prototypu/nedodělané verzi implementovaného administračního rozhraní pomocí webových technologií.



Obrázek 4.2: Vylepšený detail varianty se schvalováním.



Obrázek 4.3: Vylepšený detail varianty s detailem o schvalování.

Kapitola 5

Implementace

Po analýze funkčních požadavků, rozebrání případů užití, analýze technologií k vývoji, návrhu API, schvalovacího procesu a uživatelského rozhraní jsem konečně dospěl do fáze implementace. V této kapitole krátce popíšu použité technologie, fungování a strukturu projektu a rovněž to, které části administračního rozhraní se mi povedlo naimplementovat.

5.1 Použité technologie

Pro vývoj administračního rozhraní jsem se rozhodl použít technologie na základě analýzy přístupů k vývoji frontendu 2.6 a analýzy podpůrných knihoven 2.7. Pro základ aplikace je tedy použitý React, pro state management je použitá knihovna Redux. Pro zobrazování 3D modelů je zvolena knihovna Three.js.

Místo čistého JavaScriptu jsem se rozhodl použít TypeScript, což je jazyk postavený nad JavaScriptem, ke kterému přidává typy a typovou kontrolu. Funguje na principu, že kód se kompiluje do výsledného JavaScriptu a při této kompilaci se kontroluje typová integrita u částí, v nichž jsou uvedeny typy. Typy totiž nejsou povinné, takže jazyk zachovává flexibilnost JavScriptu, ale umožňuje definovat datové typy u libovolných částí, díky čemuž je kód více předvídatelný a čitelný pro vývojáře a např. snáze refaktovatelný. Cílem není mít u 100% zdrojového kódu definované typy, ale použít vlastnosti TypeScriptu u těch částí kódu, které jsou komplikované, nebo kde člověk očekává problémy.[34]

Pro základ vybudování UI jsem zvolil UI řešení a sadu připravených React komponent Ant Design[35]. Ta obsahuje komponenty pro základní mřížkový systém, layout, tlačítka, formulářové prvky a podobně. Díky těmto komponentám je možné systém vybudovat rychleji, než kdyby si všechny tyto prvky implementoval sám, rovněž jejich kvalita z hlediska vzhledu a uživatelské přívětivosti (očekávané chování v rámci různých stavů komponenty) je na vysoké úrovni. Pro tvorbu administračního rozhraní jsem použil šablonu Ant

Design Pro, kterou nabízí. V rámci ní jsem se rozhodl ponechat i preprocesor LESS pro tvorbu kaskádových stylů, ten nabízí místo CSS (Cascading Style Sheets) řadu šikovných vlastností jako např. zanořování bloků do sebe, mixiny a podobně.

Pro knihovnu Redux jsem se rozhodl použít malou wrappující knihovnu Dva, která mimo jiné nabízí podporu pro asynchronní akce, které jsou potřeba při dotazech na API.

Jelikož API bylo pro velkou část systému v době vývoje nedostupné, uchýlil jsem se k mockování tohoto kontraktu, jehož návrh jsem popisoval v sekci 3.1. To probíhalo přes jednoduchou nádstavbu nad NodeJS frameworkem Express, který nabízí routing systém, načítání parametrů, hlaviček a parsování obsahu požadavků, dále pak velkou řadu dalších funkcionalit, která však pro tyto účely nebyla potřeba.

5.2 Struktura projektu

Struktura projektu odpovídá obsahu složky se zdrojovými kódy *src/impl* na přiloženém médiu a je viditelná na obrázku 5.1 a vychází do značné míry z použité šablony pro React projekt. Je víceméně standardní a proto jsem do ní příliš nezasahoval. Soubory jsou rozděleny do modulů podle jednotlivých částí aplikace – modul s entitami *structure*, správu uživatelů, přihlášení atd. K tomu jsou vyčleněny sdílené komponenty, služby, modely a složka s překlady. Tento způsob zajišťuje dostatečně přehledné rozdělení s adekvátní granularitou, avšak bez zbytečného zanořování do hloubky.[36]

5.3 Jak projekt nainstalovat a spustit

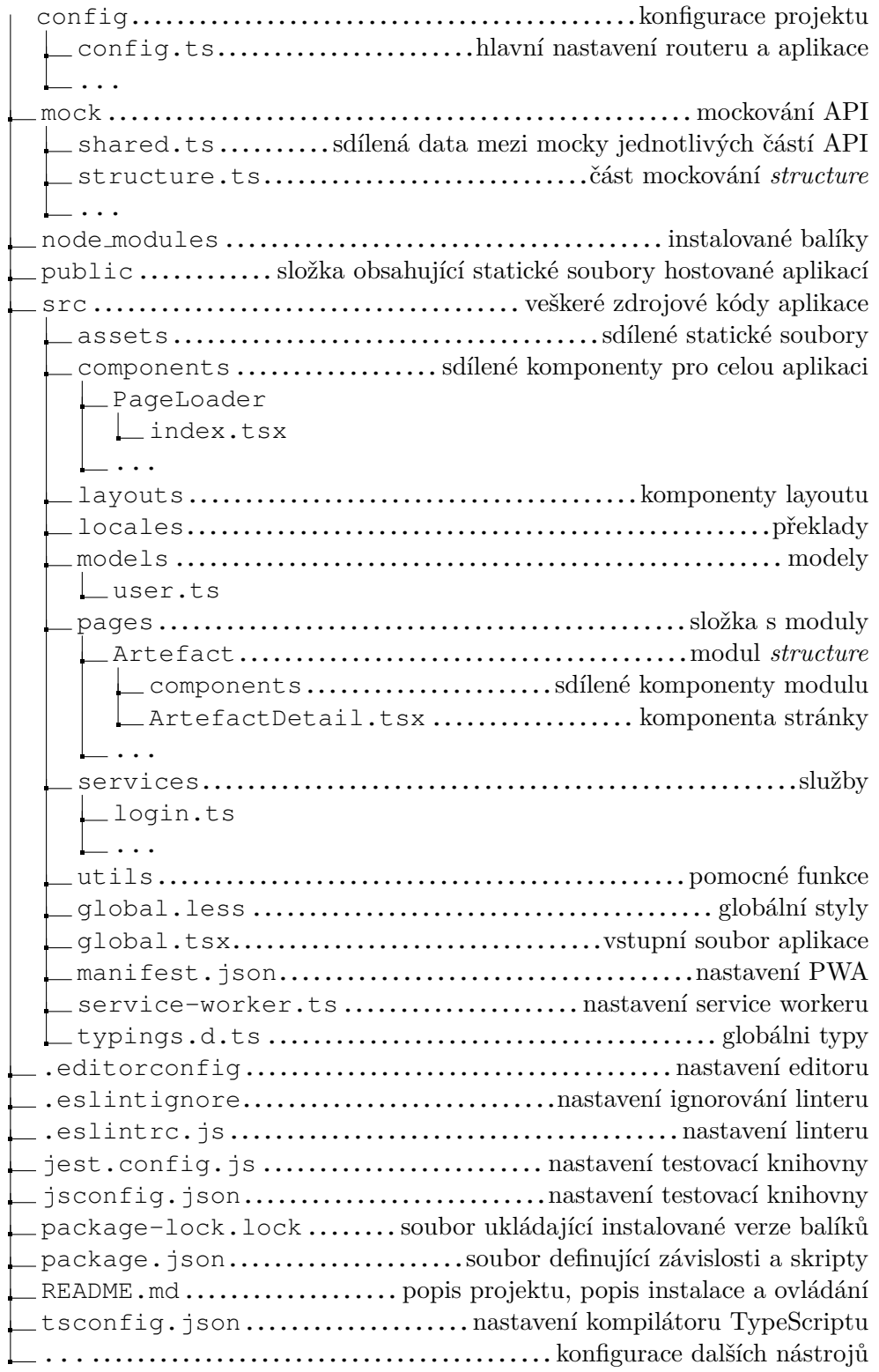
Pro spuštení projektu – vývojového serveru, kompilace produkční verze, spuštění testů či linteru a podobně – je nutné mít nainstalované JavaScriptové běhové prostředí NodeJS v minimální verzi 10.0.0, které je možné nainstalovat z těchto stránek `https://nodejs.org/en/download/`. K tomu je nutné mít nainstalovaný balíčkovací manažer npm, který se nainstaluje ve výchozím nastavení s NodeJS.

Prvně je nutné do projektu nainstalovat závislosti, což se dělá pomocí příkazu `npm i` spuštěného v kořenové složce projektu. Ten nainstaluje nejnovější verze závislostí specifikované v souboru *package.json*. Přesná konfigurace verzí se uloží následně do *package-lock.json*, ze které je možné tyto balíčky nainstalovat pomocí příkazu `npm ci`, což se provádí typicky, když chce člověk mít garantovanou přesnou konfiguraci např. při buildu v CI (Continuous Integration).

Po instalaci je možné využívat tyto příkazy:

- `npm start` – spustí vývojový server včetně mock API,

Obrázek 5.1: Struktura projektu.



- `npm run build` – spustí produkční build do složky *dist*,
- `npm run start:no-mock` – spustí vývojový server bez mock API,
- `npm run lint` – spustí linter,
- `npm test` – spustí testy,
- ... a další uvedené v *package.json*.

5.4 Mockování API

Jak již bylo zmíněno, tak jsem byl nucen z důvodů absence klíčových částí API mockovat API. Pro věrnost zážitku jsem se uchýlil k relativně širší implementaci, která umí přihlašování pod různými uživateli s různými uživatelskými právy a jejich kompletní správu (CRUD). Dále pak je namockované seznamy entit *structure* a přiřazených modelů *3D Object* CRUD včetně jejich filtrování a stránkování. Mock API však nikam neukládá zaslané soubory v rámci příloh atd., takže je toto simulováno vrácením statické adresy. API se ale tváří, že se tyto soubory uložily. Mock API neukládá nikam žádná data, proto se po vypnutí mock serveru změněný obsah odstraní.

Pro ukázku přidávám zjednodušený zdrojový kód pro přihlašování a správu uživatelů:

```
import { Request, Response } from 'express';

let users = [/*...*/];
let currentUser; // just an index

export default {
  'POST␣/api/login/account': (req: Request, res: Response) => {
    const { password, userName, type } = req.body;
    for (let i = 0;
        i < users.length && password === 'password';
        i++) {
      if (users[i].userid === userName) {
        currentUser = i;
        res.send({
          status: 'ok',
          type,
          currentAuthority: users[i].roles,
        });
        return;
      }
    }
    res.send({
      status: 'error',
      type,
```

```
      currentAuthority: 'guest',
    });
  },
  'POST /api/users': (req: Request, res: Response) => {
    users.push(req.body);
    res.status(201).send();
  },
  'PUT /api/users/:id': (req: Request, res: Response) => {
    for (let i = 0; i < users.length; i++) {
      if (users[i].userid === req.params.id) {
        users[i] = req.body;
        res.status(204).send();
        return;
      }
    }
    res.status(404).send();
  },
  // ...
}
```

5.5 Komponenty

Základ celé aplikace tkví v React komponentách. Ty se starají o zobrazování dat, zpracovávání událostí a volání služeb, akcí atd. Jejich základní rozdělení je na prezenční komponenty a kontejnery.

5.5.1 Prezentační komponenty

Prezenční komponenty se starají o to, jak věci vypadají a o vykreslování předaných dat. Jsou často psány jako funkcionální komponenty, tzn. nejdedná se o třídu, ale funkci, která bere parametry a vrací pseudo DOM komponenty. Mohou obsahovat další prezentační nebo kontejnerové komponenty, ale nejsou závislé na dalších částech aplikace, jakou jsou služby. Nestarají se o to, jak jsou data načítána ani měněna, případné události od uživatele posílají dále nadřazeným komponentám.[37]

Ukázka takové komponenty je uvedena níže:

```
const Variant = ({ variant, hideTitle }) => {
  const title = variantMapping[variant]?.title || variant;
  const icon = variantMapping[variant]?.icon;
  return hideTitle ? (
    <Tooltip title={title}>
      <VariantIcon icon={icon} />
    </Tooltip>
  ) : (
    <Text>
      {title} <VariantIcon icon={icon} />
```

```
    </Text>
  );
};
```

5.5.2 Kontejnerové komponenty

Kontejnerové komponenty se spíše zajímají o to, jak věci fungují, a skládají dohromady prezentační a další kontejnerové komponenty do větších celků, kterým poskytují data a chování (např. zpracovávání událostí). Mají často stav a volají Redux akce a jsou často generovány přes komponenty vyššího řádu jako např. connect(). I díky novému Hooks API v Reactu je ale možné je rovněž psát často jako funkcionální komponenty. To zvyšuje čitelnost dané komponenty. Toto nové API umožňuje v rámci funkcionální komponenty používat state (vnitřní stav) či navázat nějaké chování na životní cyklus aplikace.[37]

Ukázka takové komponenty je k dispozici níže:

```
const ArtefactEdit = (props) => {
  const [structure, setStructure] = useState(null);
  const fetchDetail = () =>
    querySingleArtefact(props.match.params.id).then((data) => {
      setStructure(data);
    });
  const handleSubmit = (data: Artefact): Promise<void> => {
    editArtefact(props.match.params.id, data)
      .then((editedArtefact) => {
        history.push(`/artefacts/detail/${editedArtefact.id}`);
        message.success(`Ulozeno!`);
      });
  };
  return (
    <PageLoader
      title="Editace artefaktu"
      breadcrumb={{
        routes: [
          ...routes,
          {
            breadcrumbName: 'Editovat artefakt',
          },
        ],
      }}
      loader={() => fetchDetail()}
    >
      {structure && (
        <Card>
          <Title level={2} style={{ textAlign: 'center' }}>
            Editovat artefakt {structure.name}
          </Title>
          <ArtefactForm
```

```
            onSubmit={handleSubmit}
            values={structure}
          />
          </Card>
        )}
      </PageLoader>
    );
};
```

5.6 Modely

Modely slouží pro state management aplikace pomocí knihovny Redux. Definují danou část store včetně typu, výchozího stavu, reducerů, které vytvářejí nový stav na základě dané akce a výchozího stavu a efektů, které se starají o asynchronní akce. Níže je například ořezaný model nastavení:

```
interface SettingModelType {
  namespace: 'settings';
  state: DefaultSettings;
  reducers: {
    changeSetting: Reducer<DefaultSettings>;
  };
}

const SettingModel: SettingModelType = {
  namespace: 'settings',
  state: defaultSettings,
  reducers: {
    changeSetting(state = defaultSettings, { payload }) {
      const { colorWeak, contentWidth } = payload;

      if (
        state.contentWidth !== contentWidth
        && window.dispatchEvent
      ) {
        window.dispatchEvent(new Event('resize'));
      }
      return {
        ...state,
        ...payload,
      };
    },
  },
};
```

5.7 Služby

Do služeb jsou extrahované funkční celky, které obstarávají nějaký separátní kus funkcionality, v případě administračního rozhraní je v nich extrahovaná veškerá komunikace s API. Jsou zapsané v podobě čistých TypeScript funkcí, níže je zobrazena část služby pro správu uživatelů:

```
export async function createUser(user: User) {
  return request('/api/users', {
    method: 'POST',
    data: {
      ...user,
    },
  });
}
export async function deleteUser(id) {
  return request(`/api/users/${id}`, {
    method: 'DELETE',
  });
}
// ...
```

5.8 Vizualizace 3D modelu

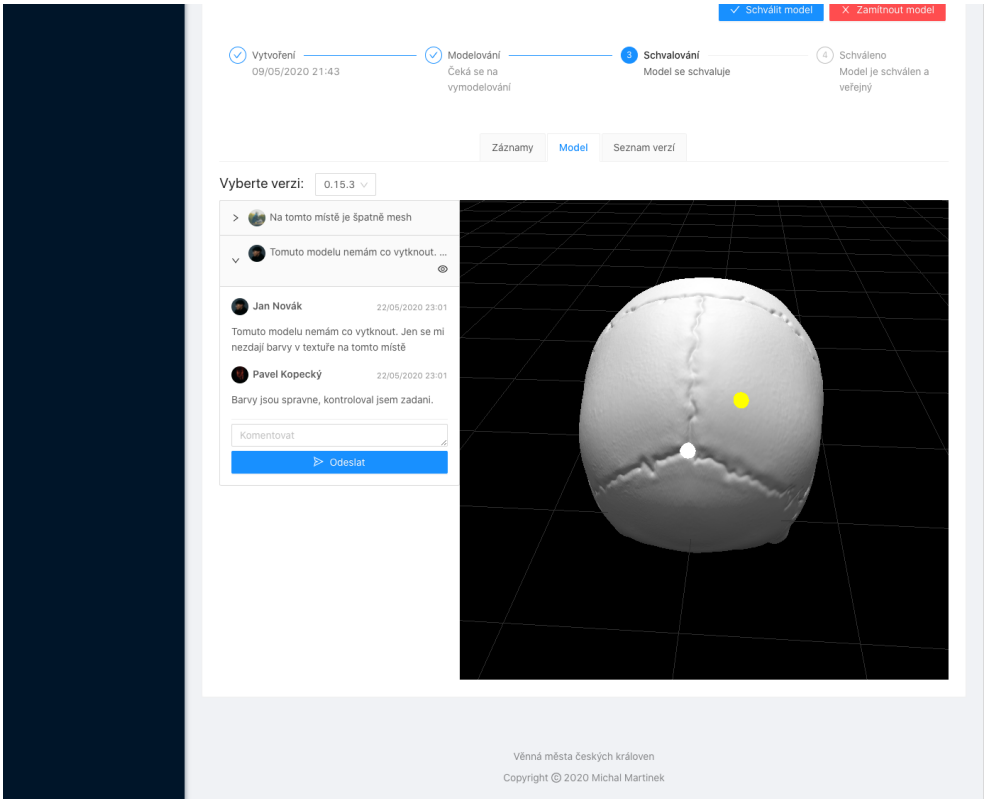
Jak bylo rozhodnuto v analýze v kapitole 2.7.2.4, tak pro vizualizaci 3D modelů byla použita knihovna Three.js[27]. Ta má předchystanou funkcionalitu pro nastavení scény, kamery, světel, načtení 3D modelu v různých formátech. Mimo jiné podporuje i rozpoznání protnutí kliku do vizualizace s objekty v dané scéně. K tomu má i podporu TypeScript, což se pro použití v tomto projektu hodí. Tato knihovna je vysokoúrovňová, takže programování probíhá docela deklarativním způsobem.

Zadrátování do React aplikace jsem provedl tím způsobem, že existuje kontejnerová komponenta, která obaluje celou vizualizaci a vazby na ostatní komponenty, které se samotnou vizualizací komunikují. V té se na startu komponenty inicializuje WebGL renderování do elementu pomocí reference. Poté se připraví scéna v rámci Three.js, která obsahuje celou vizualizaci, a nastaví se ovládání, kamery, osvětlení modelu, načtení modelu, zvýraznění označených částí a obslužné rutiny událostí.

Na obrázku 5.2 lze vidět výslednou variantu vizualizace společně s komentováním míst modelu, které je možné provést prostým kliknutím do modelu.

5.9 Implementovaná část

Kvůli rozsáhlosti administračního rozhraní a absence adekvátního API, které jsem musel řešit mockováním, jsem nenaimplementoval administrační rozhraní



Obrázek 5.2: Interaktivní komentování modelu.

v celé jeho šíři v podobě, kterou jsem rozvedl v sekci 3.3. Naimplementoval jsem kostru aplikace tak, aby tento základ administračního rozhraní obsahoval veškerou nutnou funkcionalitu pro fungování základní verze.

Konkrétně to znamená, že je naimplementovaná správa uživatelů, přihlašování, zobrazování veškerých entit *structure*, včetně jejich filtrování a správy – vytváření, editace, mazání. Dále pak zobrazení detailu těchto entit včetně jejich vytvořených modelů, ke kterým je možné zobrazit průběh schvalování, seznam jejich verzí i zobrazení samotného 3D modelu včetně možnosti komentování bodů v tomto modelu, rovněž ke schvalování přiřazovat zodpovědné schvalovatele a modeláře, kteří tento model mohou schvalovat.

Co v implementaci chybí je správa vlastního uživatelského profilu pro lidi bez role *Admin*. Dále pak Dashboard a seznamy schvalování a modelování s filtracemi dle aktuálního stavu, toto je část, která nabízí výrazné ulehčení, ale není nutná pro schvalování modelů a manipulaci s entitami *Structure* a je dosti závislá na API. A poslední chybějící částí je správa sekvencí transformací a generování pomocí jejich použití. Podstatnou složku budou tvořit rovněž notifikace o novinkách na na přiřazených schvalováních a modelováních, které jsou již v administračním rozhraní předchystané. Z mého pohledu je naimplemen-

tovaná část administračního rozhraní dobrý základ, který bude potřeba ještě dotáhnout v jistých aspektech jako např. responzivnosti, ale veškeré hlavní části funkcionality mají prvotní řešení.

Snímky implementace jsou k dipsozici v příloze D.

Závěr

V této práci jsem se zabýval vývojem webového administračního rozhraní pro projekt Věnných měst českých královen, což je projekt zaměřený na prezentaci historie věnných měst českých královen, jejíž podstatnou částí má být vizualizace zrekonstruovaných 3D modelů, které mají být spravované a schvalované v tomto administračním rozhraní.

Nejprve jsem začal sběrem a analýzou funkčních a nefunkčních požadavků a rovněž i případy užití. Pak jsem zkoumal a porovnával použití 2 existujících řešení pro schvalovací účely. Dále jsem se pak věnoval výběru frontend technologie pro vývoj tohoto administračního rozhraní pomocí vícekriteriální analýzy variant i výběru dalších pomocných knihoven.

Po analýze jsem rozebral úpravy v návrhu API, na nichž jsem se podílel v podobě kontraktu mezi klientem a serverem. Následně jsem se věnoval návrhu samotného schvalovacího procesu modelů včetně připomínkovávání, vracení k přepracování atd. Na základě analýzy jsem následně přistoupil k návrhu uživatelského rozhraní. Začal jsem sestavením seznamu úkolů a jejich grafu vzájemné provázanosti. Po tomto kroku jsem se věnoval prototypu v podobě wireframů, na jejichž základě jsem vytvořil prototyp vysoké věrnosti v nástroji Adobe XD.

Prototyp uživatelského rozhraní jsem nechal prověřit drobným kvalitativním uživatelským testováním, které jsem sám prováděl se 4 testery. Toto testování poukázalo na několik problematických míst v uživatelském návrhu a rovněž na jistá omezení prototypovacího nástroje Adobe XD. Problematická místa jsem v návrhu aplikace analyzoval a opravil.

Na základě tohoto otestovaného uživatelského rozhraní jsem přistoupil k implementaci v technologiích, které byly vybrány v analýze – pomocí knihoven React, Redux a Three.js. Implementace ale měla jistá úskalí, administrační rozhraní je relativně rozsáhlé a kvůli absenci adekvátního API jsem musel toto API celé mockovat. Proto jsem v rámci implementace stihl vyrobit funkční základ tohoto administračního rozhraní obsahující správu entit *Structure* a jejich jednotlivých modelů, schvalování těchto modelů včetně připomínkování,

interaktivního prohlížení modelu a označování míst přímo v tomto modelu a jejich komentování.

Povedlo se mi splnit cíle kladené na tuto práci a provedl jsem úspěšně analýzu, návrh, uživatelské testování návrhu rozhraní a základ implementace tohoto administračního rozhraní, ten obsahuje veškerou nutnou funkcionalitu, ale před jeho spuštěním bude nutné dotáhnout jisté chybějící části a aspekty pro dobrou použitelnost a rovněž jej napojit na reálné API, až bude k dispozici.

Literatura

[1] *Věnná města českých královen [online].* [cit. 2020-04-10]. Dostupné z: https://www.kralovskavennamesta.cz/about.html

[2] Karl, W.; Joy, B.: *Software Requirements,Third Edition.* Microsoft Press, 2013, ISBN 978-0-7356-7966-5, 619 s.

[3] Bittner, K.; Spence, I.: *Use Case Modeling.* Addison-Wesley Professional, 2003, ISBN 978-0201709131, 347 s.

[4] *ELO Java Client Workflow.* ELO Digital Office GmbH, 2017, 165 s.

[5] Mulholland, B.: *The Consultant's Guide to Process Street is Here! [online].* [cit. 2020-04-15]. Dostupné z: https://www.process.st/launch-consultants-guide-to-process-street/

[6] *Process Street: Full Review [online].* Keep Productive, [cit. 2020-04-15]. Dostupné z: https://www.youtube.com/watch?v=MiiM4rLSChM

[7] *Frameworkless Movement [online].* Frameworkless Movement, [cit. 2020-05-20]. Dostupné z: http://frameworklessmovement.org/

[8] *Can I use web components? [online].* Can I use..., [cit. 2020-03-13]. Dostupné z: https://caniuse.com/#search=web%20components

[9] Overson, J.; Strimpel, J.: *Developing Web Components.* O'Reilly Media, Inc., 2015, ISBN 978-1-491-94902-3, 252 s.

[10] *React [online].* Facebook Inc., [cit. 2020-05-20]. Dostupné z: https://reactjs.org/

[11] bin Uzayr, S.; Cloud, N.; Ambler, T.: *JavaScript Frameworks for Modern Web Development.* apress., 2019, ISBN 978-1-4842-4995-6, 555 s.

[12] *Angular [online].* Google, [cit. 2020-05-20]. Dostupné z: https://angular.io/

[13] You, E.: *Vue.js [online].* [cit. 2020-05-20]. Dostupné z: https://vuejs.org/

[14] *The State of JavaScript 2019 [online].* State of JavaScript, [cit. 2020-03-14]. Dostupné z: https://2019.stateofjs.com/

[15] *Ember.js [online].* Tilde Inc., [cit. 2020-05-20]. Dostupné z: hhttps://emberjs.com/

[16] Strazzullo, F.: *Frameworkless Front-End Development.* apress, 2019, ISBN 978-1-4842-4967-3, 257 s.

[17] *MDN web docs [online].* Mozilla and individual contributors, [cit. 2020-04-20]. Dostupné z: https://developer.mozilla.org/en-US/

[18] Shan, P.: *Angular vs React vs Ember vs Vue – JS framework comparison [online].* Void Canvas, [cit. 2020-03-14]. Dostupné z: http://voidcanvas.com/angular-vs-react-vs-ember-vs-vue-js/

[19] Sharma, N.: *React vs Angular vs Ember vs Vue: Which Is the Best JavaScript Framework? [online].* Apptunix, [cit. 2020-03-15]. Dostupné z: https://www.apptunix.com/blog/react-vs-angular-vs-ember-vs-vue-best-javascript-framework/

[20] Fotr, J.; Švecová, L.: *Manažerské rozhodování - Postupy, metody a nástroje.* Ekopress, 2016, ISBN 978-80-87865-33-0.

[21] *Flux vs Redux : What Differentiate Them from Each Other [online].* Aglowid IT Solutions, [cit. 2020-03-15]. Dostupné z: https://yourstory.com/mystory/flux-vs-redux

[22] *Flux [online].* Facebook Inc., [cit. 2020-05-20]. Dostupné z: https://facebook.github.io/flux/

[23] *Redux - A Predictable State Container for JS Apps [online].* Dan Abramov and the Redux documentation authors, [cit. 2020-05-20]. Dostupné z: https://redux.js.org/

[24] *MobX [online].* [cit. 2020-05-20]. Dostupné z: https://mobx.js.org/README.html

[25] Bilák, T.: *Možnosti využití frameworku A-Frame pro rozšířenou realitu.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[26] *WebGL [online].* Khronos Group., [cit. 2020-05-20]. Dostupné z: https://www.khronos.org/webgl/wiki/Main_Page

[27] *Three.js [online].* [cit. 2020-04-16]. Dostupné z: https://threejs.org/

[28] *A-Frame [online].* [cit. 2020-05-20]. Dostupné z: https://aframe.io/

[29] Fanie, R.: *Modern API Design with ASP.NET Core 2: Building Cross-Platform Back-End Systems.* apress., 2018, ISBN 978-1-4842-3518-8, 245 s.

[30] Sůvová, D.: *Věnná města českých královen I. - úprava textur.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[31] Zajíc, M.: *Věnná města českých královen I. - úprava textur.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[32] Wolker, M.; Takayama, L.; Landay, J. A.: High-fidelity or low-fidelity, paper or computer? Choosing attributes when testing web prototypes. *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting*, 2002.

[33] Krug, S.: *Nenuťte uživatele přemýšlet.* Computer press, a.s., 2010, ISBN 978-80-251-2923-4, 165 s.

[34] Freeman, A.: *Essential TypeScript: From Beginner to Pro.* apress., 2019, ISBN 978-1-4842-4978-9, 540 s.

[35] *Ant Design [online].* XTech, [cit. 2020-05-20]. Dostupné z: https://ant.design/

[36] Mangin, A.: *How to better organize your React applications? [online].* Medium, duben 2016, [cit. 2018-04-24]. Dostupné z: https://medium.com/@alexmngn/how-to-better-organize-your-react-applications-2fd3ea1920f1

[37] Abramov, D.: *Presentational and Container Components [online].* Medium, březen 2015, [cit. 2018-04-20]. Dostupné z: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Příloha A

Seznam použitých zkratek

API Application Programming Interface

AST Abstract Syntax Tree

CI Continuous Integration

CLI Command Line Interface

CRUD Create, Read, Update, Delete

CSRF Cross-Site Request Forgery

CSS Cascading Style Sheets

DOM Document Object Model

ECM Enterprise Content Management

GPU Graphics processing unit

Hi-Fi High-Fidelity

JSX JavaScript XML

Lo-Fi Low-Fidelity

npm Node Package Manager

PWA Progressive Web Apps

SEO Search Engine Optimalization

SO Stack Overflow

SPA Single-Page Application

VMCK Venná města českých královen

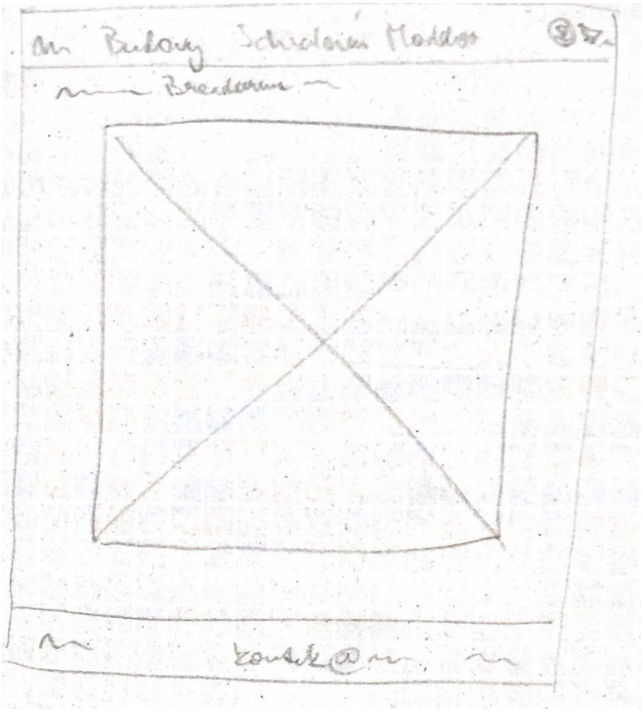
VR Virtual reality

XXS Cross-site scripting

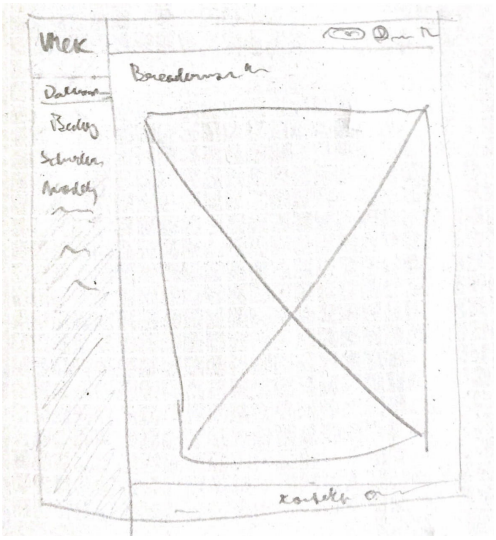
YAML YAML Ain't Markup Language

Příloha **B**

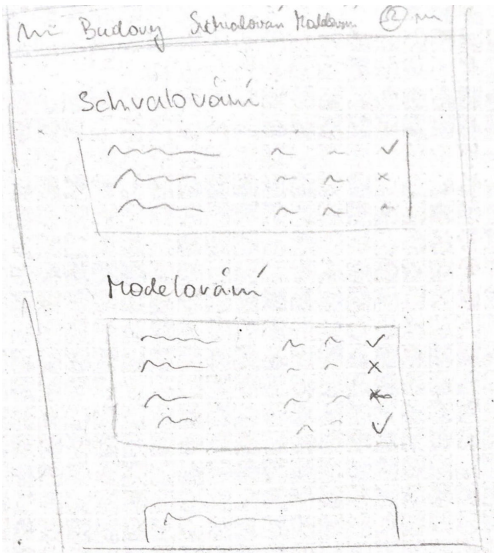
Wireframy (LoFi)



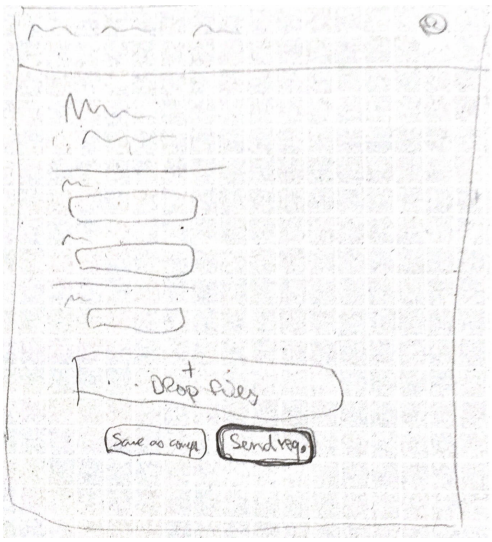
Obrázek B.1: Základní layout I.



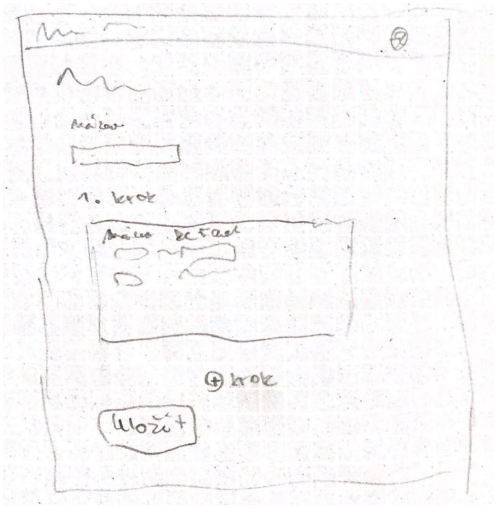
Obrázek B.2: Základní layout II.



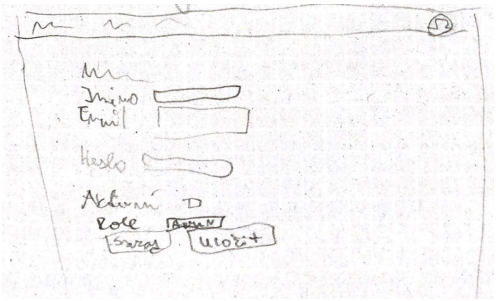
Obrázek B.3: Dashboard.



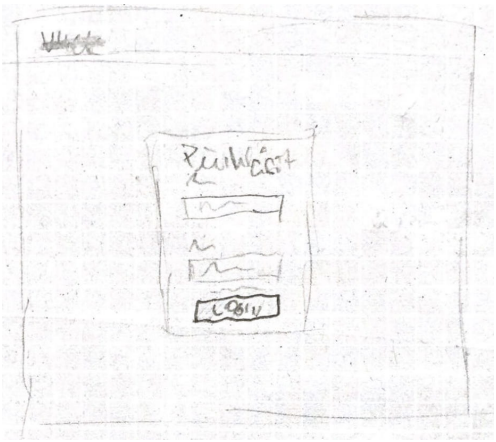
Obrázek B.4: Vytvoření/editace *structure*.



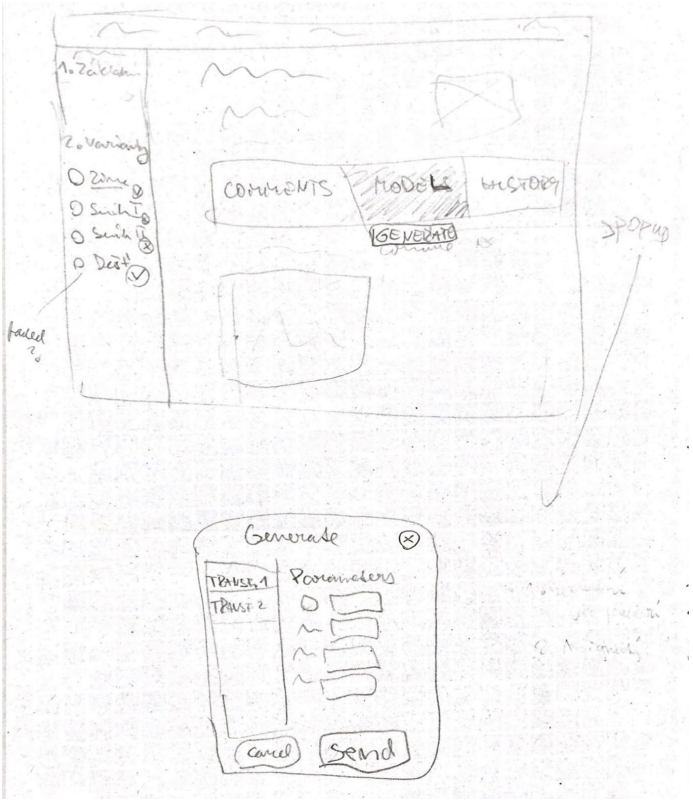
Obrázek B.5: Vytvoření/editace sekvence tranformací.



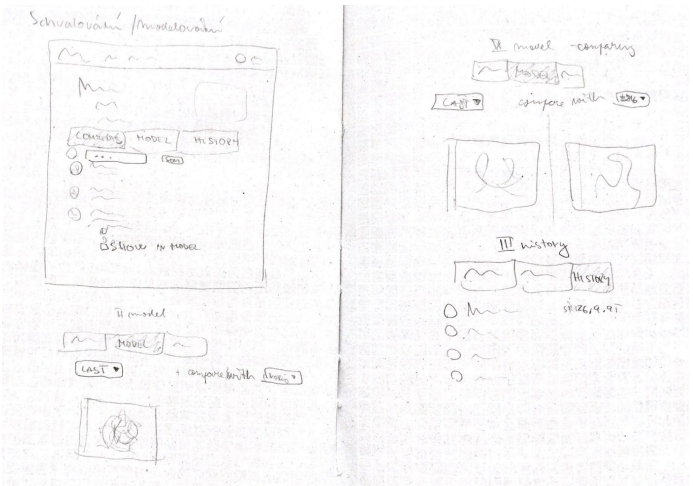
Obrázek B.6: Vytvoření/editace uživatele.



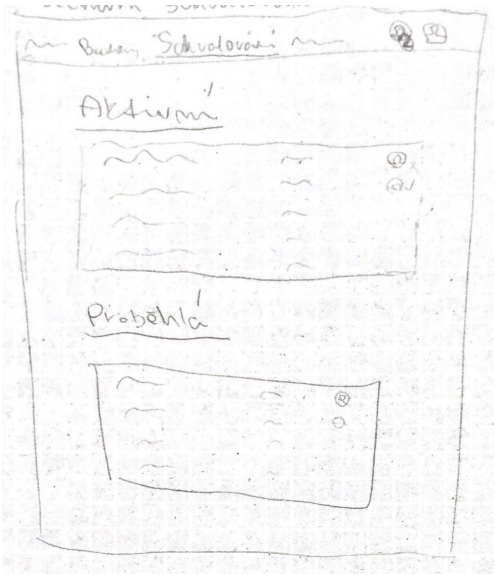
Obrázek B.7: Přihlašování.



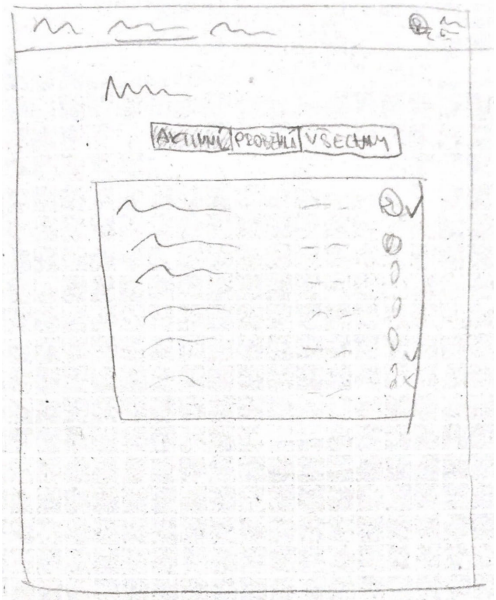
Obrázek B.8: Schvalování variant.



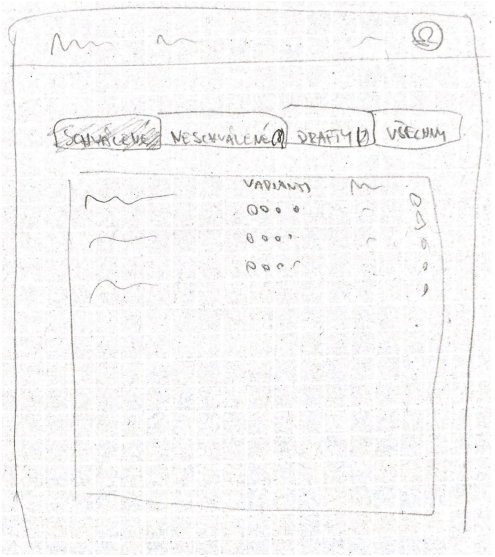
Obrázek B.9: Schvalování/modelování.



Obrázek B.10: Seznam schvalování/modelování I.

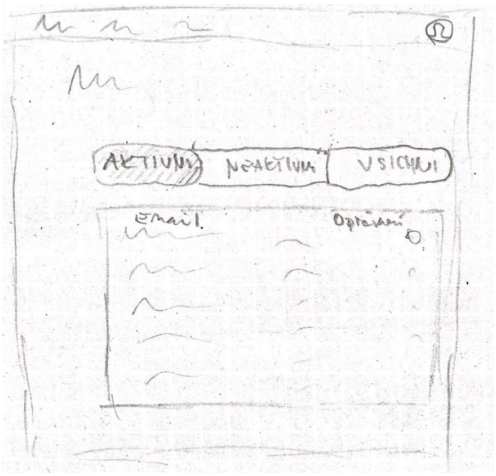


Obrázek B.11: Seznam schvalování/modelování II.

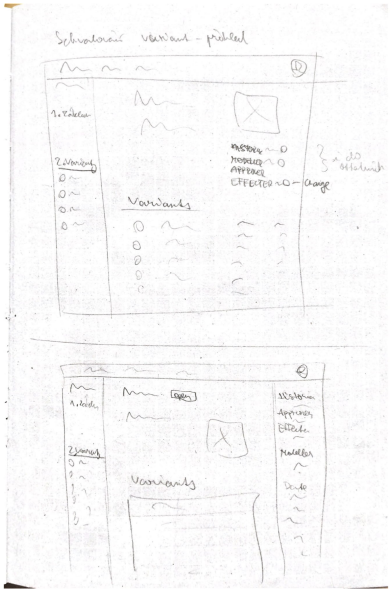


Obrázek B.12: Seznam *structure*.

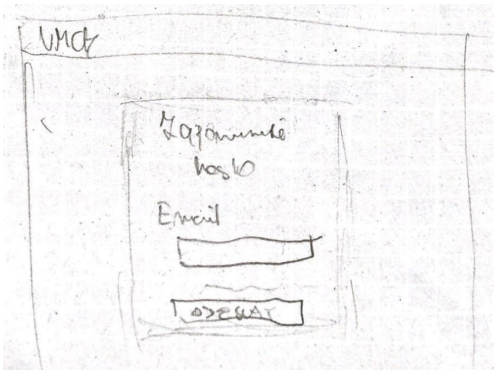
Obrázek B.13: Seznam sekvencí tranformací.



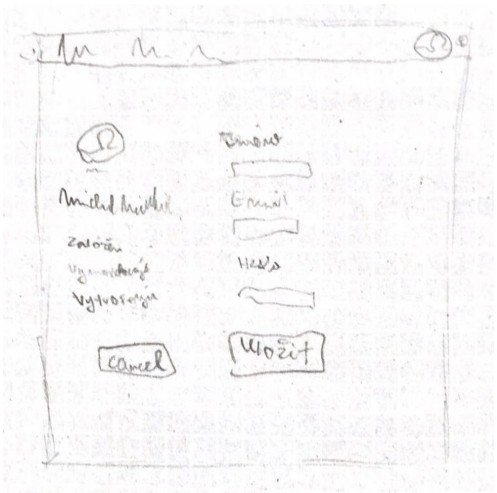
Obrázek B.14: Seznam uživatelů.



Obrázek B.15: Seznam variant *structure*.



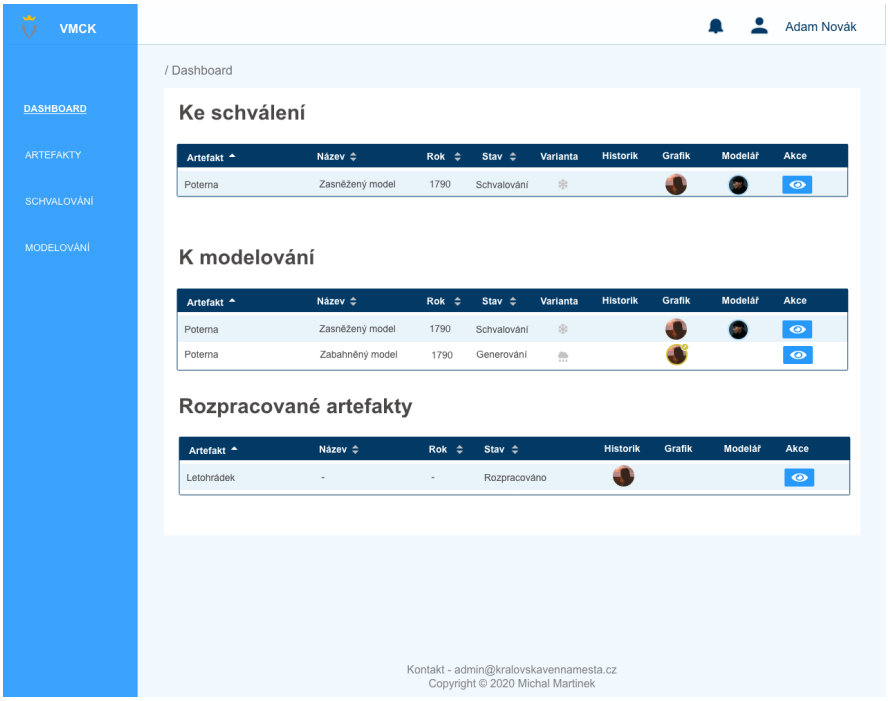
Obrázek B.16: Zapomenuté heslo.



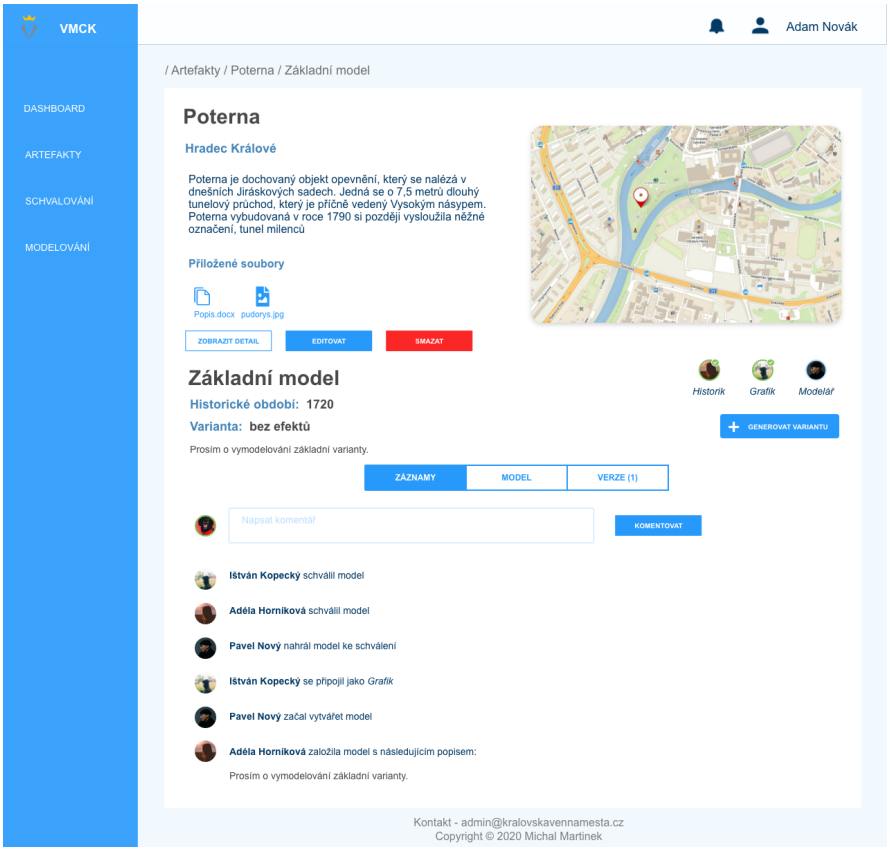
Obrázek B.17: Změna profilu včetně změny hesla.

Příloha C

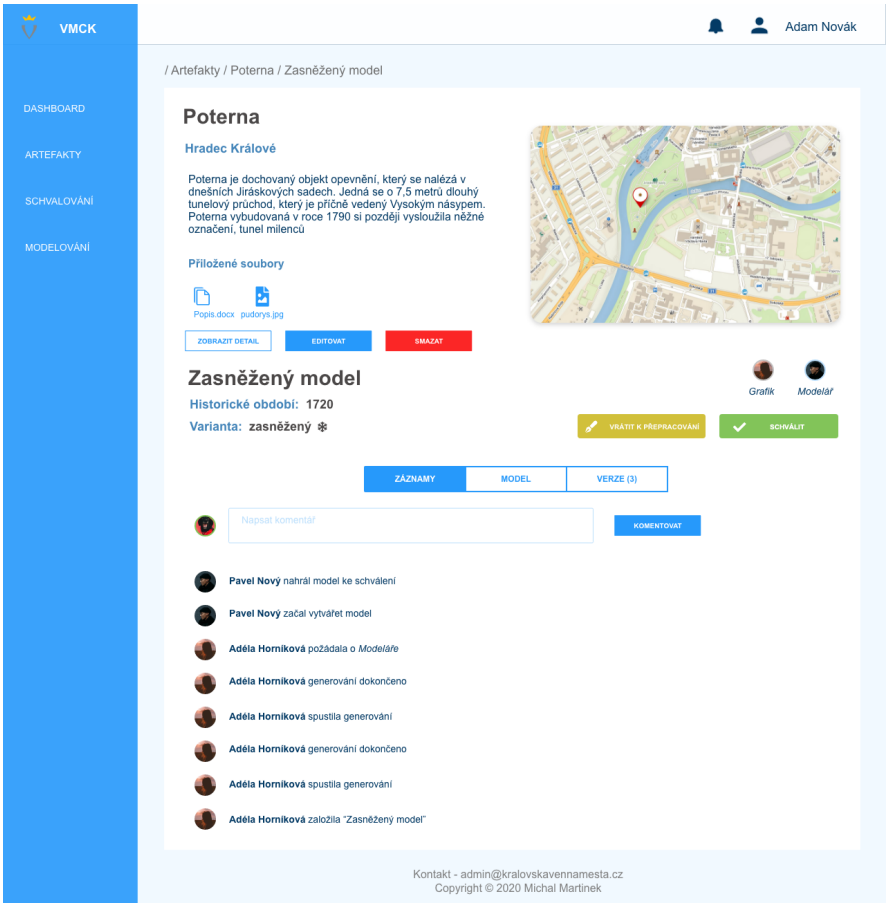
Snímky Wi-Fi prototypu



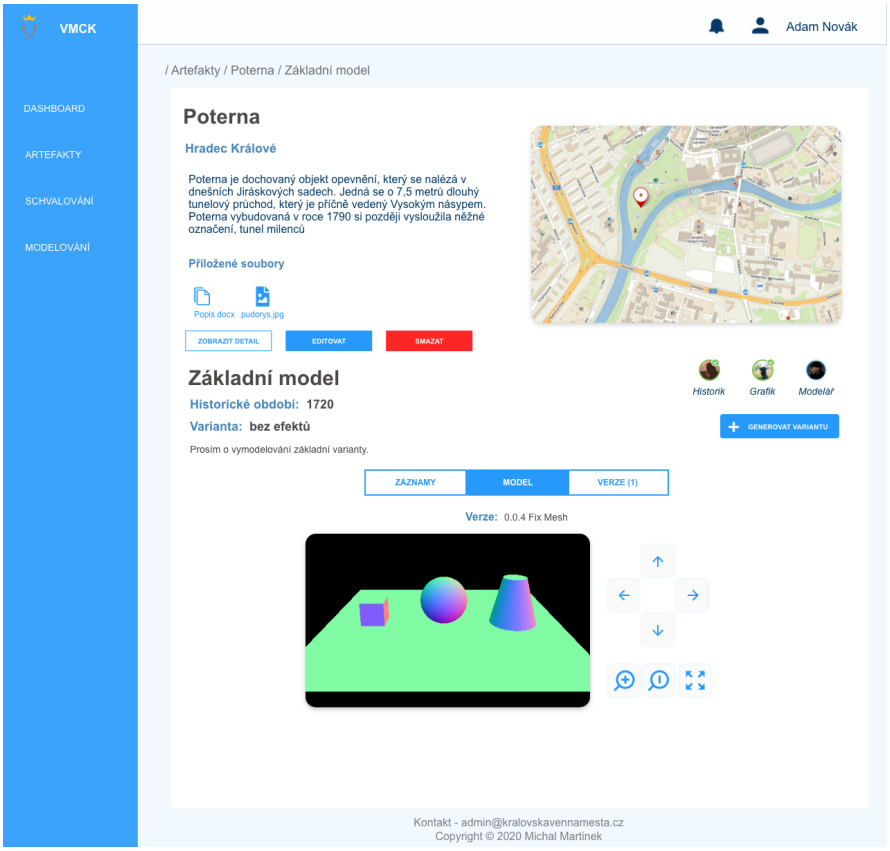
Obrázek C.1: Dashboard.



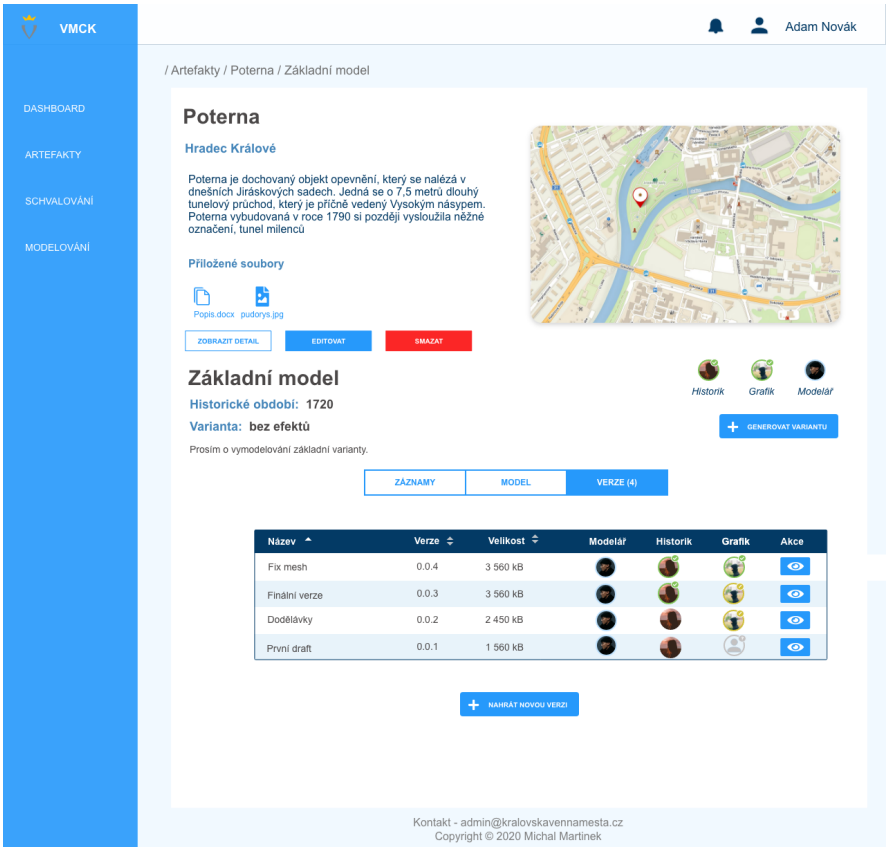
Obrázek C.2: Schválený model.



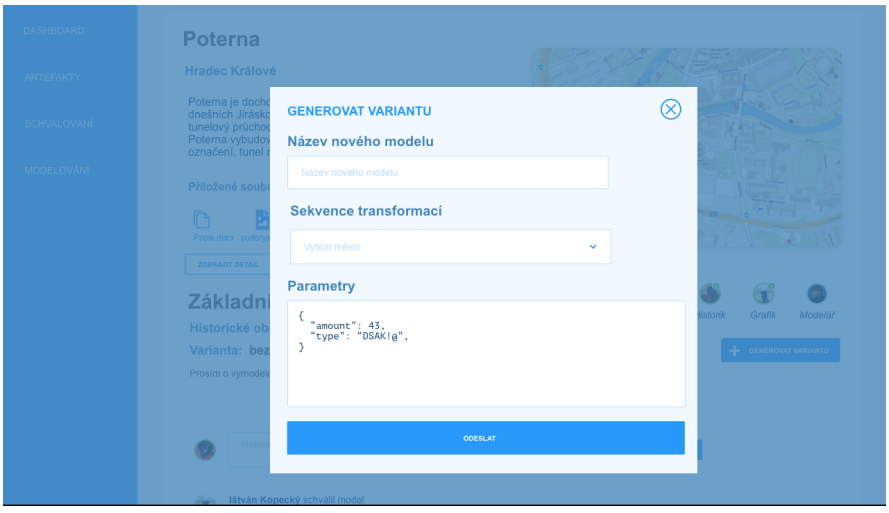
Obrázek C.3: Schvalování modelu.



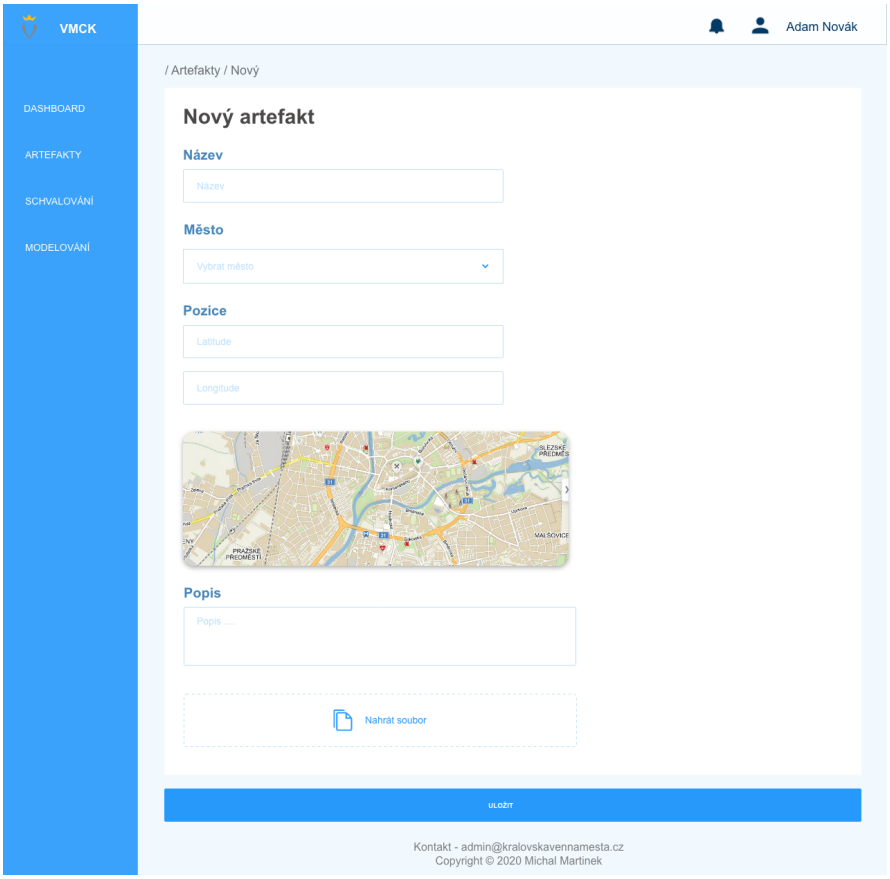
Obrázek C.4: Prohlížení modelu.



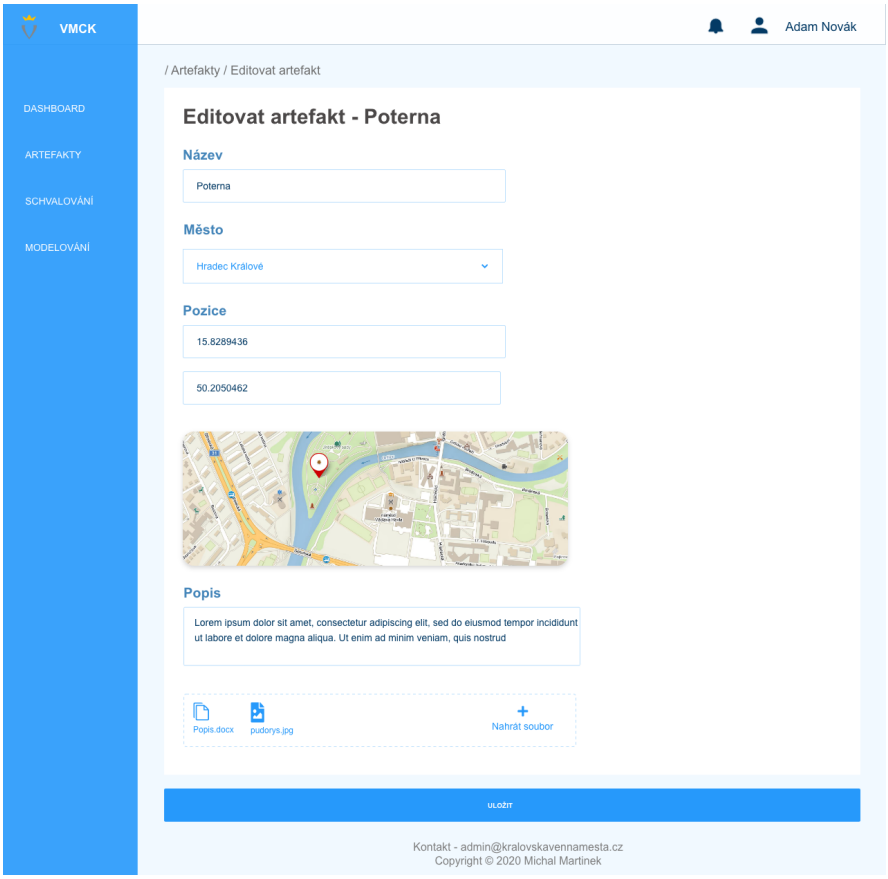
Obrázek C.5: Historie verzí modelu.



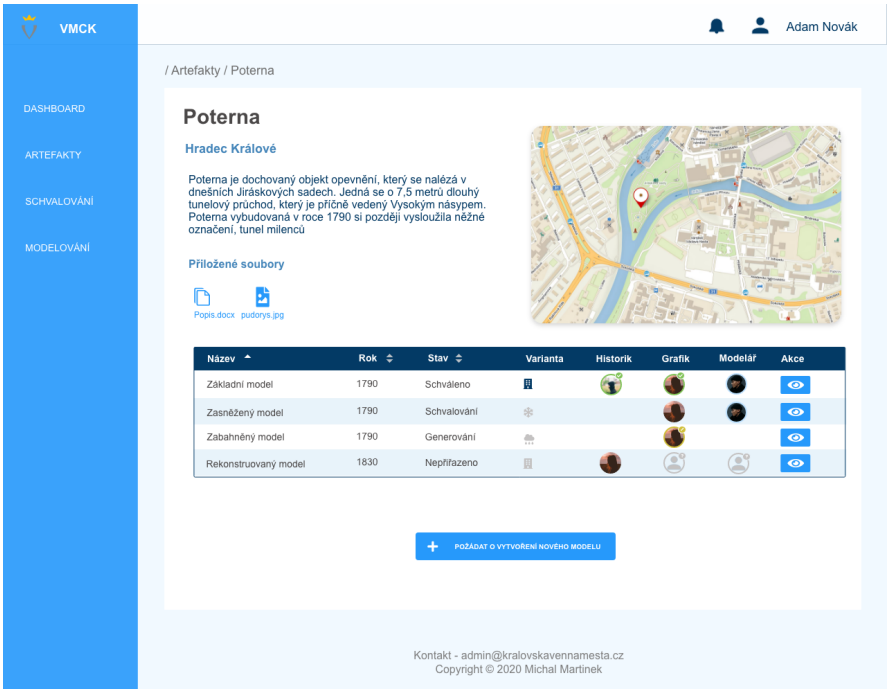
Obrázek C.6: Generování varianty.



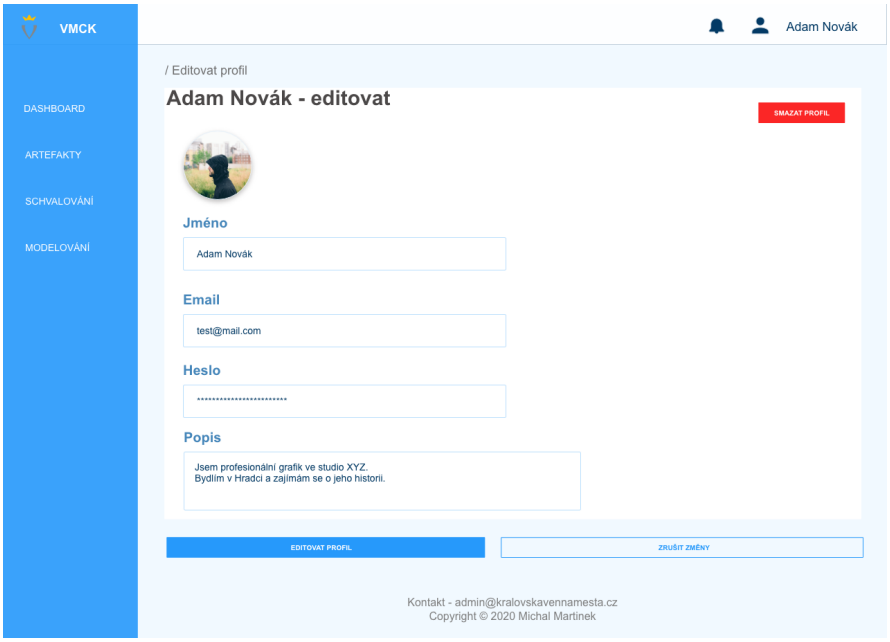
Obrázek C.7: Nový *structure*.



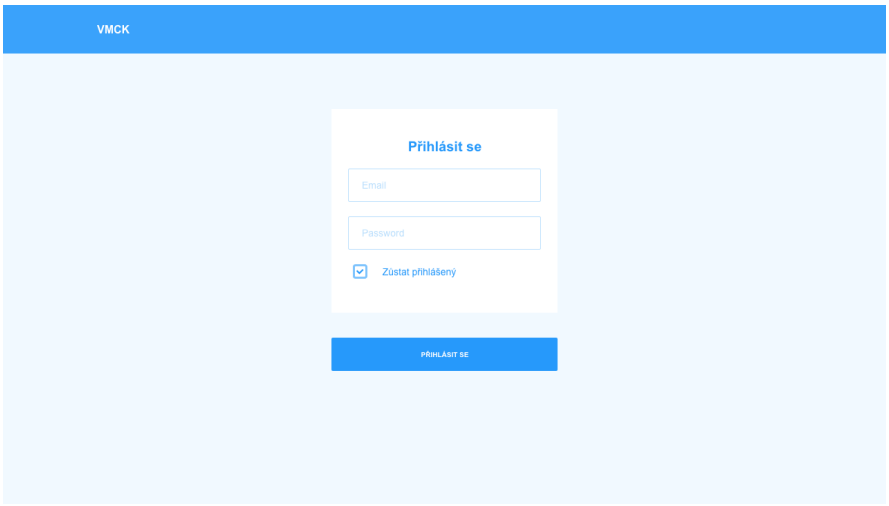
Obrázek C.8: Editace *structure*.



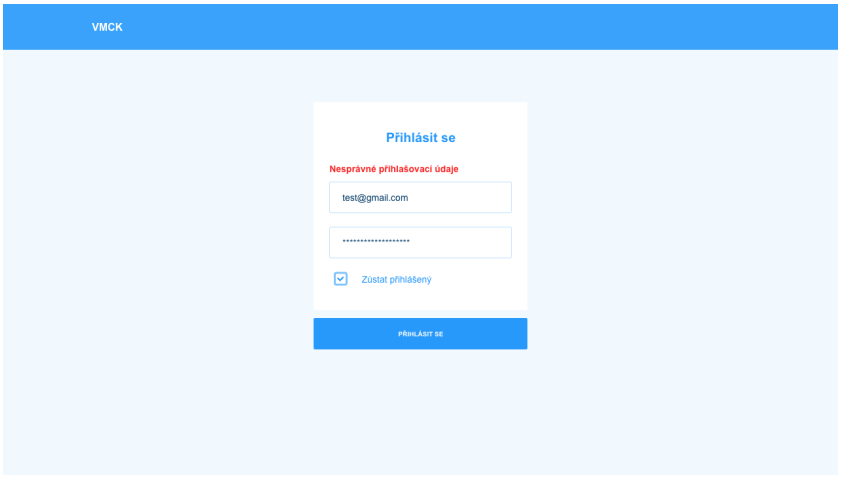
Obrázek C.9: Detail *structure*.



Obrázek C.10: Editovat profil.



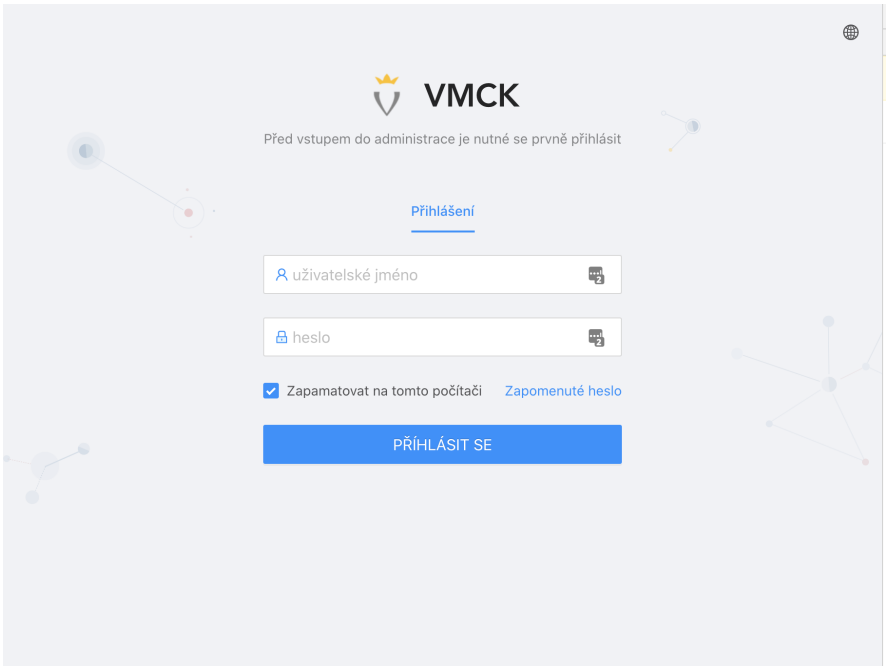
Obrázek C.11: Přihlášení.



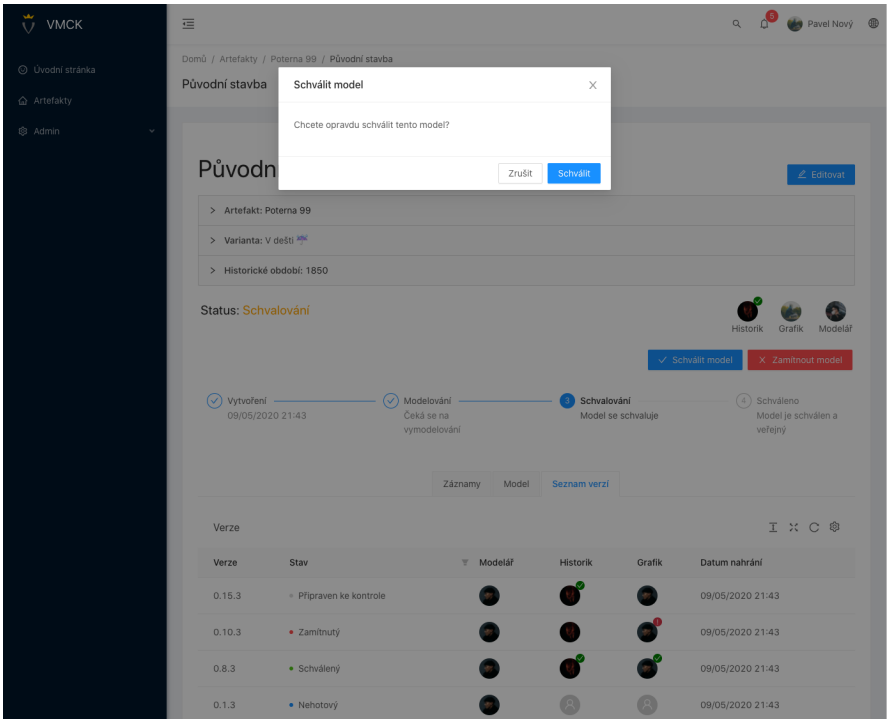
Obrázek C.12: Přihlášení – chybová hláška.

Příloha D

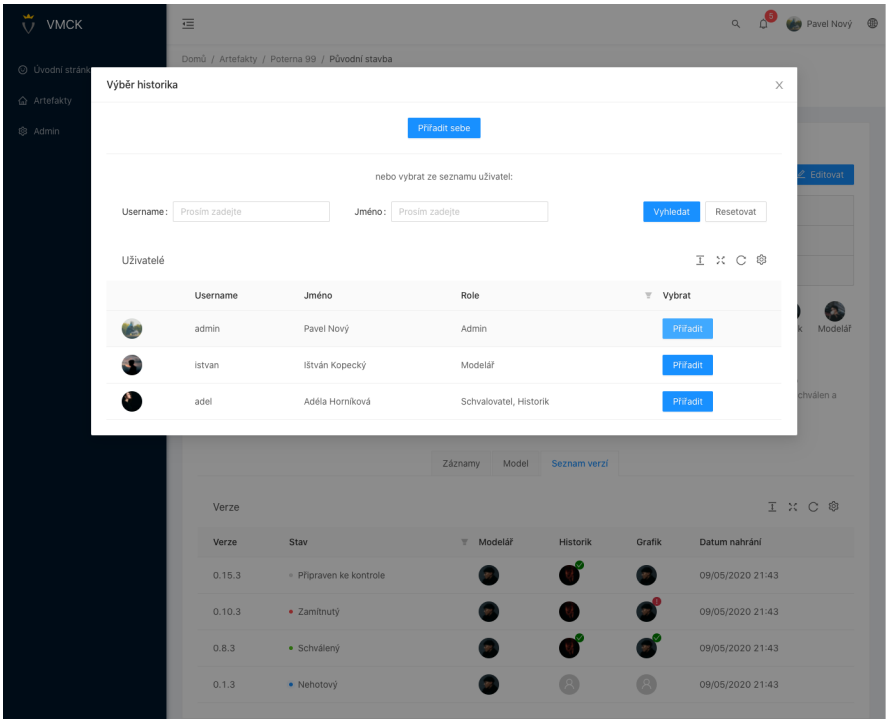
Snímky obrazovky aplikace



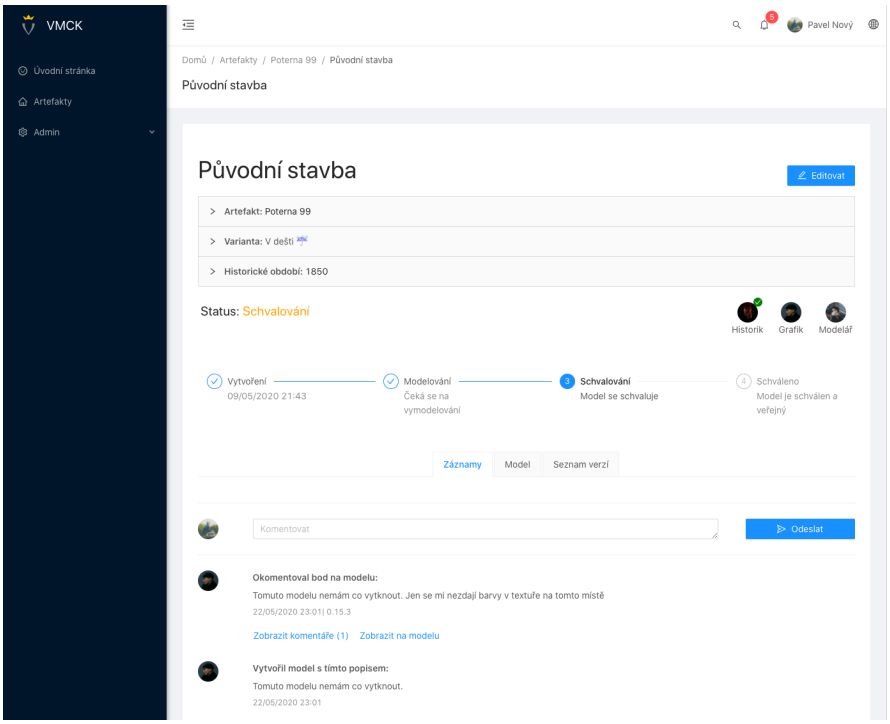
Obrázek D.1: Přihlášení.



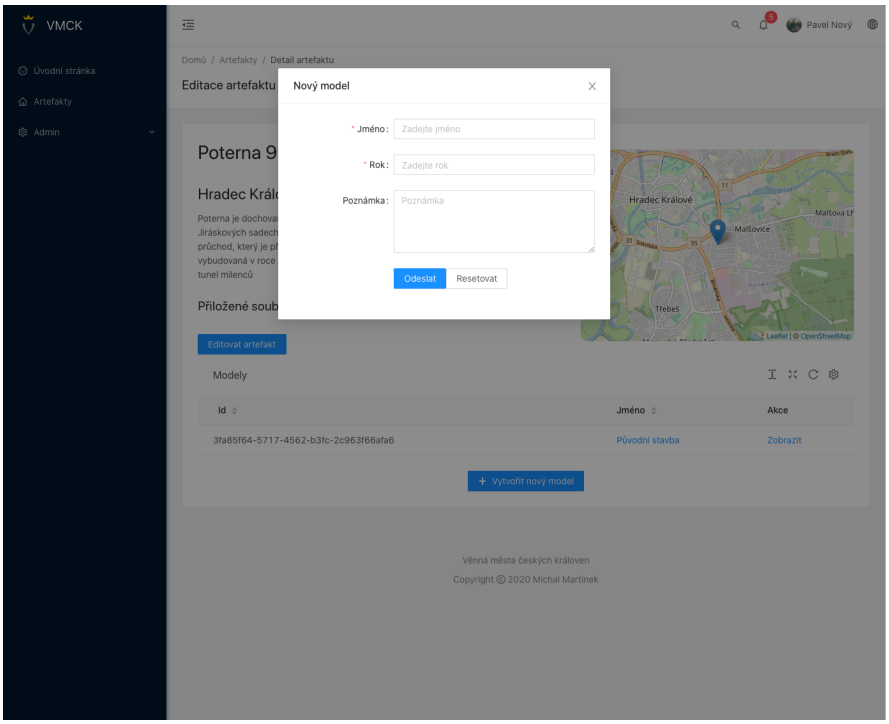
Obrázek D.2: Schvalování modelu.



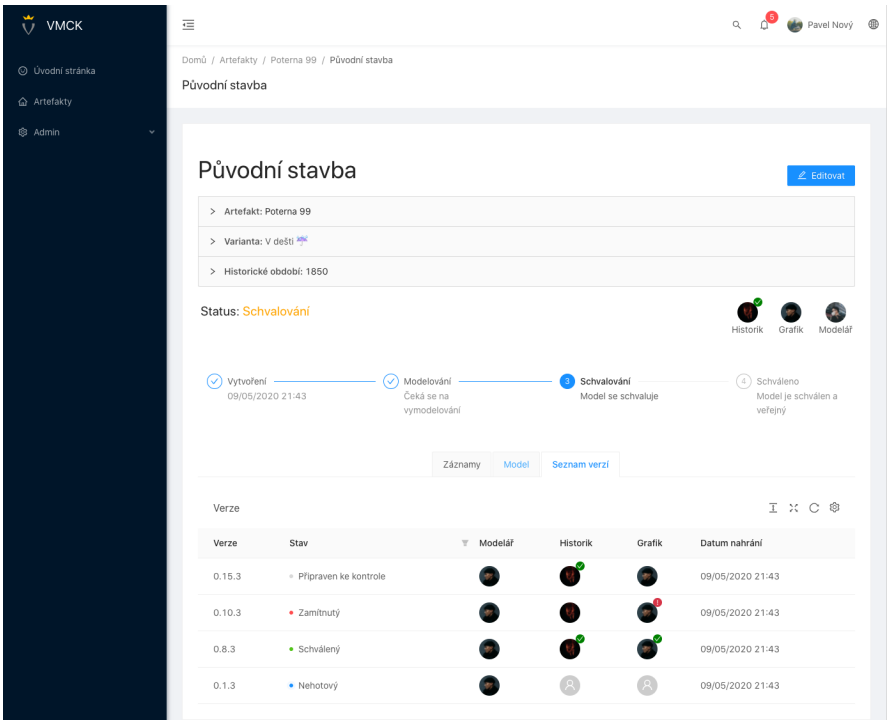
Obrázek D.3: Přiřazení uživatele k modelování.



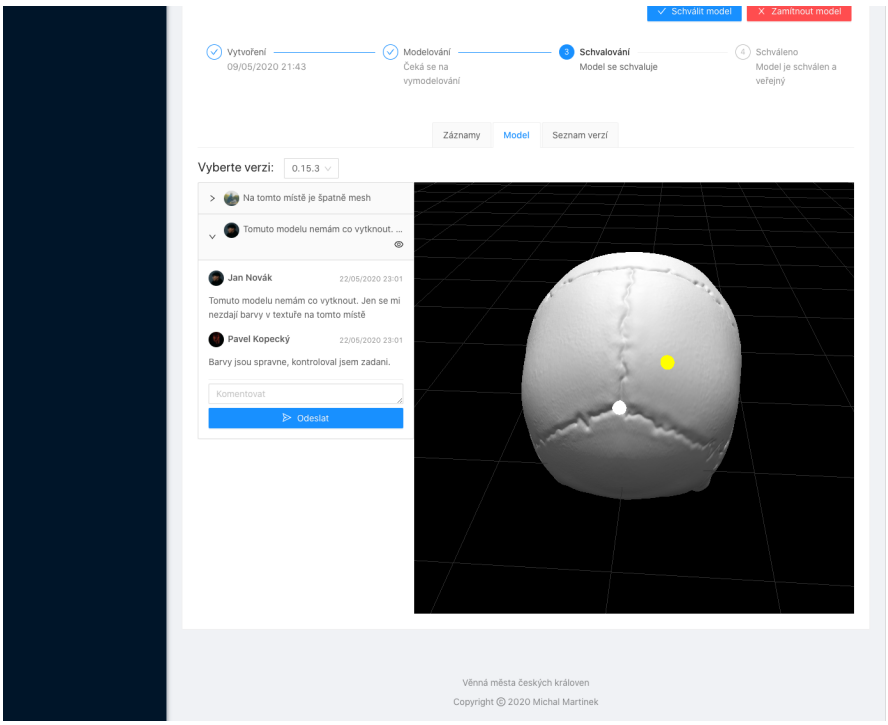
Obrázek D.4: Seznam akcí u *3D object*.



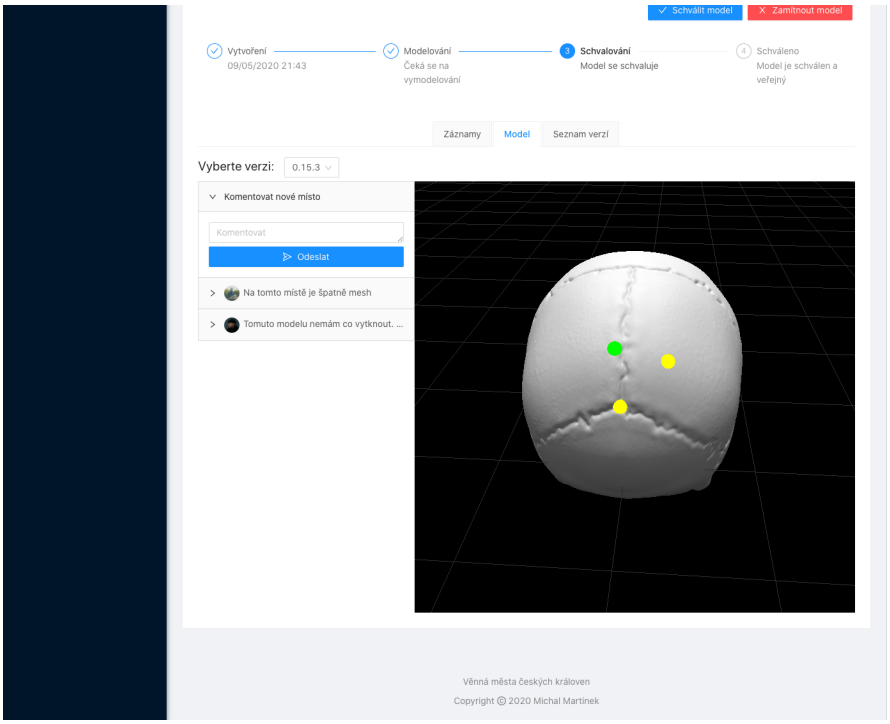
Obrázek D.5: Formulář vytvoření nového *3D object*.



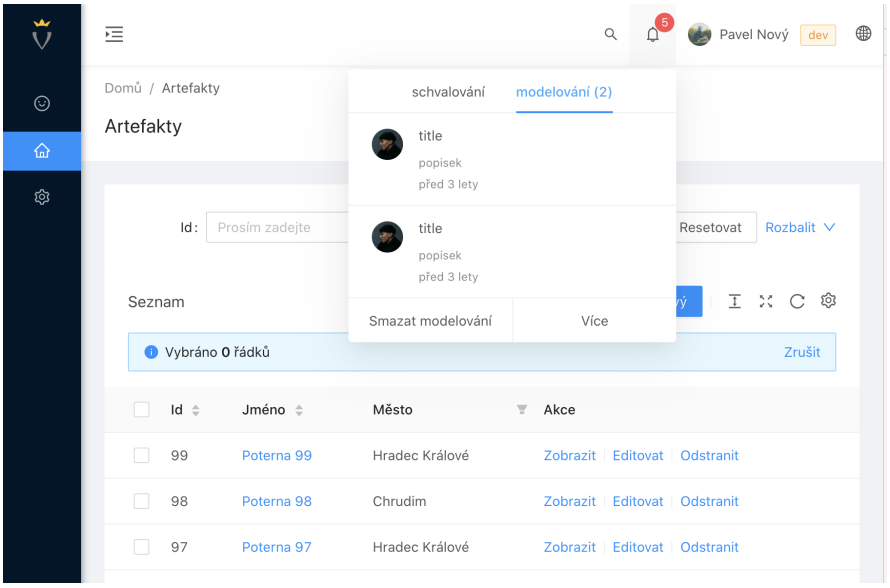
Obrázek D.6: Seznam verzí *3D object*.



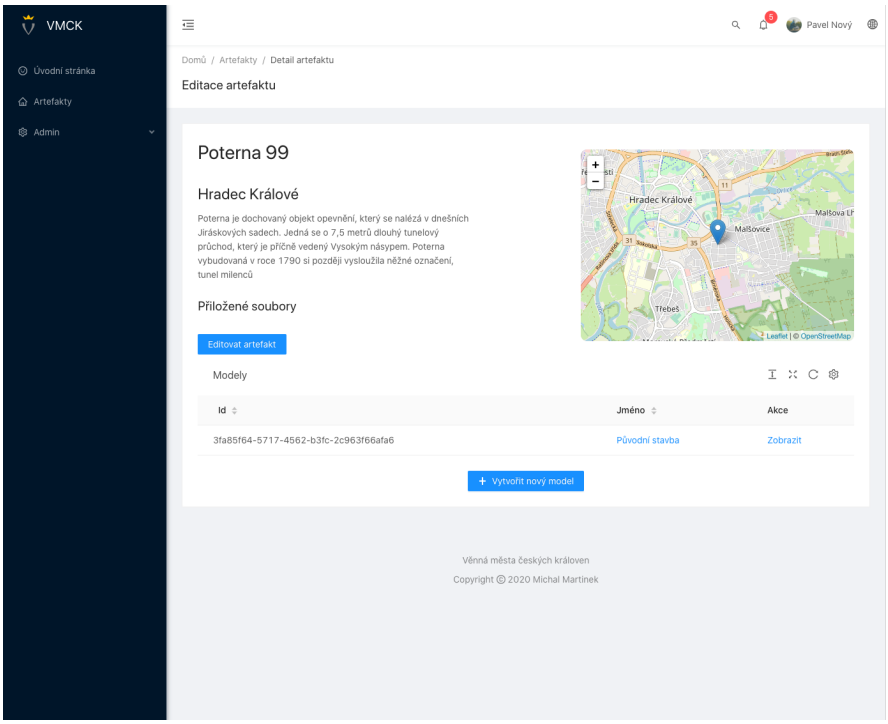
Obrázek D.7: Interaktivní komentování modelu.



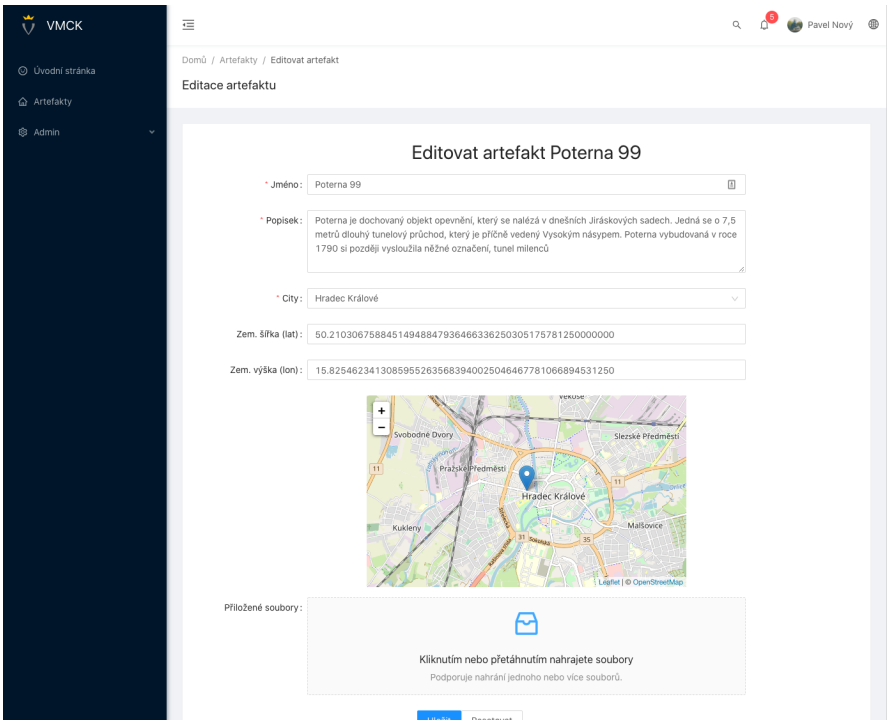
Obrázek D.8: Interaktivní komentování modelu II.



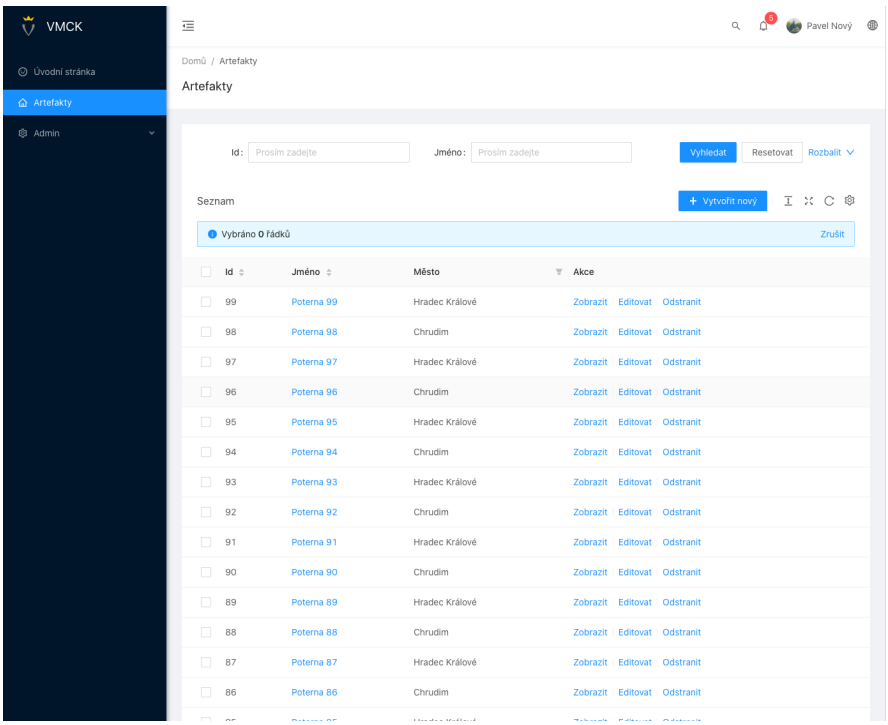
Obrázek D.9: Notifikace.



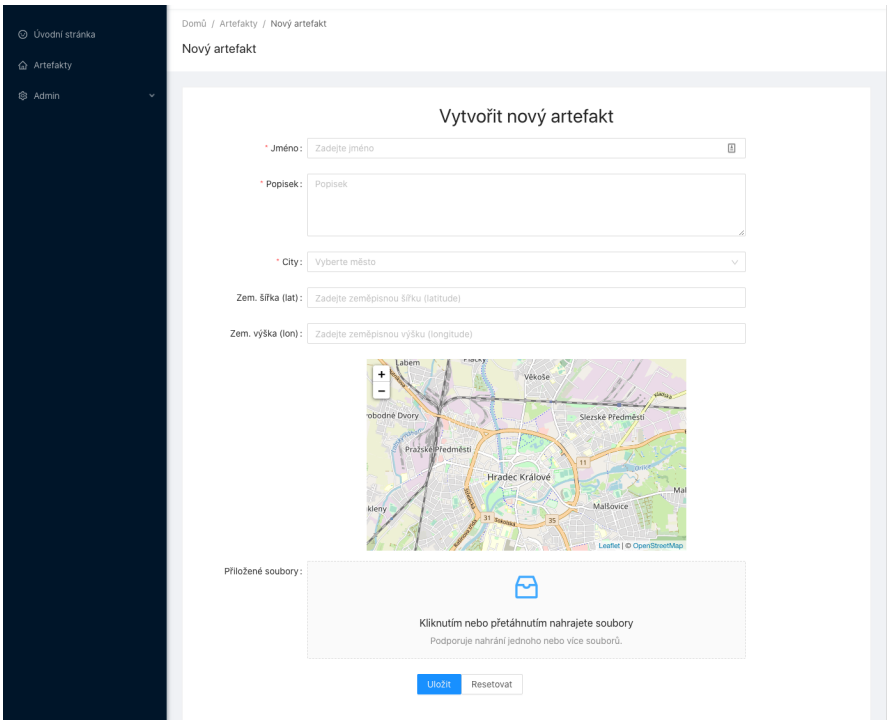
Obrázek D.10: Detail entity *structure*.



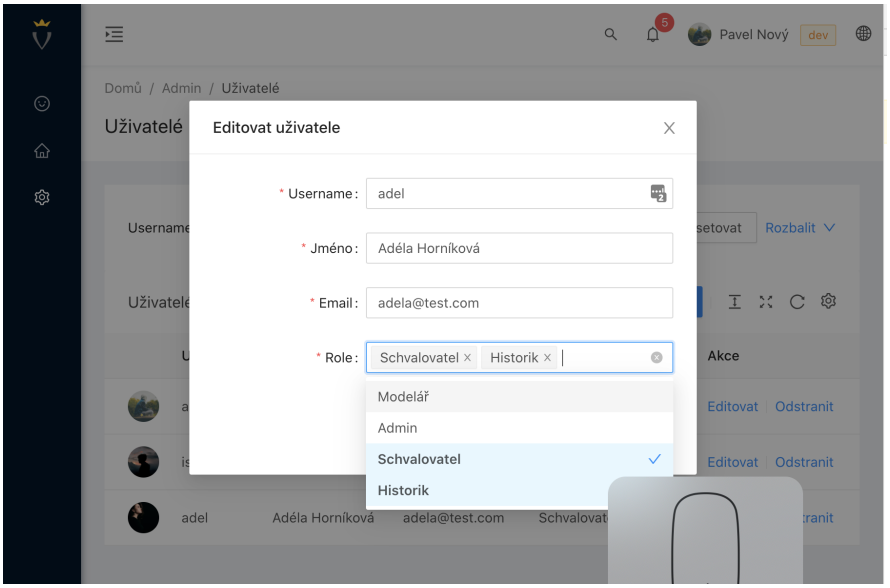
Obrázek D.11: Editace entity *structure*.



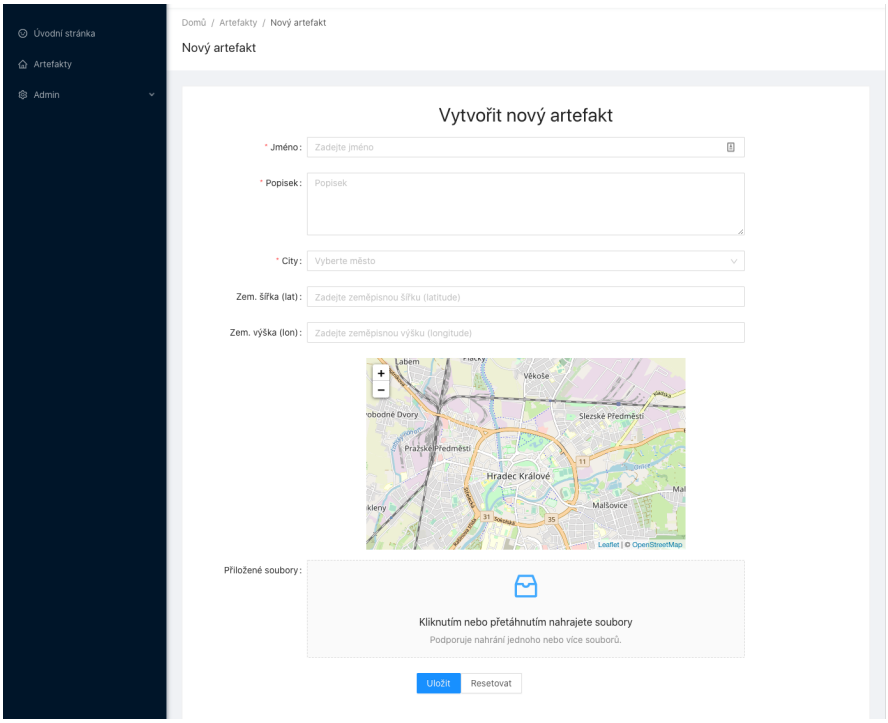
Obrázek D.12: Seznam entit *structure*.



Obrázek D.13: Formulář entity *structure*.



Obrázek D.14: Formulář uživatele.



Obrázek D.15: Seznam uživatelů.

Příloha **E**

Obsah přiložené SD karty