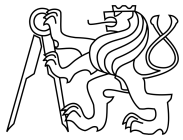
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Webové interaktivní rozhraní pro návrh blokových schémat
Student:	Bc. Jan Tesařík
Vedoucí:	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat webové rozhraní pro aplikaci, která je implementována jako webová služba. Rozhraní umožní uživateli ve webovém prohlížeči vkládat a spojovat funkční bloky do celků a vytvářet pracovní procesy (workflow). Rozhraní umožní i schémata ukládat a načítat ve formátu JSON. Cílem práce není implementovat funkcionality bloků na straně serveru.

1) Prozkoumejte a inspirujte se současnými softwarovými nástroji, které by bylo možné využít pro danou úlohu.
2) Navrhněte frontend webové aplikace. Po diskuzi s vedoucím práce zvolte vhodné implementační nástroje. V rámci backendu navrhněte pouze volání webové služby. Vstupem pro webovou službu je vytvořené schéma a výstupem hodnoty předávané zpět do rozhraní pro jejich vizualizaci.
3) Navržené řešení implementujte a ověřte jeho funkčnost.
4) Proveďte uživatelské testování a zpětnou vazbu zapracujte do webového rozhraní.
5) Webovou aplikaci zdokumentujte, včetně celkové a kontextové nápovědy a manuálu k rozhraní.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 15. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Webové interaktivní rozhraní pro návrh blokových schémat

Bc. Jan Tesařík

Vedoucí práce: doc. RNDr. Ing. Marcel Jiřina, Ph.D.

9. května 2017

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 9. května 2017 .....

Odkaz na tuto práci

Tesařík, Jan. *Webové interaktivní rozhraní pro návrh blokových schémat.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cílem této práce je navrhnout a implementovat vhodné webové rozhraní, tedy front-end webové aplikace, pro v současné době vznikající back-end projektu, který je znám pod jménem Surmon. Smyslem projektu je umožňit uživatelům vytvářet algoritmy pomocí grafického rozhraní pro sestavování blokových schémat. To jsou v podstatě diagramy znázorňující požadovaný tok dat přes bloky schématu, na kterých jsou data zpracovávána a měněna. Některé bloky pak slouží jako vstup do algoritmu a některé jako výstup. V této práci se budu zabývat návrhem a implementací tvorby těchto schémat ve webovém rozhraní, komunikací front-endu s back-endem, ale nikoliv funkcionalitou jednotlivých bloků, která je implementována v back-endu. Touto funkcionalitou může být například zpracovávání obrázků, matematické funkce a mnoho dalších.

Klíčová slova blokové schéma, diagram, webová aplikace, Javascript, front-end, uživatelské rozhraní

Abstract

The aim of this thesis is to design and implement appropriate web interface or in other words, a front-end of a web application, for a back-end that is currently being developed. The project as a whole is called Surmon and it

allows users to create algorithms by constructing block schemes using a graphical interface. Block schemes are generally diagrams depicting a flow of data through the blocks of the scheme, on which the data is processed and used for various calculations. Some blocks serve as the input and some as the output of the scheme. In this thesis, I will be dealing with the design and implementation of the creation of these schemes on the client's side, communication with the back-end, but the functionality of the blocks is the back-end's responsibility. The functionality can be for example image processing or mathematical functions among many others.

Keywords block scheme, diagram, web application, Javascript, front-end, user interface

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Surmon [1] je projekt firmy Surmon, s.r.o., který je v současnosti realizován jako desktopová aplikace, jejímž úkolem je umožnit uživatelům vytvářet algoritmy pomocí skládání schémat z bloků a hran, které bloky propojují. Tyto bloky poskytují veliké množství různých funkcí – zpracovávání obrázků, matematické funkce, zobrazování výsledků, poskytnutí vstupu a mnoho dalších. Bloky tedy pracují s různými typy dat (obrázky, čísla, tabulky a jiné). Schémata jsou organizována do projektů a v rámci jednoho projektu je možné používat schémata uvnitř jiných schémat jako bloky (takzvané superbloky).

Desktopová aplikace, která je v současné době používaná, běží na platformě Netbeans a je napsána v jazyce Java. Vzhledem k požadavku vytvořit webovou aplikaci probíhá v současnosti oddělování back-endu a front-endu aplikace, aby mohl existovat samostatný back-end, který by poskytoval API různým klientům (a tedy i v této práci řešenému webovému rozhraní). Z tohoto důvodu vznikla také potřeba implementovat zmíněný front-end webové verze aplikace Surmon, který by byl schopen komunikovat s postupně vznikajícím back-endem a nabídl by uživatelům stejnou funkcionalitu jako desktopová verze.

Realizace webového rozhraní přinese uživatelům možnost využívat služeb aplikace Surmon a přistoupit ke svým uloženým projektům odkudkoliv za podmínky připojení k internetu. Ovšem kvůli zavedení komunikace mezi back-endem a front-endem se dá předpokládat, že dojde k určitému zpomalení používání aplikace přes webové rozhraní. Proto bude nutné implementovat webové rozhraní tak, aby pracovalo efektivně a nebylo nuceno často komunikovat s back-endem.

Toto téma jsem si vybral proto, že spadá do mého studijního oboru „Webové inženýrství“ a týká se implementace front-endu a uživatelského rozhraní v jazyce Javascript, což jsou konkrétnější podobory, kterými bych se chtěl v budoucnu profesionálně zabývat. Proto jsem rád, že mám možnost řešit realizaci takovéto aplikace a naučit se přitom některé techniky návrhu a implementace front-endu v Javascriptu.

Kapitola 1

Cíl práce

Cílem této práce je navrhnout a vytvořit webové rozhraní pro aplikaci Surmon. Jedná se pouze o tvorbu samotného front-endu, který bude komunikovat s back-endem výměnou dat ve formátu JSON. Kvůli tomu bude také potřeba navrhnout model komunikace front-endu s back-endem. Webové rozhraní by mělo poskytnout uživatelům nástroje pro vytváření schémat pomocí bloků a hran, které je propojují, ukládání a načítání projektů z back-endu, importování schémat z disku a upravování vlastností jednotlivých bloků schématu. Po vytvoření schématu a nastavení bloků bude uživatel moci spustit schématem reprezentovaný algoritmus, jehož výpočet se provede na back-endu a výsledky budou prezentovány uživateli ve webovém rozhraní.

Součástí práce je také řešení uživatelského rozhraní front-endu, které bude zpracováno do podoby specifikace uživatelského rozhraní (dále jen UI specifikace). Uživatelské rozhraní se bude inspirovat rozhraními podobných aplikací a bude provedeno uživatelské testování, podle jehož výsledků by mělo být webové rozhraní upraveno. Dalším dokumentem, který kromě UI specifikace vznikne, je manuál pro uživatele, který bude popisovat použití aplikace. Nedílnou součástí bude též dokumentace kódu pro programátory, kteří v budoucnosti přijdou do kontaktu s kódem front-endu.

Existují určité požadavky, které jsou na webové rozhraní kladeny a kterými by se rozhraní mělo řídit. Mezi ně patří především konzistence s existující desktopovou aplikací a kompatibilita se vznikajícím back-endem. Jinými slovy, je potřeba, aby vytvářený front-end poskytoval uživatelům ty funkce, které poskytuje desktopová aplikace a aby byl schopen komunikovat s back-endem. Dalším požadavkem je rychlost, jelikož se jedná o webovou aplikaci a ta potřebuje pro práci využívat síťovou komunikaci, která může práci s aplikací zpomalovat.

Kapitola 2

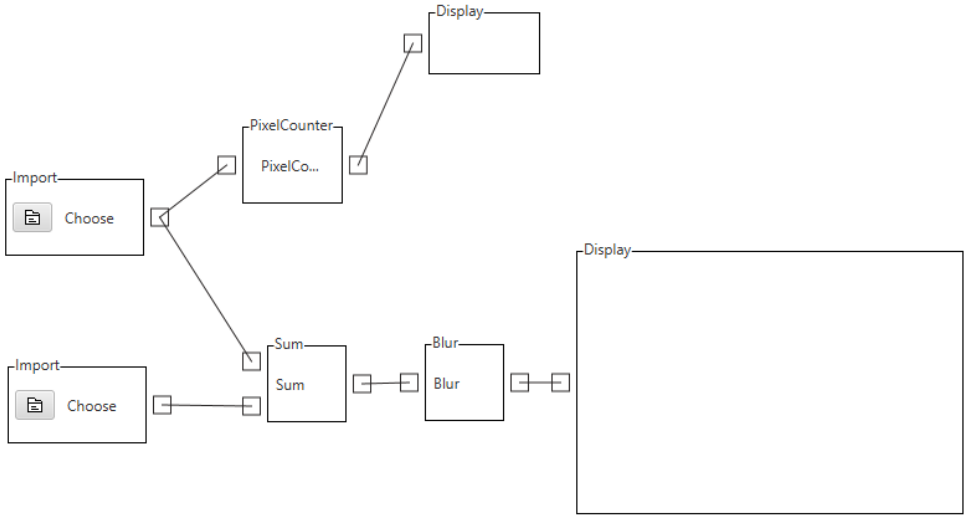
Analýza a návrh

V této kapitole se budu zabývat zkoumáním existující desktopové verze aplikace Surmon a podobných existujících aplikací, abych zjistil, jaká funkcionalita je od aplikace očekávána a jaké formy uživatelského rozhraní a způsoby interakce s aplikací bych mohl do webového rozhraní zavést. Dále budu zkoumat, jak by mohlo být webové rozhraní postaveno a jakým způsobem by mělo komunikovat s back-endem. V neposlední řadě se budu zabývat analýzou dostupných technologií a postupů pro vývoj webového rozhraní. Kromě této analytické činnosti budu vždy na základě získaných poznatků formulovat návrh na řešení daného problému.

2.1 Analýza funkcionality existující desktopové aplikace

Desktopová aplikace nabízí poměrně širokou funkcionalitu, a navíc se stále vyvíjí a prochází změnami. Například jednou z větších změn je přechod na datový formát JSON místo XML pro reprezentaci schémat. Mnou vytvářené webové rozhraní bude mít skoro stejnou funkcionalitu, neboť se vlastně jedná pouze o webovou variantu aplikace. Proto jsem se rozhodl nejprve prozkoumat, jak desktopová aplikace funguje a co umí, abych zjistil důležité požadavky na funkcionalitu webového rozhraní.

Nejdříve zde popíši, jak fungují bloková schémata v aplikaci Surmon. Blok reprezentuje nějakou operaci nad daty, popřípadě poskytuje vstupní data či zobrazuje výstup. Každý blok má nula až několik vstupních pinů a nula až několik výstupních pinů. Tyto piny slouží k propojování bloků a poskytují, respektive přijímají určité typy dat. Z jednoho výstupního pinu může vést neomezeně mnoho hran do vstupních pinů ostatních bloků, ale do vstupního pinu může vést pouze jedna hrana. Vstupní bloky nemají vstupní piny a výstupní bloky nemají výstupní piny. Vytvořené schéma (popřípadě jeho označenou část) je možné aktivovat (spustit), čímž se začnou zpracovávat vstupní data



Obrázek 2.1: Příklad schématu z desktopové aplikace Surmon

na blocích schématu, která se pak předávají dále až do výstupních bloků, kde jsou výsledná data zobrazena. Příklad schématu je na obrázku 2.1.

Některé bloky mají nastavitelné vlastnosti, které ovlivňují zpracování dat. Těmito vlastnostmi jsou typicky parametry funkcí, které bloky reprezentují, ale patří mezi ně i data nahraná ve vstupních blocích. Bloky také mohou mít více vstupů nebo výstupů, což se pozná podle množství vstupních či výstupních pinů, které obsahují. Některé vstupní piny mají tu vlastnost, že se po propojení duplikují (k bloku se přidá kopie daného pinu), aby mohl do bloku vést další vstup.

Existují také zvláštní bloky zvané superbloky, což jsou schémata umístěná v jiných schématech v podobě bloku. Díky tomu není nutné mít všechny bloky v jednom schématu a také to umožňuje použití jednoho schématu ve více jiných schématech.

Nejvíce funkcionalit v aplikaci Surmon se týká tvorby a manipulace se schématem, jelikož toto je hlavní náplní práce s touto aplikací. Nicméně je také potřeba poskytnout uživatelům funkcionalitu týkající se projektů, do kterých jsou schémata ukládána. Konkrétními základními funkcionalitami, které aplikace nabízí, jsou:

- přidání bloku z palety dostupných bloků do plochy schématu
- přidání schématu jako superbloku do plochy schématu
- propojení dvou bloků, jejichž piny jsou kompatibilní (pracují se stejným typem dat)

- zobrazení všech vlastností bloku v panelu na pravé straně obrazovky
- úprava editovatelné vlastnosti bloku (v bloku samotném nebo v pravém panelu)
- změna velikosti bloku
- aktivace schématu / části schématu / bloku
- zobrazení dat výstupního bloku na větší ploše
- rozpojení hrany
- smazání bloku / části schématu / celého schématu
- zobrazení zdrojové podoby schématu (formát JSON)
- otevření / zavření projektu
- vytvoření / smazání projektu
- přidání nového schématu do projektu
- zobrazení / zavření schématu
- smazání schématu z projektu
- uložení zobrazeného schématu
- uložení všech schémat ze všech otevřených projektů
- přejmenování projektu nebo schématu

Webové rozhraní by mělo zmíněnou funkcionalitu implementovat a také by mělo zahrnovat funkcionalitu práce s uživatelskými účty, neboť by bylo vhodné, aby mohly být projekty a schémata ukládány na serveru k určitým uživatelským účtům. Surmon se bude určitě do budoucna měnit a vyvíjet, z čehož bude vyplývat potřeba zahrnout novou funkcionalitu do webového rozhraní.

2.2 Analýza uživatelského rozhraní desktopové aplikace a podobných aplikací

Kvůli inspiraci pro návrh uživatelského rozhraní, popřípadě pro tvorbu funkcionalit, jsem analyzoval desktopovou aplikaci Surmon a několik podobných aplikací, které také pracují se schématy či diagramy nějakého druhu. Existuje několik aplikací, které umožňují vytváření spustitelných diagramů – RapidMiner Studio [2], IBM SPSS Modeller [3], Dell Statistica [4] a další. Toto jsou

všechno desktopové aplikace, a proto jsem pro analýzu zvolil i některé webové aplikace, které spouštění vytvořených diagramů nepodporují. V této sekci tedy popíši uživatelská rozhraní analyzovaných aplikací, ale podrobnější popis nalezených kladů a záporů uživatelských rozhraní se nachází v UI specifikaci (viz přiložené CD).

2.2.1 Surmon – desktopová aplikace

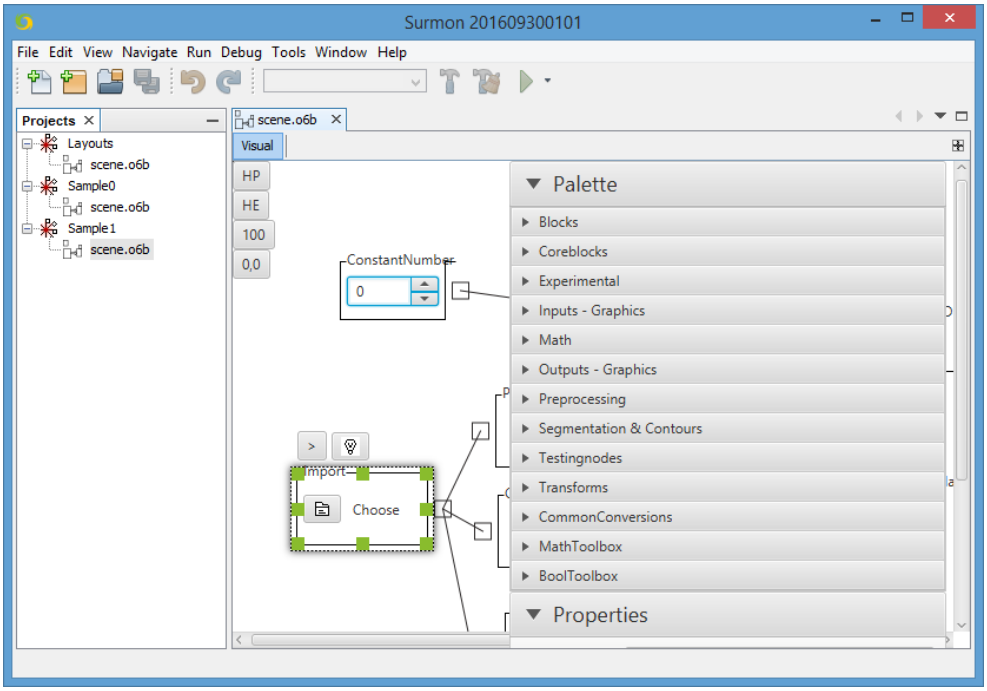
Desktopová aplikace Surmon je právě předělávána, a tedy některé její části jsou v tuto chvíli nedodělané. Z tohoto důvodu je pravděpodobné, že některé nedostatky, na které jsem během analýzy narazil, jsou pouze způsobeny rozpracováným stavem aplikace a nebo také omezeními platformy NetBeans [5], na které je aplikace postavena.

Po spuštění aplikace se zobrazí okno s prázdným obsahem (jsou zobrazena pouze horní menu). Uživatel tedy musí nejprve vytvořit projekt stejným způsobem jako v NetBeans a poté se mu zobrazí u levé hrany okna panel s otevřenými projekty. Po vytvoření a otevření nového schématu v projektu se u pravé hrany okna zobrazí paleta dostupných bloků a pod ní vlastnosti označeného bloku, které je možné zde editovat. Otevřená schémata se zobrazují v záložkách nad plátnem pro tvorbu schématu. Mezi těmito záložkami mohou být i záložky výstupů aktivovaných schémat. Každé schéma by mělo být možné zobrazit buď pomocí bloků nebo jeho zdrojového kódu ve formátu JSON, ale zatím je funkční jen vizualizace pomocí bloků. V aplikaci je prezentováno veliké množství funkcí, které jsou umístěny v nabídkách položek hlavního menu. Některé z těchto funkcí se však netýkají práce se schématy a nejspíše by v aplikaci neměly být, jelikož mohou odvádět pozornost uživatele.

Uživatelské rozhraní používá kontextovou nápovědu, která má podobu tooltipů u pinů či bloků, a také poskytuje odkazy na webové stránky dokumentace aplikace Surmon pro jednotlivé bloky. Práce s bloky je mírně těžkopádná, neboť se bloky dají škálovat a posouvat pouze po označení bloku klikem, přičemž v tomto stavu zase není možné měnit vlastnosti bloku zobrazené přímo v něm samotném. Takto zobrazené vlastnosti jsou zatím přítomny jen u malé části dostupných bloků a zdá se, že těmito bloky jsou hlavně vstupní bloky, u kterých se nastavuje vstup. Většina bloků pak nemá žádný obsah kromě svého názvu. U všech bloků se po jejich označení objeví menu pro práci s blokem (aktivace, smazání, odkaz na nápovědu, atd.). Uživatelské rozhraní je vidět na obrázku 2.2.

2.2.2 RapidMiner Studio

RapidMiner Studio [2] je aplikace pro vytváření procesů týkající se strojového učení, hlubokého učení, dolování dat z textu a prediktivní analýzy. Schémata se skládají z bloků a jejich struktura a spouštění funguje na podobném principu



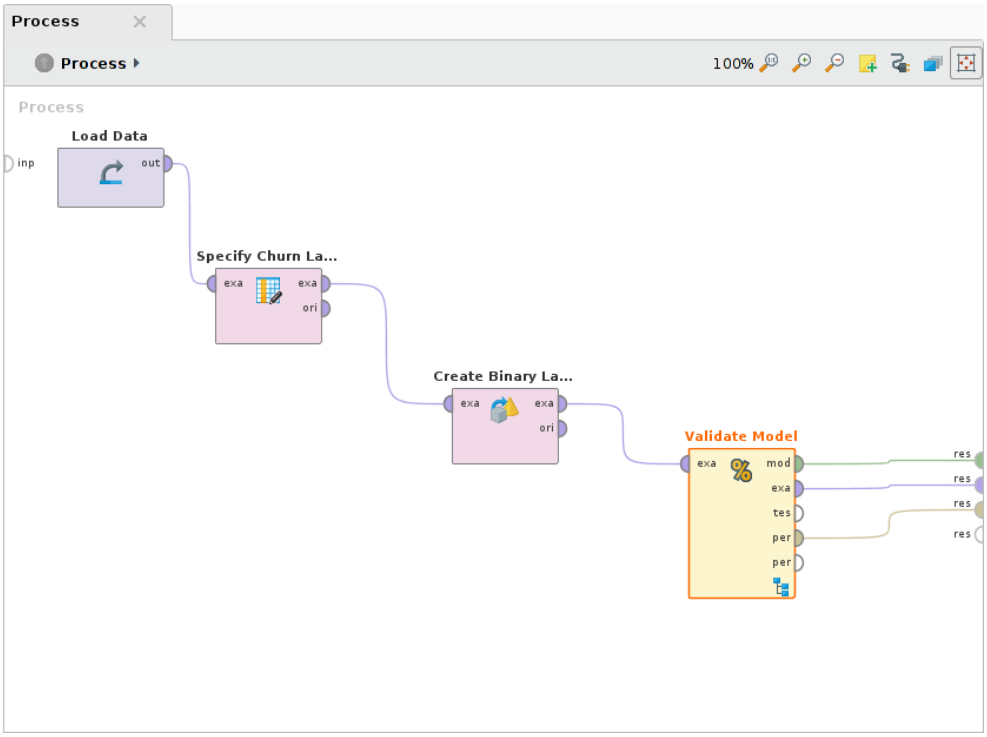
Obrázek 2.2: Zobrazení celého uživatelského rozhraní desktopové verze aplikace Surmon.

jako Surmon. RapidMiner Studio poskytuje velmi širokou množinu nástrojů a jedná se o velmi profesionálně vyhlížející software.

Základní rozložení komponent uživatelského rozhraní je poměrně standardní pro takovéto aplikace. Skládá se ze dvou postranních panelů a plochy pro tvorbu procesů, která je uprostřed. Pro začínající uživatele se při prvním použití zobrazí při levé hraně okna ještě panel s návodem jak vytvořit první jednoduchý proces. Tento návod je velmi jednoduchý na pochopení a kromě něj poskytuje aplikace spoustu různých nápověd rozmístěných různě po obrazovce. Aplikce nabízí opravdu hodně možností nápovědy, včetně kontextové nápovědy a pomocných automatických nástrojů (například doporučování bloků, které by mohly být přidány do procesu).

Oproti Surmonu je zde rozdíl v obsazích postranních panelů. Levý obsahuje paletu vstupních bloků a paletu operátorů (vnitřní bloky) a pravý panel zobrazuje vlastnosti označeného bloku a také nápovědu k označenému bloku. V aplikaci neexistuje koncept projektů, které by zapouzdřovaly více procesů, takže zde není žádné okno pro zobrazení seznamu projektů – zobrazují se pouze otevřené procesy, a to v záložkách v záhlaví plochy pro tvorbu procesů.

Práce se schématem je podobná jako v aplikaci Surmon, ale některé detaily se liší. Například není možné upravovat vlastnosti bloků uvnitř nich samotných, z výstupního pinu může vycházet pouze jedna hrana, na plátně procesu



Obrázek 2.3: Zobrazení schématu v RapidMiner Studio

nemůže být blok, který není nijak propojen. Po spuštění procesu se automaticky zobrazí všechny výsledky a pro každý proces je možné přepínat mezi jeho blokovou reprezentací a výsledky jeho spuštění. RapidMiner Studio má i určité nedostataky, mezi které patří třeba těžkopádnější zbavování se varovných dialogů a neexistence menu pro práci s označeným blokem. Ukázka části uživatelského rozhraní RapidMiner Studia je vidět na obrázku 2.3.

2.2.3 Draw.io

Draw.io [6] je aplikace pro tvorbu nejen všech možných typů diagramů, jako jsou UML a BPMN diagramy či pouze obecné diagramy bez konkrétních významů bloků a charakteristik, ale také pro tvorbu grafů a wireframů. Aplikace je napsána pomocí jazyků HTML, SVG, CSS a Javascript, tedy se nejedná o jednu z mnoha Flash aplikací. Draw.io je také dostupné jako doplněk pro aplikaci jménem Confluence [7].

Základní rozložení komponent této aplikace má mnoho společných rysů s RapidMiner Studiem. Opět je zde levý panel s paletou bloků (je možné v nich vyhledávat), vedle něj plátno pro tvorbu diagramů a nakonec pravý panel s vlastnostmi označeného bloku, popřípadě vlastnostmi plátna, pokud zrovna není žádný blok označen. Najednou může být zobrazen pouze jeden

diagram, který však může mít více stránek. Ty jsou pak umístěné pod plátnem pro tvorbu diagramů.

Některé odlišnosti Draw.io od Surmonu vyplývají z toho, že diagramy v Draw.io nereprezentují spustitelné algoritmy, ale pouze statické diagramy, které popisují nějaký proces či model. Proto třeba bloky diagramů nemají konkrétní piny a nemají žádné nastavitelné parametry, pouze textový obsah, který však může mít i bohatší strukturu. Editovatelné vlastnosti bloku zobrazené v pravém panelu zahrnují pouze grafické vlastnosti. Bloky jsou v paletě reprezentovány jen obrázkem místo textu, na rozdíl od Surmonu i RapidMiner Studia, protože se jedná o standardní definované tvary, kterým návrháři pracující s těmito diagramy rozumí.

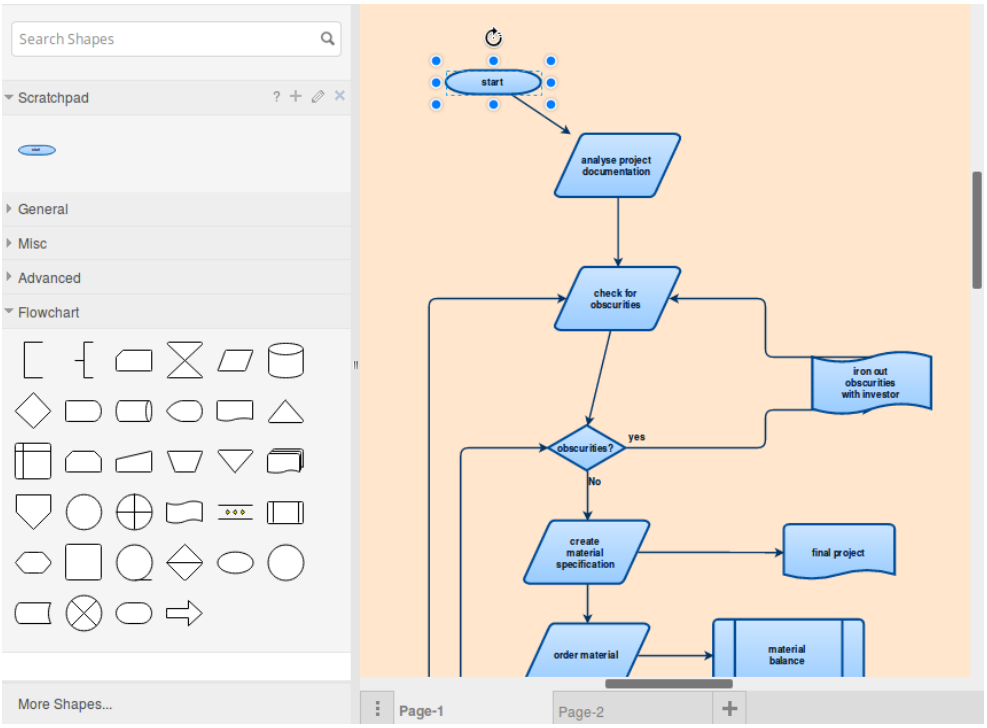
Draw.io má jednu nepěknou vlastnost, a tou je pomalost. Aplikace se při práci chová trhaně a také načítání bloků z určité kategorie bloků může zabrat i pár vteřin. Dalším nedostatkem je práce s hranami diagramů, které mají občas nelogický průběh a divný tvar. Jinak je aplikace dobře použitelná a nabízí širokou škálu nástrojů, jako například vytvoření vlastního bloku, uložení konkrétního bloku z diagramu jako předvytvořený blok, rozsáhlé grafické úpravy bloků, šest různých způsobů uložení diagramu – do prohlížeče, na disk nebo do různých externích online úložišť, jako je například Dropbox [8]. Bloky se při přesouvání zarovnávají podle ostatních bloků a když je blok právě tažen, tak zůstává na místě a místo něj se přesouvá pouze jeho zástupce. Až když uživatel pustí tlačítko myši, tak se blok přesune a přepočítají se hrany. Toto urychluje práci s aplikací, jelikož se hrany nemusí neustále přepočítávat během tažení. Uživatelské rozhraní je vidět na obrázku 2.4.

2.2.4 Lucidchart

Lucidchart [9] je další aplikace jako Draw.io, která též slouží jako nástroj pro vytváření různých diagramů, grafů a wireframů. Také je založena na stejných technologiích, jako Draw.io. Tato aplikace je první z mnou testovaných, která umožňuje ukládání diagramů na server pod daný uživatelský účet. Lucidchart je v základní neplacené verzi ořezán o některé možnosti, ale pro analýzu uživatelského rozhraní to nevadí.

Jedním z rozdílů Lucidchartu oproti ostatním analyzovaným aplikacím je to, že zahrnuje správu uživatelských účtů. Součástí aplikace tedy není jenom samotné vytváření diagramů, ale také tvorba účtu. Díky účtům je možné provádět akce jako sdílení diagramů s ostatními uživateli a přidávání komentářů k diagramům. V obrazovce pro vytváření diagramů jsou v horní liště přítomna tlačítka pro tyto akce a také se tam nachází zapnutí prezentačního módu, ve kterém je zobrazen diagram včetně jeho stránek na celou obrazovku a nedá se upravovat. Uživatel Lucidchartu má k dispozici stránku zobrazující všechny jeho diagramy, popřípadě diagramy s ním sdílené.

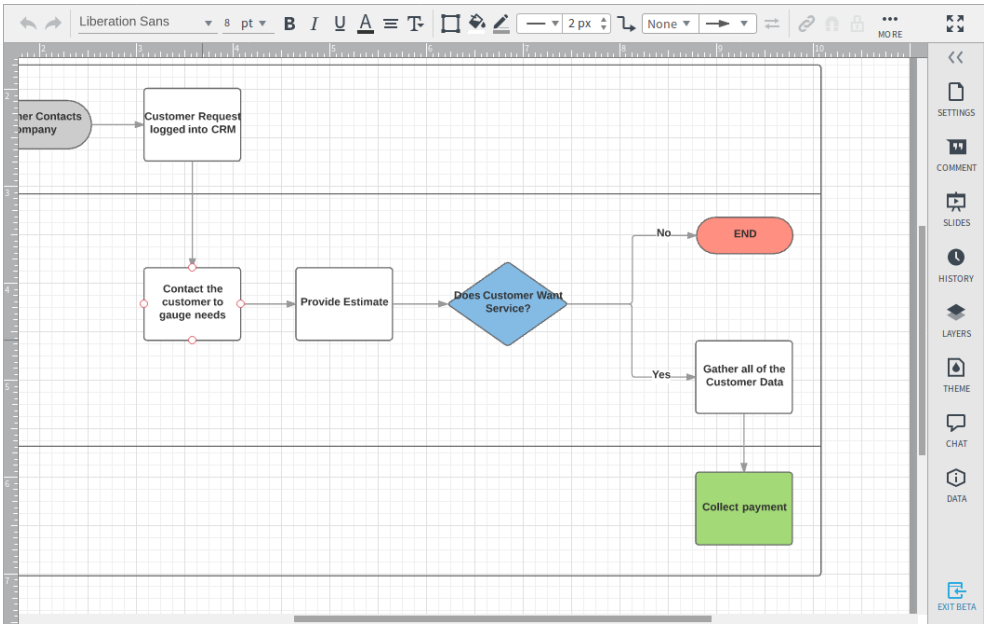
Mimo uživatelských účtů se v souhrnu dá říci, že uživatelské rozhraní Lucidchartu je velmi podobné Draw.io. Je zde však pár odlišností v rámci zá-



Obrázek 2.4: Levý panel a kreslící plátno aplikace Draw.io

kladního rozložení komponent. Jednou z nich je umístění záložek reprezentující stránky diagramu nad plátno pro tvorbu diagramu. Ikona „+“ pro vytvoření nové záložky je umístěna dále od záložek, než je tomu u Draw.io, nicméně nové záložky se dají přidávat i přes menu „Page“ (nikoliv menu „File“, kde jsem funkcionalitu původně očekával). Výraznější změnou je však pravý panel – ten pouze zobrazuje menu s položkami, kliknutím na které se zobrazují různá okna pro úpravu určitého typu vlastností bloku nebo samotného diagramu. Díky takto malému pravému panelu je k dispozici více místa pro plátno diagramu, nicméně na druhou stranu nejsou žádné vlastnosti bloku po jeho označení ihned vidět a je třeba nejprve kliknout na jednu položku z pravého panelu.

Práce s aplikací je o trochu rychlejší, než v případě Draw.io a také zde není takový problém se směřováním hran. Uživatelské rozhraní automaticky zarovnává bloky, roztahuje plochu, když už se do ní diagram nevejde a umožňuje posouvat bloky pomocí šipek na klávesnici. Jednou ze zajímavějších funkcionalit je zobrazení navigátoru, což je minimapa diagramu pro zlepšení orientace. Také je možné nastavit si pravidelné připomínání, že je potřeba aktualizovat nějaký diagram. Ukázka uživatelského rozhraní Lucidchartu je na obrázku 2.5.



Obrázek 2.5: Kreslící plátno a pravý panel aplikace Lucidchart

2.3 Návrh uživatelského rozhraní

Nejdříve jsem navrhoval uživatelské rozhraní, jelikož jeho prvotní návrh není závislý na platformě, použitých technologiích nebo jiných konkrétních aspektech webového rozhraní. V této sekci budu popisovat, jak jsem postupoval při návrhu uživatelského rozhraní a zaměřím se na popis jeho komponent a vlastností (detailnější popis navrženého uživatelského rozhraní se nachází v přiložené UI specifikaci na CD). Z tohoto návrhu pak budu vycházet při realizaci uživatelského rozhraní front-endu.

2.3.1 Úlohy

Nejprve jsem se zaměřil na to, které úlohy bude uživatel s aplikací provádět, abych věděl, co všechno bude potřeba do uživatelského rozhraní zahrnout a jakým způsobem by měly být jednotlivé úlohy prezentovány uživateli. Úlohy jsem seskupil do kategorií podle toho, kterých částí aplikace se týkají. Nakreslil jsem graf, který znázorňuje, ze které komponenty uživatelského rozhraní jednotlivé úlohy pochází a také které komponenty ovlivňují (toto vše je dostupné v UI specifikaci).

2.3.2 Wireframy a komponenty

Pomocí wireframů jsem definoval, jak by mělo vypadat rozmístění komponent uživatelského rozhraní v různých stavech aplikace. Tyto wireframy ještě

nezachycují výslednou podobu uživatelského rozhraní, ale můžou sloužit k počátečnímu testování a také jako základ pro vytvoření finálního návrhu či přímo implementace uživatelského rozhraní. Ve výsledku jsem tedy rozdělil uživatelské rozhraní na komponenty, jak je vidět na obrázku 2.6. Komponenty jsou zodpovědny za následující funkcionalitu:

- **Hlavní menu** – Poskytuje základní akce, jako je vytvoření nového projektu, nového schématu, otevírání projektů a jejich ukládání. Vytvoření schématu z hlavního menu vloží nové schéma do právě aktivního projektu, což je ten, který je v panelu projektů zvýrazněn. Toto menu by mělo být tím prvním, čeho si uživatelé všimnou a s čím začnou pracovat.
- **Panel projektů** – Tento panel zobrazuje seznam otevřených projektů. Schémata obsažená v projektu se dají zobrazovat a schovávat klikem na položku projektu – panel tedy funguje jako takzvaný akordión [10]. Z tohoto panelu se také schémata otevírají a dají se s nimi skrze panel projektů provádět určité operace (mazání, přejmenování, uložení, export). Nad projekty lze provádět tytéž operace, ale navíc lze do nich přidávat nová či importovat existující schémata.
- **Záložky** – Při otevření schématu z panelu projektů se zobrazí záložka v záložkovém řádku pod hlavním menu, která po kliknutí na ní zobrazí příslušné schéma. Záložky se dají zavírat, přičemž po zavření záložky se zobrazí schéma z následující záložky, popřípadě z té předchozí, pokud žádná následující není.
- **Přepínač pohledu na schéma** – Existují celkem tři pohledy na schéma, módy zobrazení schématu, mezi kterými se dá přepínat. Nejdůležitějším je vizuální zobrazení schématu jako diagramu, díky čemuž se dá schéma vytvářet a měnit. Dalším je zobrazení zdrojového kódu ve formátu JSON a nakonec zobrazení výsledků aktivace schématu, jsou-li nějaké.
- **Plátno schématu** – Plocha, kde se různým způsobem schéma zobrazuje. Při vizuálním zobrazení schématu lze do něj přidávat bloky, propojovat je hranami přes jejich piny a upravovat jejich parametry, které mohou být různých datových typů.
- **Paleta bloků** – Odtud se do schématu primárně přetahují bloky (do plochy se mohou přetahovat i schémata z panelu projektů jako superbloky), čímž se do něj vkládají. Bloky jsou v paletě uspořádány do kategorií a bloky dané kategorie se dají zobrazit nebo skrýt kliknutím na položku kategorie (jako je tomu u panelu projektů).
- **Panel vlastností** – Zde dochází k zobrazování vlastností označeného bloku. Mezi takto zobrazené vlastnosti patří standardní vlastnosti, jako je například název nebo identifikátor bloku. Dále tam patří i specifické vlastnosti pro daný typ bloku.

Obrázek 2.6: Zobrazení, do jakých komponent je uživatelské rozhraní rozděleno

- **Akce se schématem** – Toto menu umístěné nad plátnem obsahuje akce, které ovlivňují schéma jako celek. Patří sem aktivace schématu, smazání všech bloků a hran a export schématu.

Superbloky jsou v návrhu řešeny tak, že každé schéma obsahuje takzvané bloky vstupního a výstupního rozhraní. Blok vstupního rozhraní je speciální blok, jehož výstupní piny definují vstupní rozhraní (vstupní piny) daného schématu. Zpočátku je tento pin pouze jeden, ale propojováním s vnitřními bloky schématu se bude tento pin postupně duplikovat. Blok výstupního rozhraní funguje analogicky, ale pro výstupy schématu. Z tohoto vyplývá, že každé schéma může být samostatné, nebo může využít nějaký pin bloku vstupního či výstupního rozhraní, a tím pádem je připraveno komunikovat s ostatními schématy či samostatnými bloky.

V rámci uživatelského rozhraní front-endu bude uživatelům přístupná jak kontextová nápověda, tak manuál použití aplikace. Kontextová nápověda bude typicky formou tooltipů u bloků a pinů. Texty těchto tooltipů bude možné měnit, jako to jde u desktopové verze Surmonu, ale v každém případě by měly tooltipy mít nastaven počáteční text, který vysvětluje, co který blok dělá a jaké typy dat piny přijímají nebo naopak posílají dále. Ve zmíněném manuálu pak budou popsány jednotlivé komponenty na obrazovce a také postupy, jak má uživatel provést určité akce.

2.3.3 Inspirace jinými aplikacemi

Některé prvky uživatelských rozhraní, která jsem zkoumal v rámci analýzy mě zaujaly a rozhodl jsem se je zapracovat do svého návrhu. Jedním z prvků je tažení zástupného elementu místo samotného bloku na plátně, jako je tomu v Draw.io, takže se během tažení nemusí překreslovat hrany, které jsou na blok napojené (překreslí se až po skončení přesunu bloku). Z návrhu je také vidět, že jsem se inspiroval RapidMiner Studiem, odkud jsem převzal zobrazování výsledků spuštění schématu v samostatném módu zobrazení schématu. Také jsem navrhl, že bloky vstupního a výstupního rozhraní schématu budou ve všech schématech od začátku přítomny, stejně jako vstupní a výstupní piny na krajích procesů v RapidMiner Studiu.

Jsou i další prvky, které by neměly ve webovém rozhraní Surmonu chybět a které jsou přítomny v analyzovaných aplikacích. Jsou jimi například zarovnávání bloků podle ostatních bloků, podpora klávesových zkratek nebo lepší průběhy hran a vytváření vlastních bloků. Nicméně tyto prvky nejsou zcela nezbytné, avšak mohly by zpříjemnit práci uživatele s aplikací. Každopádně jsem se na ně primárně nezaměřoval, jelikož je důležitější, aby nejprve byla implementována hlavní funkcionalita webového rozhraní.

2.3.4 Vlastnosti uživatelského rozhraní

Uživatelské rozhraní by mělo splňovat určité vlastnosti, které uživatelům usnadňují práci s ním. Jsou různé zdroje, které uvádí různé vlastnosti, ale jednou z nejuznávanějších sad vlastností je deset heuristik použitelnosti podle Jakoba Nielsena [11]. Z tohoto důvodu a také kvůli tomu, že už jsem s heuristikami pracoval, jsem se snažil se jich držet.

Uživatelské rozhraní jsem navrhoval tak, aby bylo konzistentní s tím, na co jsou uživatelé zvyklí při práci s podobnými aplikacemi. Tomu pak odpovídá rozložení komponent na obrazovce, zejména postranní panely a kreslící plocha uprostřed. Takovéto rozložení je typické pro aplikace, které slouží k vytváření diagramů. Nicméně aplikace, které jsem analyzoval, se liší od aplikace Surmon v tom, že nepracují s projekty, takže nepotřebují zobrazovat seznam projektů uživateli. Naopak u aplikací, které s projekty pracují (Netbeans [12], Eclipse [13] a mnoho dalších), existuje seznam otevřených projektů a je umístěn v levém panelu. Právě těmito aplikacemi jsem se inspiroval při navrhování umístění seznamu projektů.

Důležitou vlastností pro uživatelské rozhraní bývá často responzivnost ve smylu schopnosti uživatelského rozhraní přizpůsobovat se změnám velikosti obrazovky. Předpokládá se, že se aplikace Surmon bude používat především na stolních počítačích a noteboocích, neboť se jedná o produkt, který využívají firmy. Z tohoto důvodu je uživatelské rozhraní navrženo především pro zařízení s větší obrazovkou, avšak implementuji ho takovým způsobem, aby bylo alespoň do určité míry responzivní. Nemá smysl podporovat telefony, neboť

by práce se aplikací na nich byla velmi zdlouhavá a nepříjemná, ale použití na tabletu už je realističtější. Každopádně by se při použití na tabletu rozvržení komponent na obrazovce nezměnilo (určitě ne nijak razantně), ale spíše by se jednotlivé komponenty zmenšily, popřípadě by se staly vysouvacími.

Aby mohlo být uživatelské rozhraní efektivněji používáno, bývá dobrým zvykem poskytnout uživateli možnost použití klávesových zkratek, díky kterým je uživatel schopen provést různé úkony rychleji. Webové rozhraní aplikace Surmon by také mělo klávesové zkratky podporovat, aby bylo možné efektivněji provést často se opakující akce. Některé klávesové zkratky, jako například escape pro zavření dialogů nebo enter pro potvrzení formulářů jsou standardně podporovány buďto samotnými prohlížeči nebo některými javascriptovými knihovnami. Kromě nich by mělo uživatelské rozhraní podporovat alespoň tyto klávesové zkratky:

- **delete** – smazání označeného bloku nebo označené hrany
- **ctrl+s** – uložení právě otevřeného schématu
- **ctrl+c a ctrl+v** – zkopírování označeného bloku
- **ctrl+d** – duplikace označeného bloku
- **ctrl+šipka doleva nebo doprava** – přepínání mezi záložkami nejnižší zobrazené úrovně záložek
- **f1** – zobrazení manuálu

Snažil jsem se o to, aby bylo uživatelské rozhraní pokud možno co nejjednodušší, srozumitelné a aby nebylo prezentováno veliké množství informací a akcí najednou. Z tohoto důvodu jsem se rozhodl, že se budou menu u projektů a schémat v levém panelu zobrazovat jen po zobrazení daného projektu či schématu nebo po najetí myší na jejich název. Kromě toho bych chtěl použít samostatné ikony pro reprezentaci různých akcí, ale bude potřeba v rámci uživatelského testování otestovat, jestli jsou ikony pro uživatele srozumitelné.

2.4 Kompatibilita

Při návrhu front-endu webové aplikace je důležité myslet na kompatibilitu napříč prohlížeči. Tato kompatibilita bývá typicky narušena použitím některých funkcionalit, které jsou implementovány až v pozdějších verzích prohlížečů. Míru kompatibility lze zvýšit například použitím polyfillů (knihovna doimplementovávající určitou funkcionalitu pro starší prohlížeče) anebo použitím postupů, které zajistí, že i když na jiném prohlížeči (typicky starší verze) některá funkcionalita nepoběží, bude aplikace poskytovat alespoň základní funkcionalitu. Tyto postupy jsou „graceful degradation“ a „progressive enhancement“.

Graceful degradation znamená, že se nejdříve implementuje front-end s veškerou funkcionalitou pro moderní prohlížeče a poté se upravuje tak, aby podpoval také starší verze prohlížečů i za cenu poskytnutí jen základnějších nástrojů [14]. Progressive enhancement se naopak nejprve zaměřuje na podporu těch nejstarších relevantních verzí prohlížečů a poté se přidává moderní funkcionalita dostupná modernějším prohlížečům [14]. V této práci použiji Graceful degradation, neboť webové rozhraní Surmonu je zaměřeno spíše na novější verze prohlížečů, a navíc pravděpodobně nevyužiji mnoho nových prvků Javascriptu, HTML nebo CSS.

Pro zjištění toho, které verze prohlížečů bude třeba zohlednit, jsem se podíval na to, jaký mají jednotlivé **desktopové** verze prohlížečů podíl na trhu (kolik procent uživatelů na desktopech je používá), jelikož bude webové rozhraní používáno především (a zatím výhradně) na desktopových zařízeních. Podle webové stránky „NetMarketShare“ [15] a také na základě dostupnosti starších verzí prohlížečů jsem zvolil následující verze prohlížečů, na kterých budu provádět testování kompatibility po realizaci webového rozhraní:

- **Google Chrome verze 42** – podíl 1.01 %
- **Mozilla Firefox verze 40** – podíl 0.74 %
- **Opera verze 38** – velmi malý podíl, nicméně všechny verze Opery mají malý podíl, a navíc je Opera hodně podobná Google Chrome, takže teoreticky by nemuselo být nutné Operu testovat
- **Internet Explorer verze 10** (dále jen IE) – podíl 0.88 %, nicméně by bylo ideální, aby byla aplikace funkční i na verzích 9 a 8, i když bez určité funkcionality (použití graceful degradation), protože mají podíl 3.08 % resp. 3.18 %, ale pravděpodobně se podporou těchto prohlížečů nebudu zabývat
- **Edge verze 14** – podíl 4.90 %, nicméně pokud bude aplikace běžet na IE 10, pak bude běžet i na Edge 12 (a tedy i na pozdějších verzích), neboť oba prohlížeče podle stránky „Can I Use“ [16] podporují potřebnou funkcionalitu
- **Safari** testovat nebudu, protože nejsem vlastníkem platformy OS Mac a na Linuxu či Windows je možné zprovoznit pouze verzi 5, která je příliš zastaralá

Pro zjištění, které verze prohlížečů je potřeba podporovat a hlavně které už potřeba podporovat není, by bylo ideální použít nějaký nástroj pro analýzu návštěv, třeba Google Analytics[17], ale tím se v této práci zabývat nebudu. Pro zjišťování, které funkcionality jsou kterými prohlížeči podporovány běžně používám zmíněnou stránku „Can I Use“[16], takže se jí budu držet i v rámci této práce.

2.5 Data v aplikaci

Webové rozhraní bude pracovat s několika typy objektů. Tyto typy vychází z těch typů, které používá existující desktopová aplikace Surmon a které back-end bude zpracovávat a ukládat. Použité typy objektů, jejich vztahy a některé vlastnosti či skupiny vlastností jsou vidět na obrázku 2.7, ze kterého lze vyčíst, že typů není mnoho a že datový model není celkově příliš složitý.

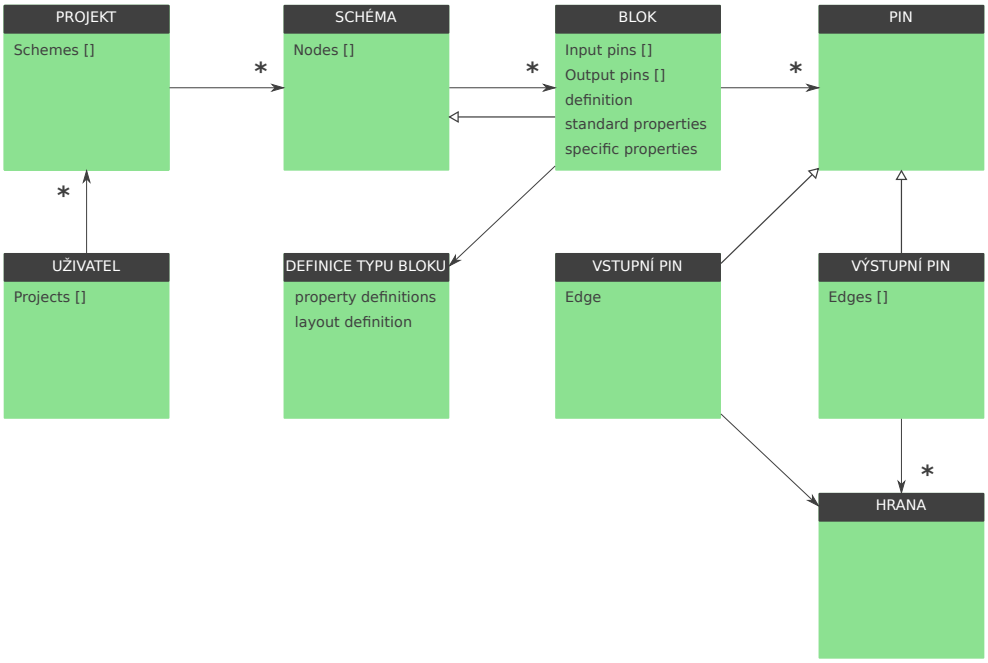
Z typů objektů vyobrazených na obrázku datového modelu stojí za zmínku „Definice typu bloku“, protože instance tohoto typu objektu existují jen pro účely front-endu. Tyto definice totiž slouží pro vytvoření instancí konkrétních bloků, které se pak na definici odkazují, a tím pádem mají vždy k dispozici informace o tom, jak se mají určité vlastnosti bloku zobrazit, jaký mají datový typ, jaké jsou počáteční hodnoty vlastností bloku a další. Samotný blok tedy ukládá současné hodnoty vlastností a definice ukládá metadata o vlastnostech. Příklad definice je vidět na obrázku 2.8.

Součástí definice bloků je i rozmístění zobrazených vlastností uvnitř bloku. Tato rozmístění mohou být definována dvěma způsoby. Oba způsoby jsou ukázány na obrázku 2.8. Jeden způsob reprezentuje rozmístění jako 2D pole, ve kterém řádky odpovídají v řadě za sebou zobrazeným vlastnostem v bloku. Tyto řádky mohou mít různý počet vlastností. Druhý způsob je komplexnější, ale umožňuje vytvářet různorodější rozmístění. Je založen na tom, že se do sebe vnořují objekty, které svůj obsah zobrazují buď horizontálně nebo vertikálně. Uvnitř jejich obsahu mohou být samotné vlastnosti nebo další vnořené objekty.

Obrázek 2.8 také ukazuje, že součástí definic bloků jsou také definice jejich pinů. Ty však nejsou na obrázku 2.7 nakresleny, protože nemají oproti samotným pinům žádnou informaci navíc, takže není potřeba s nimi pracovat zvlášť.

Z diagramu datového modelu je také vidět, že schémata dědí vlastnosti bloku. To proto, že schéma je vlastně také blok, který má funkcionalitu definovanou svojí vnitřní strukturou. Tím pádem jde pracovat se schématem jako s obyčejným blokem, ale navíc má svojí vlastní funkcionalitu a vlastní chování.

Data, se kterými bude webové rozhraní pracovat, jsou trochu odlišná oproti podobě dat na serveru. Na vznikajícím back-endu se budou z dat od klienta vytvářet instance různých tříd (například různé typy bloků), jelikož je pro jeho implementaci použit jazyk Java. Narozdíl od toho se na straně webového klienta s objekty zachází jinak, jelikož se pracuje s Javascriptem, takže může mít každá instance jednoho typu objektu jiné vlastnosti a proměnné nemají definovaný datový typ. Kromě toho na straně klienta nemusí být definovány funkcionality jednotlivých typů bloků, a proto stačí, aby byl na front-endu definován pouze obecný objekt bloku.



Obrázek 2.7: Diagram znázorňující typy objektů, se kterými bude webové rozhraní pracovat.

2.6 Architektura

V této části budu probírat aspekty architektury webového rozhraní. Nejprve se zaměřím na interakci mezi front-endem a back-endem a navrhnu rozhraní pro komunikaci mezi nimi. Dále se budu zabývat vnitřní architekturou samotného front-endu. Prozkoumám možnost použití frameworků a navrhnu strukturu webového rozhraní – rozdělení front-endu na jednotlivé moduly, jejich úlohy a komunikaci mezi nimi.

Jelikož se bude webová aplikace Surmon skládat z front-endu na klientské straně a z back-endu běžícího na serveru, jedná se o standardní Client-server architekturu aplikace jako celku. Nicméně i samotné webové rozhraní bude mít svou vnitřní architekturu, a tou bude MVC (Model-View-Controller), neboť se jedná o standardní architekturu pro webové aplikace a jejich části, tedy front-end i back-end. Architektonický vzor MVC přináší výhody jako efektivnější a přehlednější vývoj, možnost poskytnout více různých zobrazení pro data modelu bez zbytečné duplikace kódu a také to, že změny v jednotlivých částech architektury se tolik neovlivňují [18].

```
{
    class: "org.surmon.coreblocks.ConstantNumber",
    displayName: "ConstantNumber",
    category: "Input",
    specificProperties: {
        inputProperties: {
            number: {
                dataType: "number",
                value: 0
            },
            text: {
                dataType: "text",
                value: "abc"
            },
            range: {
                dataType: "range",
                min: 0,
                max: 100,
                value: 30
            }
        }
    },
    pins: [
        {
            type: "java.lang.Double",
            displayName: "",
            tooltip: "[Double]",
            id: "out0",
            index: 0,
            direction: "output",
            duplicateOnConnect: false
        }
    ],
    layout: [
        ['file', 'number'],
        ['sliderProperty']
    ],
    /* alternativní reprezentace stejného rozmístění */
    layout: {
        direction: 'vertical',
        content: [
            {   /*vnořený layout*/
                direction: 'horizontal',
                content: ['file', 'number']
            },
            'sliderProperty'
        ]
    }
}
```

Obrázek 2.8: Příklad JSON kódu pro definici typu bloku ConstantNumber

2.6.1 Komunikace se serverem

Důležitou součástí architektury Client-server je komunikace klienta se serverem. Nabízí se dva různé postupy: tradiční a SPA (Single page application). Tradiční postup je charakteristický tím, že server poskytuje několik HTML stránek, na které se klient dotazuje, takže po odpovědi na dotaz dojde k načtení celé stránky. SPA přístup, na rozdíl od tradičního, využívá na straně klienta Javascript pro generování kusů kódu HTML a klient nikdy nepožaduje celou HTML stránku ze serveru, nýbrž jen konkrétní data, podle kterých pak sám upraví strukturu zobrazené stránky – klient tedy od spuštění aplikace pracuje pouze na jedné a té samé stránce [19].

Tento přístup je rozhodně pro účely webové verze Surmonu výhodnější, protože při práci s webovým rozhraním se bude vždy měnit pouze některá z částí uživatelského rozhraní, takže není potřeba posílat serveru požadavky na načtení celých stránek. To by navíc způsobovalo ztrátu dat o stavu na straně klienta. Z těchto důvodů jsem se rozhodl vytvořit webové rozhraní jako SPA.

2.6.1.1 Návrh komunikace se serverem v aplikaci Surmon

Webové rozhraní bude komunikovat se serverem pomocí asynchronních požadavků. V rámci této práce je mým úkolem navrhnout, jaké požadavky bude webové rozhraní serveru posílat a jakou podobu budou mít vyměňovaná data. Server by měl poskytovat REST API, jenže jelikož není serverová část aplikace v současné době hotová, budu pro testování funkčnosti webového rozhraní používat vlastní modelová data poskytovaná buďto z lokálního serveru nebo přímo z front-endu. Všechna vyměňovaná data budou ve formátu JSON.

Komunikaci jsem se snažil navrhnout tak, aby nebylo potřeba posílat na server zbytečně mnoho požadavků. Mezi případy komunikace se serverem nebudu zahrnovat požadavky týkající se uživatelských účtů, jelikož nejsou v rámci této práce prioritou. Navrhl jsem následující požadavky a podoby vyměňovaných dat (příklad komunikace na obrázku 2.9):

- **Získání definic bloků při načtení SPA** – aby bylo možné vytvářet instance bloků, musí nejprve webové rozhraní získat definice jednotlivých typů bloků včetně kategorií, do kterých patří
  - **URI** – http://doména/blockDefinitions
  - **požadavek** – operace GET, neposílá žádná data
  - **odpověd** – pole všech definic bloků
- **Načtení samotného projektu** – projekty budou požadovány podle jejich ID
  - **URI** – http://doména/projects/[project id]

  - **požadavek** – operace GET, neposílá žádná data
  - **odpověd** – objekt projektu, který bude kromě svých vlastností obsahovat pole s daty o všech v něm obsažených schématech

- **Načtení naposledy otevřených projektů**
  - **URI** – http://doména/projects/recent
  - **požadavek** – operace GET, neposílá žádná data
  - **odpověd** – server vrátí klientovi pole jím naposledy otevřených projektů se všemi jejich schématy

- **Vytvoření projektu** – na serveru je vytvořen a uložen nový projekt, který je poté zaslán klientovi
  - **URI** – http://doména/projects
  - **požadavek** – operace POST, na server se posílá objekt s požadovanými vlastnostmi nového projektu (pravděpodobně pouze název projektu)
  - **odpověd** – prázdný projekt s nastavenými požadovanými vlastnostmi

- **Uložení projektu**
  - **URI** – http://doména/projects/[project id]
  - **požadavek** – operace PUT (klient zná ID ukládaného projektu), posílanými daty je projekt s polem změněných schémat
  - **odpověd** – server nevrací žádná data (klienta po odeslání požadavku pouze zajímá, jestli byl požadavek úspěšně proveden či nikoliv)

- **Vytvoření schématu** – na serveru je vytvořeno a uloženo nové schéma, které je přiřazeno k danému projektu a poté je zasláno klientovi
  - **URI** – http://doména/projects/[project id]/schemes
  - **požadavek** – operace POST, posílanými daty je objekt s požadovanými vlastnostmi nového schématu (pravděpodobně pouze název schématu).
  - **odpověd** – prázdné schéma s nastavenými požadovanými vlastnostmi

- **Uložení schématu**
  - **URI** – http://doména/projects/[project id]/schemes/[scheme id]

  - **požadavek** – operace PUT, posílá schéma se všemi svými částmi, jako jsou bloky a hrany
  - **odpověd** – server nevrací žádná data

- **Smazání projektu** – současně smaže všechna schémata, která byla součástí projektu
  - **URI** – http://doména/projects/[project id]
  - **požadavek** – operace DELETE, neposílají se žádná data
  - **odpověd** – server nevrací žádná data

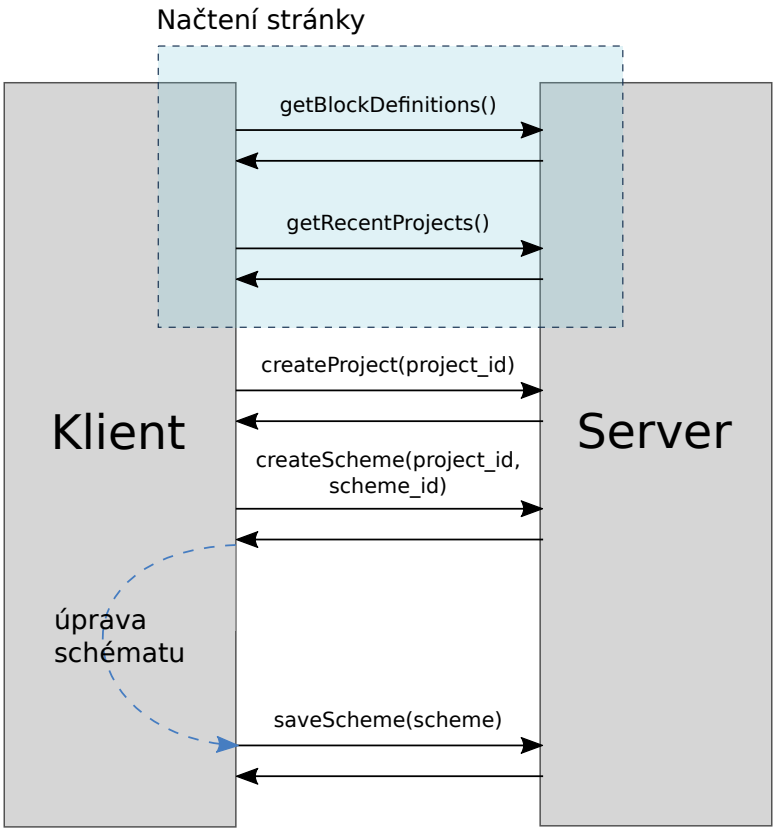
- **Smazání schématu** – analogický požadavek k požadavku na smazání projektu
  - **URI** – http://doména/projects/[project id]/schemes/[scheme id]
  - **požadavek** – operace DELETE, neposílají se žádná data
  - **odpověd** – server nevrací žádná data

- **Aktivace schématu**
  - **URI** – http://doména/schemes/[scheme id]/activate
  - **požadavek** – operace GET, neposílají se žádná data (použijí se data schématu uložená na serveru)
  - **odpověd** – server vrátí objekt, který mapuje výstupy aktivace na vlastnosti výstupních bloků schématu

- **Uložení a aktivace schématu** – tento požadavek se postará o uložení a následnou aktivaci schématu
  - **URI** – http://doména/schemes/[scheme id]/activate
  - **požadavek** – operace POST, pošle se schéma, které má být aktivováno
  - **odpověd** – server vrátí objekt, který mapuje výstupy aktivace na vlastnosti výstupních bloků schématu

K signalizování toho, jakým způsobem byl požadavek zpracován nebo zdali nastala chyba, by měl server vracet příznačné HTTP status kódy. Jako informační zdroj pro navržení toho, které návratové kódy bude REST API používat, jsem použil stránku „REST API Tutorial“ [20]. Používané kódy budou tyto:

- **200 (OK)** – úspěch; při poslání dat v odpovědi
- **204 (No Content)** – úspěch; při nezaslání žádných dat v odpovědi



Obrázek 2.9: Příklad komunikace front-endu aplikace Surmon s back-endem, která zahrnuje načtení stránky webového rozhraní a vytvoření projektu se schématem, které je pak změněno a uloženo na server.

- **201 (Created)** – úspěch; při vytvoření nové instance projektu nebo schématu
- **304 (Not Modified)** – při dotazu na zdroj, který nebyl změněn, takže se vezme cachovaná odpověď
- **404 (Not Found)** – neúspěch; například u požadavku týkajícího se neexistujícího projektu či schématu
- **405 (Method Not Allowed)** – neúspěch; naznačuje, že metoda odeslaného požadavku není pro dané URI povolená

2.6.2 Vnitřní architektura webového rozhraní

Zde se budu zabývat architekturou uvnitř samotného webového rozhraní. K tomuto účelu bych mohl použít nějaký z existujících front-endových frameworků, které v této části práce zhodnotím. Poté popíši vnitřní architekturu webového rozhraní, kterou jsem si zvolil.

2.6.3 Frameworky

Front-endové frameworky typicky umožňují vytvářet webová rozhraní, která mají vnitřní architekturu MVC a fungují jako SPA. Tyto frameworky jako je Angular [21], React [22], Knockout [23] nebo mnoho dalších mají své výhody. Díky nim je možné vytvářet komplexnější front-endy systematičtěji a rychleji, neboť typicky umožňují vytváření kódu ve formě modulů a samy se starají o mechanismy komunikace mezi moduly a o mechanismy vykreslování.

Frameworky však také mají podle [24] několik nevýhod. Zavádí určitou strukturu komunikace mezi komponentami, což může být i nevýhodou, jelikož některé věci probíhající uvnitř frameworku nemá vývojář možnost ovlivnit a změnit (nebo jen stěží). Jinými slovy, nejsou příliš flexibilní a vývojář pak nemá nad kódem takovou kontrolu. Dalším problémem jsou aktualizace frameworků, které mohou způsobit mnoho chyb v existujícím kódu. V neposlední řadě jsou frameworky obtížněji testovatelné a mohou pro testování vyžadovat další framework.

Frameworky také mohou mít problémy s rychlostí vykreslování DOM elementů (DOM budu používat pro označení objektové reprezentace HTML dokumentu, se kterou pracuje prohlížeč). Tato rychlost závisí na způsobu, kterým jsou elementy vykreslovány při změně datového modelu. Pokud by se při přidání jednoho prvku do modelu překreslil celý odpovídající seznam, bylo by vykreslování značně neefektivní. Co se týče zmíněných frameworků, tak ty implementují vykreslování lépe (hlavně React pomocí virtuálního DOMu [25]), nicméně úprava DOMu je i tak pomalejší, než ruční změna. Frameworky totiž musí při překreslování části uživatelského rozhraní nejprve zjistit, jak se model změnil a jak se má DOM překreslit, kdežto bez frameworku a se znalostí části DOMu, kterou je potřeba změnit, je možné provést úpravu s minimálním počtem operací.

2.6.3.1 Zvolený architektonický styl webového rozhraní

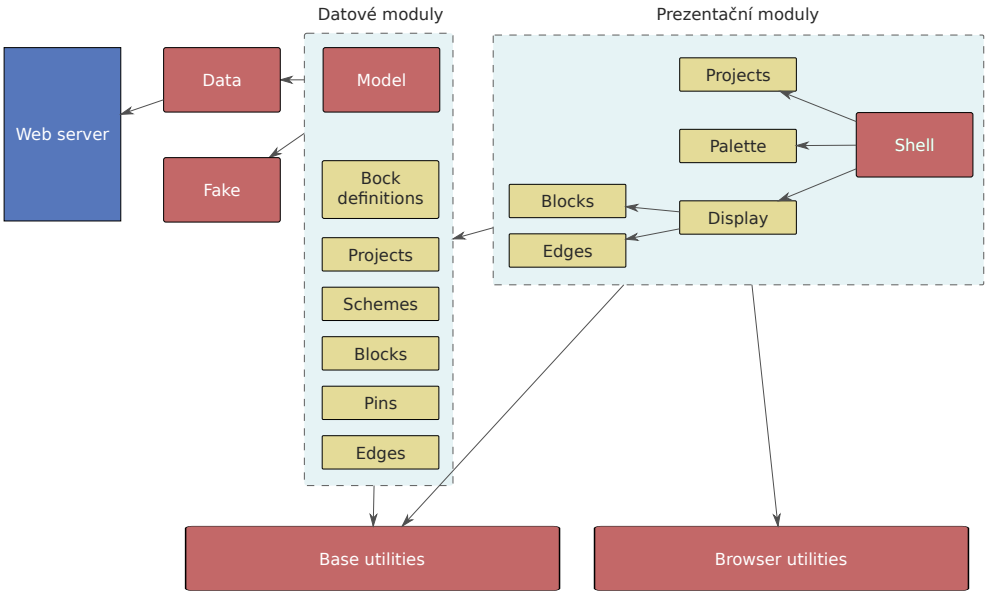
S ohledem na zmíněné vlastnosti frameworků jsem se rozhodl nepoužít žádný existující framework, ale architektonický styl pro tvorbu SPA, který je popsán v knize „Single page Web Applications“ [19]. Tento styl jsem zvolil také proto, že se už po několik let jeho autorům osvědčuje v praxi a lze s ním systematicky a nepříliš zdlouhavě vytvářet front-endy webových aplikací. V této části budu tedy popisovat, jak tento architektonický styl vypadá a jak jej aplikuji na webové rozhraní aplikace Surmon.

Podle mnou zvoleného architektonického stylu má být webové rozhraní rozděleno do modulů, které mohou být různých typů:

- **Prezentační** – Toto jsou hlavní moduly, jejichž úkolem je spravovat určitou část uživatelského rozhraní a implementovat funkcionalitu s ní spojenou. Tyto moduly jsou vlastně view a controller z MVC, neboť se starají o vykreslování částí uživatelského rozhraní a také obsahují funkce zpracovávající události, které řídí aktualizace modelu a viewu voláním příslušných metod. Prezentační moduly vytváří hierarchickou strukturu, ve které probíhá komunikace od rodičů k dětem a kde je kořenovým modulem takzvaný „shell“ modul. Ten se stará o globální funkcionalitu, jakou může být například správa cookies nebo manipulace s URI.
- **Datové** – Ve front-endu mají také být takzvané datové moduly, které spravují určité typy dat (objektů) aplikace a prezentační moduly je využívají pro získání dat k vykreslování. Datové moduly se v podstatě skládají ze dvou částí. Jednou je definice prototypu určitého typu objektu (například prototyp bloku), který definuje metody pro tento typ objektu. Druhou částí jsou funkce, které nepracují s konkrétními instancemi tohoto typu objektu, ale zabývají se typicky operacemi nad kolekcemi všech instancí daného typu objektu. Důležitou funkcí je vytvoření instance objektu, kde se aplikuje prototyp na nový objekt a inicializují se hodnoty jeho vlastností podle parametrů funkce.
- **Pomocné** – Součástí zvoleného architektonického stylu jsou též moduly poskytující pomocné funkce ostatním modulům. Tyto „pomocné“ moduly jsou dva, z nichž jeden poskytuje funkcionalitu úplně všem modulům a druhý pouze prezentačním modulům, protože pracuje s uživatelským rozhraním.
- **Komunikující s back-endem** – Posledním typem modulu je modul komunikující se serverem, který posílá požadavky back-endu a získaná data z odpovědí posílá datovým modulům, které je zpracují.

Architektura webového rozhraní se tedy bude skládat především z datové části (modelu) a z části zabývající se prezentací a delegováním událostí vyvolaných uživatelem. Architektura datové části bude vycházet z datového modelu, který je popsán v sekci „Data v aplikaci“. Za každý typ objektu datového modelu bude vytvořen jeden datový modul. Co se týče prezentačních modulů, tak jsem navrhl tyto moduly a jejich zodpovědnosti:

- **shell** – má na starost základní komponenty uživatelského rozhraní, kterými jsou hlavní menu, postranní panely a hlavní plocha pro tvorbu schémat
- **projects** – spravuje panel projektů v levém panelu aplikace



Obrázek 2.10: Diagram znázorňující diagram komunikace mezi moduly (červené a žluté bloky) a dalšími subjekty (modré prvky)

- **palette** – spravuje paletu stavebních bloků v pravém panelu aplikace
- **display** – má na starost vše, co se nachází mezi postranními panely, tedy záložky, menu schématu a plátno se schématem
- **blocks** – zabývá se operacemi nad bloky schémat a jejich piny
- **edges** – zabývá se operacemi nad hranami schémat

Na diagramu z obrázku 2.10 jsou vidět všechny moduly, které jsem pro webové rozhraní navrhl a také komunikace mezi nimi. V tomto diagramu nejsou zobrazeny zcela všechny cesty komunikace, protože by to nebylo příliš přehledné. Šipky v diagramu znamenají volání metod modulů a směr proti šipkám zaznamenává tok dat.

Komunikace znázorněná na obrázku 2.10 probíhá tak, že se hierarchicky inicializují prezentační moduly počínaje shellem, který v inicializaci předává reference na datové moduly svým potomkům a ty pak volají různé metody těchto datových modulů. „Model“ modul je hlavním datovým modulem, přes který jsou inicializovány ostatní datové moduly. Komunikace mezi datovými moduly odpovídá vztahům mezi typy objektů, které jsou danými moduly reprezentovány. Metody datových modulů pracují buď synchronně (provedou akci a pak se běh programu vrátí k volajícímu) nebo asynchronně, v případě potřeby komunikace se serverem, kdy po dokončení AJAX požadavku datový

modul pošle hromadnou zprávu a každý modul, který má zaregistrován posluchač pro tento druh zprávy, pak tuto zprávu zpracuje.

Model dále komunikuje s takzvaným „data“ modulem, který posílá požadavky na webový server a získává odpovědi, které předá modelu zavoláním modelem předložené funkce. Alternativou pro data modul pro jednoduší testování front-endu je „fake“ modul, který simuluje server tak, že přímo v sobě obsahuje definované instance různých objektů a ty předává modelu. Tím pádem není pro testování nutně potřeba žádný server.

Jména modulů uvedená v této sekci jsou pouze zjednodušená. V rámci realizace budu podle rad v knize „Single page Web Applications“ [19] pojmenovávat moduly ve webovém rozhraní tak, že jméno modulu bude obsahovat zřetězená jména jeho rodičovských modulů a předponu „surmon“. Tedy například „surmon.display.blocks“. Modul shell se do názvů prezentačních modulů nepřidává a to nejspíš proto, že z něj vychází všechny ostatní prezentační moduly, a tudíž by nijak blíže nespecifikoval daný modul. Předpona „surmon“ poukazuje na projekt, ke kterému modul patří. K datovým modulům se navíc přidává předpona „model“, takže například „surmon.model.projects“, aby bylo jasně vidět, že se jedná o jiný typ modulu.

2.7 Prostředky pro tvorbu webového rozhraní

Před implementací front-endu je potřeba analyzovat, jaké jazyky, knihovny a aplikace jsou k dispozici pro vytvoření webového rozhraní a také dokumentů, které jsou též součástí této práce.

2.7.1 Technologie

Nejprve zde budu popisovat volbu implementačních jazyků, jejich knihoven, popřípadě frameworků. Co se týče jazyků pro implementaci funkcionalit webového rozhraní, tak zde tak veliký výběr není. Pro tyto účely by se dal využít Flash, Javascript nebo teoreticky i Java. Nicméně jazyk Javascript je v poslední době mnohem kompetentnější, než dříve, poskytuje veliké množství různých funkcí a API, a navíc má oproti Flashi a Javě tyto výhody [19]:

- pro jeho běh není potřeba žádných pluginů do prohlížečů
- potřebuje méně prostředků než zmíněné pluginy vyžadující run-time prostředí
- použití jednoho jazyka pro celý front-end
- je v prohlížečích nativní a nezavádí nekonzistence při interakci se stránkami

Ze zmíněných důvodů tedy použiji jazyk Javascript pro implementaci funkcionalit webového rozhraní. Jazyk Javascript využiji i na straně serveru za účelem vytvoření testovacího back-endu pro simulované obsluhování požadavků z webového rozhraní. Použiji k tomu node.js (s Express frameworkem), díky kterému je hlavně jednoduchá práce s formátem JSON, neboť Javascript je s ním schopen nativně pracovat. Webové rozhraní bude díky komunikaci se simulovaným serverem připraveno komunikovat s jakýmkoliv reálným serverem, který bude poskytovat stejné REST rozhraní, jako testovací server.

Dalšími jazyky pro tvorbu uživatelského rozhraní jsou HTML a CSS, jejichž použití je samozřejmostí. Nicméně ve spoustě webových aplikacích, hlavně ve SPA, je HTML kód generován pomocí Javascriptu, popřípadě jsou fragmenty HTML kódu psány do proměnných v Javascriptu, ale nikoliv do samostatných HTML dokumentů. Mnou vytvářený front-end bude pracovat s HTML kódy stejnými způsoby.

Co se týče CSS, tak využiji při implementaci webového rozhraní některé prvky CSS 3. Tato verze CSS přináší nové věci, jako jsou přechody, animace, počítání hodnot pomocí funkce „calc()“, definice webových fontů a spoustu dalších [26], některé z nichž se mohou hodit pro zjednodušení práce programátora nebo pro vylepšení uživatelského zážitku.

Pro usnadnění zápisu CSS pravidel se používají CSS preprocesory, jako jsou SASS [27] a LESS [28]. Tyto preprocesory umožňují vývojářům efektivněji definovat styly pomocí rozšířených konstrukcí. Napsané dokumenty v jazycích preprocesorů musí být napřed zkompilovány do jazyka CSS, takže vzniknou CSS dokumenty, které jsou pak použity pro stylování vykreslených elementů. Nutnost kompilace kódu do CSS sice zpomaluje vývoj, ale tato nevýhoda je mizivá oproti výhodám vyplývajícím z přehlednosti a zjednodušení psaní definic stylů. Pro účely této práce využiji SASS, protože je rozvinutější a rozšířenější.

Jako formát pro webovou grafiku rád používám SVG, což je vektorový formát v jazyce XML. SVG je možné přímo vkládat do kódu jazyka HTML a SVG elementy lze stylovat pomocí CSS a lze s nimi manipulovat v Javascriptu skoro stejně jako s HTML elementy. Tento formát jsem se rozhodl použít pro reprezentaci ikon akcí v uživatelském rozhraní a také pro reprezentaci hran propojujících bloky. Pro vykreslování hran schématu (popřípadě celého schématu) by se ještě dal využít element canvas, ale ten obecně není příliš vhodný pro aplikace, kde je hodně využívána interakce pomocí myši.

Pro komunikaci se serverem budu využívat technologii AJAX, kde vyměňovaná data budou ve formátu JSON. Požadavky posílané na back-end budou používat metody GET, POST, PUT a DELETE.

2.7.1.1 Knihovny

Pro Javascript existuje nejen spousta knihoven, ale i polyfillů pro zvýšení kompatibility napříč prohlížeči. Nejznámější knihovnou je jQuery [29], která

se stala v podstatě standardem pro implementaci front-endů v případech, kdy není použit framework. Tato knihovna usnadňuje práci s objekty, zpracováním událostí, posíláním AJAX požadavků a hlavně s manipulací s DOM elementy. Proto tuto knihovnu využiji i v této práci. Existuje také takzvaná „jQuery UI“ [30] knihovna, která nabízí k využití různé komponenty uživatelského rozhraní – například dialogy, akordióny, posuvné elementy, přechodové efekty a mnoho dalších. Počítám s tím, že některé z těchto nabízených komponent použiji.

Existují některé knihovny pro Javascript, které poskytují rozhraní pro tvorbu diagramů, což je vlastně zobecnění tvorby blokových schémat, čímž se Surmon zabývá. Veliká část těchto knihoven však není kompatibilní s požadavky na webové rozhraní, hlavně pak s požadavkem na přesný počet a přesné umístění pinů bloků a také proto, že potřebuji mít kontrolu nad implementačními detaily tvorby diagramů. Jednoduše řečeno, tyto knihovny nejsou dostatečně flexibilní nebo efektivní, abych je mohl využít.

Jednou z takovýchto knihoven je JointJS [31], která poskytuje rozhraní pro vytváření nástrojů pro tvorbu diagramů. Avšak tato knihovna je rozsáhlá, a navíc vyžaduje zahrnutí několika jiných Javascriptových knihoven, což by pravděpodobně zpomalovalo práci s aplikací, a kromě toho bych stejně většinu funkcionality této knihovny nevyužil. Jednodušší knihovnou je jsPlumb [32], která je co do velikosti rozhodně menší a pracuje velmi efektivně. Přesto však není zcela vhodná pro tuto práci kvůli výše zmíněným důvodům.

Zaměřil jsem tedy svou pozornost na knihovny poskytující pouze část funkcionality tvorby diagramů. Mezi takovéto knihovny patří třeba Interact.js [33], což je knihovna zabývající se především posouváním elementů na plátně a škálováním elementů. Nicméně stejné funkcionality je možné dosáhnout použitím komponent knihovny jQuery UI, což je výhodnější z toho hlediska, že jsou komponenty konzistentní s knihovnou jQuery, kterou k implementaci použiji. Vhodnými komponentami jquery UI by pro tuto práci mohly být:

- **draggable** – umožňuje posouvání elementů, omezení posunu v rámci určitého elementu, pohybování se po mřížce a další
- **droppable** – nastavení určitého elementu jako cíle přetažení jiného elementu
- **resizable** – přidání úchytů pro škálování zvoleného elementu

Existují i kvalitní jQuery moduly vytvořené komunitou, z nichž některé se zabývají propojováním elementů pomocí hran ([34], [35]). Tyto moduly však typicky kreslí hrany do jednoho <svg> elementu, což znemožňuje implementovat, aby se hrany a bloky mohly navzájem překrývat. V každém případě bych potřeboval mít nad implementací hran větší kontrolu, a proto jsem se rozhodl implementovat vlastní spojování elementů pomocí hran.

Kromě zmíněných Javascriptových knihoven použiji ještě knihovny gevent [36], TaffyDB [37] a možná také uriAnchor [38]. Gevent a uriAnchor jsou moduly pro jQuery – gevent se zabývá vytvářením vlastních událostí (implementace vzoru Publish-subscribe) hlavně pro účely spuštění nějaké operace po provedení asynchronní akce a uriAnchor usnadňuje manipulaci s fragmenty URI, což se může hodit pro implementaci historie ve front-endu. TaffyDB je knihovna umožňující vytvářet kolekce objektů, které fungují jako tabulky relační databáze, tedy je například možné vyhledávat objekty v kolekcích podle určité vlastnosti nebo také používat agregační operátory nad výsledky dotazu.

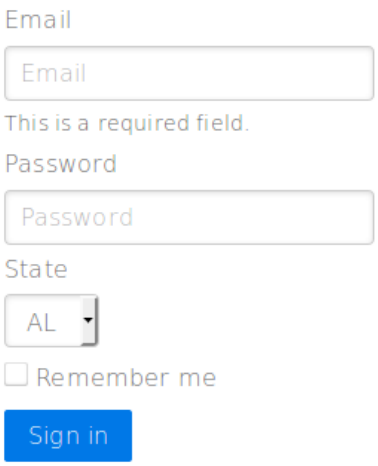
Vzhledem k tomu, že součástí uživatelského rozhraní jsou i dialogy, bylo by vhodné k jejich vytváření použít knihovnu. Tvorbu dialogů jsem se rozhodl vyřešit knihovnou Vex [39], kterou už jsem v minulosti zkoušel a která je flexibilní v tom, že umožňuje vývojářům definovat různé typy dialogů a jejich obsahy. Tato knihovna je též jednoduchá na použití, a navíc poskytuje předdefinované styly pro dialogy, takže není třeba vytvářet vlastní.

Uvažoval jsem o použití nějaké knihovny/frameworku, který se zabývá především prezentací (tedy jazyky HTML a CSS). Mezi tyto frameworky patří Bootstrap [40], Foundation [41], Material Design Lite [42] a mnoho jiných. Zmíněné frameworky se rozhodně hodí pro rychlou tvorbu vizuálně konzistentních stránek, ale zase do aplikace zavádí své způsoby pojmenování tříd elementů. Také je těžší přebít CSS selektory definované v těchto frameworcích a obsah si podle vlastních představ přizpůsobit. Z těchto důvodů nepoužívám takovéto frameworky často, nicméně jsem narazil na jeden framework jménem Pure.css [43] (obrázek 2.11), který je velmi malý a poskytuje jen základní styly a komponenty uživatelského rozhraní, které se dají jednodušeji přizpůsobit. Pravděpodobně Pure.css použiji pro aplikování základních stylů na elementy a poté vytvořím vlastní styly, které určí finální vzhled elementů.

Co se týče ikon, rozhodl jsem se použít nějakou knihovnu pro vkládání SVG ikon do uživatelského rozhraní pomocí CSS. Vybral jsem si Fontello [44], což je aplikace pro vygenerování CSS šablony z ikon, které si uživatel vybere z rozsáhlé nabídky v aplikaci. Jsou zde nabízeny ikony z různých zdrojů (například ze známého Font Awesome [45]), nicméně některé ikony, které by se mi hodily, jsem nenalezl, takže jsem si vybral ikony jim podobné. Možná bude potřeba udělat si své vlastní ikony nebo sehnat ikony z jiného zdroje, ale to by mělo vyplynout z uživatelského testování.

2.7.1.2 Aplikace

Pro realizaci této práce budu používat určité aplikace pro různé účely. Část těchto aplikací slouží především k tvorbě obrázků či diagramů. Mezi tyto grafické aplikace patří Inkscape [46], Draw.io a Pencil project [47]. Inkscape slouží k tvorbě vektorové grafiky, která je vhodná pro web, nicméně využívám Inkscape také ke kresbě ilustrační grafiky do různých dokumentací. Draw.io je primárně nástroj pro tvorbu diagramů, čehož využiji v rámci dokumentací, i



Obrázek 2.11: Příklad formuláře vytvořeného s pomocí Pure.css

když pro menší diagramy jsem zvyklý používat opět Inkscape. Pencil project je užitečnou aplikací pro kreslení wireframů a je oproti mnoha podobným aplikacím zcela zdarma, a proto jsem ho použil v návrhu uživatelského rozhraní pro vytvoření wireframů.

Dále použiji některé aplikace, které usnadňují správu projektu a pomáhají automatizovat složité operace nad projektem. Jednou z těchto aplikací je Bower [48], který pomáhá vývojářům spravovat front-endové knihovny v projektu stahováním a aktualizováním knihoven z předdefinovaných repozitářů. Další aplikací je Grunt [49], který umožňuje automatizovat různé úpravy projektu a práci se soubory projektu. Grunt čerpá funkcionalitu z různých svých modulů. Moduly, které jsem zvyklý používat, umožňují například kontrolovat použití standardů Javascript kódu, automaticky načítat stránky po změně zdrojových souborů, komprimovat zdrojové soubory a další.

Jelikož součástí této práce bude i dokumentace, hledal jsem vhodnou aplikaci pro její jednoduché vytvoření. Nalezl jsem program jménem JSDoc [50], pomocí kterého je možné vytvořit dokumentaci ke kódu standardním způsobem přes komentáře přímo v kódu. Tyto komentáře popisují funkce či jiné části kódu nebo různé abstraktní prvky (například vyvolávané události). JSDoc může používat uživateli vytvořené šablony pro zobrazení vygenerované dokumentace, která je ve formátu HTML.

2.7.2 Shrnutí použitých prostředků

V této části shrnuji, které všechny prostředky jsem zvolil pro realizaci mé práce. U jazyků zmiňuji použití knihoven a u aplikací zmiňuji, jakou hrají při vytváření této práce roli.

- **HTML**
- **CSS** – použití knihoven Pure.css a Fontello pro ikony
- **SASS** – preprocesor pro jazyk CSS
- **SVG** – škálovatelná grafika, reprezentace hran
- **Javascript** – použití knihoven jQuery, gevent a uriAnchor moduly do jQuery, resizable, draggable a droppable moduly pro knihovnu jQuery UI, knihovna taffyDB
- **Grunt** – automatizace procesů pro ulehčení vývoje a sestavovacích procesů
- **Bower** – správa front-endových modulů
- **Inkscape** – tvorba webové grafiky a grafiky pro dokumentační materiály
- **JSDoc** – tvorba dokumentace
- **Pencil project** – tvorba wireframů
- **Draw.io** – tvorba diagramů do dokumentací

Kapitola 3

Realizace

Tato kapitola se zaměřuje na popis toho, jak jsem postupoval při implementaci webového rozhraní aplikace Surmon, a jaký je výsledek mé práce. Budu zde popisovat svůj postup od založení projektu přes různé realizační detaily až po dokončení práce. Jako součást popisu realizace také uvedu výsledky uživatelského testování a porovnám vytvořené výsledné webové rozhraní s existující desktopovou aplikací. Nebude zde ani chybět rozbor možných vylepšení webového rozhraní do budoucna.

3.1 Založení a nastavení projektu

Nejprve se zde zaměřuji na první část realizace projektu, kterou je jeho založení. Popíši zde, jak jsem si rozvrhl adresářovou strukturu a kam jsem umístil jednotlivé typy souborů. Poté se zaměřím na popis získávání externích knihoven a také na popis modulů aplikace Grunt, které jsem použil pro zjednodušení vývoje webového rozhraní.

První věcí, kterou jsem zahájil tvorbu projektu, bylo samozřejmě vytvoření adresářové struktury. Kořenový adresář projektu jsem rozdělil na dva podadresáře – jeden pro vývojářskou verzi projektu a druhý pro produkční verzi. V obou podadresářích je pak struktura stejná. Je zde umístěn kořenový HTML dokument webového rozhraní a složka „assets“ s ostatními zdrojovými soubory, která je členěna na obrázky, CSS styly, Javascriptové zdrojové kódy a SASS dokumenty.

V kořenovém adresáři jsou také umístěny definiční soubory pro aplikace Bower a Grunt. Pomocí Boweru jsem postupně stahoval knihovny potřebné pro implementaci front-endu. Tyto externí knihovny jsou všechny umístěny pod jednou složkou v kořenovém adresáři. Některé knihovny (například Fontello nebo jQuery UI) se však typicky nestahují kompletní, ale vývojář si vybere pouze některé části knihovny. V těchto případech není možné použít Bower, takže jsem takovéto knihovny musel stáhnout manuálně a umístil jsem je do složky v kořenovém adresáři zvlášť.

Pomocí modulů Gruntu jsem nastavil automatické akce, které se hodí při vývoji projektu, popřípadě pro vygenerování produkční verze z vývojářské verze. Začal jsem definováním akce pro automatickou aktualizaci všech oken prohlížeče se spuštěnou aplikací při změně zdrojového kódu, díky čemuž není potřeba znovu načítat stránku manuálně. Kromě toho jsem definoval akci pro překlad SASS dokumentů do CSS dokumentů po uložení nějakého SASS dokumentu. Mimo těchto akcí, které stále běží na pozadí jsem definoval jednorázové akce, jako je kontrola SASS, CSS a Javascriptového kódu na použití osvědčených praktik a standardů.

V rámci práce jsem neřešil vygenerování produkční verze webového rozhraní, nicméně jsem již stáhl a nastavil některé Grunt moduly pro tento účel a naplánoval jsem, jaké operace budou pro vygenerování produkční verze potřeba. Nezbytné bude minimalizovat zdrojové soubory (hlavně CSS a Javascript soubory), a tyto minimalizované soubory pak spojit do jednoho jediného souboru pro každý jazyk zvlášť. Složka se SASS dokumenty v produkční verzi vůbec nemusí být, neboť se jedná pouze o technologii pro usnadnění vývoje. Kvůli minimalizaci a zmenšení počtu zdrojových souborů bude nutné vygenerovat nové kořenové HTML soubory, které budou tuto změnu reflektovat a budou stahovat pouze nové minimalizované zdrojové soubory.

3.2 Prototyp uživatelského rozhraní a tvorba stylů

Ještě předtím, než jsem začal s vytvářením modulů a jejich metod, soustředil jsem se na vytvoření základního prototypu uživatelského rozhraní. Proto v této části budu popisovat, jak jsem tento prototyp vytvářel, přičemž se hlavně zaměřím na to, jak jsem definoval styly, které jsou na prototyp aplikovány, a jejichž rozšířením vznikne výsledná podoba uživatelského rozhraní.

Tvorbu prototypu jsem započal založením HTML dokumentu, do kterého jsem zpočátku vložil HTML strukturu celého uživatelského rozhraní (tedy všech jeho komponent). To proto, abych mohl předem vytvořit základní styly pro rozmístění komponent na obrazovce podle wireframů, které jsem vytvářel v rámci návrhu uživatelského rozhraní. Zatím jsem však nechtěl definovat konkrétní barvy nebo zcela finální podobu uživatelského rozhraní, ale snažil jsem se o přehledné rozmístění a zobrazení komponent a informací. Tento prototyp je vlastně základním kamenem uživatelského rozhraní, na kterém jsem dále stavěl. HTML dokument prototypu jsem později rozdělil do několika částí, kde každá část je spravována určitým prezentačním modulem.

Styly jsem vytvářel v jazyce SASS, který jsem pomocí modulu Gruntu překládal do CSS. Využil jsem schopnosti jazyka SASS importovat jeden SASS soubor do druhého a založil jsem několik souborů, z nichž jeden je hlavní a do něj jsou importovány všechny ostatní. Díky tomu se při překladu do CSS vygeneruje jen jeden soubor, který je pak referencován z hlavičky kořenového HTML souboru webového rozhraní. Některé SASS soubory jsou pojmenovány

podle prezentačního modulu, pro jehož část uživatelského rozhraní definují styly, například „_surmon.display.blocks.scss“ označuje soubor, který se vztahuje k modulu „surmon.display.blocks“. Podtržítko před názvem znamená, že tento soubor je součástí nějakého jiného SASS souboru (je do něj importován). Vytvořil jsem také několik souborů, které nejsou pojmenovány podle prezentačních modulů. Jsou to tyto soubory:

- **surmon.abstractComponents.scss** – Definice stylů selektorů reprezentující abstraktní komponenty. Jejich využívání je podobné dědění tříd v programovacích jazycích – ostatní selektory mohou dědit styly těchto selektorů abstraktních komponent. Dědičnost stylů selektorů může mít své nevýhody, jak je uvedeno v [51], takže se vyplatí dědit pouze styly abstraktních komponent, které jsou definovány na jednom místě a nikde jinde se v SASS kódu nevyskytují.

- **surmon.globalModifiers.scss** – Obsahuje definice stylů pro třídy, které pochází z knihoven, a také definuje základní styly pro třídy, které se využívají ve více komponentách uživatelského rozhraní.

- **surmon.mixins.scss** – Styly zde definované se nepoužívají samostatně, nýbrž jen jako součástí nějakých jiných selektorů. Mixiny definuji, když chci přidat nějaký styl do více selektorů.

- **surmon.variables.scss** – Obsahuje definice všech proměnných, které jsou v SASS souborech používány. Typicky se proměnné používají hlavně pro barvy, ale hodí se třeba i pro definici odsazení, které má být stejné pro více komponent. Díky tomuto souboru stačí změnit hodnoty proměnných na jednom místě a po přeložení souboru se hodnoty vloží do všech míst, kde byla proměnná použita.

Co se týče využití funkcionalit jazyka SASS, tak nejsou mé stylovací dokumenty příliš bohaté. Typicky při psaní SASS kódu používám proměnné, mixiny, dědičnost, ale hlavně zanořování selektorů, což usnadňuje přemýšlení nad definovanými pravidly jako nad hierarchií stylů. Jako u názvů modulů a jiných souborů, dal před názvy CSS tříd a identifikátorů předponu, která však není „surmon“, ale pouze „sur“, aby to bylo kratší a přehlednější, jelikož se tahle předpona vyskytuje v dokumentech velmi často. Názvy tříd a identifikátorů se skládají z více částí, které mají přibližovat umístění elementů, na které bude třída aplikována. Například název **sur-palette-category** naznačuje, že tato třída se týká elementů reprezentující kategorii, které se nachází uvnitř komponenty paleta. Někdy by řetězec slov oddělených pomlčkou byl tak dlouhý, že jsem některé komponenty na nižší úrovni vynechával. Na obrázku 3.1 je ukázka mého SASS kódu, kde jsou vidět názvy tříd a identifikátorů, proměnné a zanořování selektorů.

```
#sur-display-controls {
    background-color: $distinct-color;
    height: $menu-height;

    .sur-switch {
        padding: 0.3em 0.8em;
        margin-top: 0.2em;
        margin-left: 0.3em;

        &.sur-active {
            color: $text-color-light;
            background-color: $active-bg-color;
        }
    }
}
```

Obrázek 3.1: Kus SASS kódu definující styly pro menu nad plátnem schématu. Výrazy začínající znakem „$“ jsou SASS proměnné.

3.3 Tvorba modulů

V této sekci popisuji, jak jsem postupoval při tvorbě modulů a uvádím charakteristiky modulů a jejich částí. Nejdříve se zaměřím na základní charakteristiky společné pro všechny moduly a na jejich životní cyklus. Následně budu popisovat prezentační moduly a HTML struktury, které vykreslují, a také se zaměřím na to, jak je vykreslování uživatelského rozhraní ovlivněno datovým modelem. Nakonec přejdu k datovým modulů a objektům, se kterými pracují.

Každý modul má svůj vlastní javascriptový soubor, v jehož kódu je modul reprezentován objektem, který zapouzdřuje funkcionalitu daného modulu. Soubory modulů se jmenují stejně jako moduly v nich definované (například „surmon.display.blocks.js“). Moduly mají veliké množství funkcí, některé z nichž však nemohou být z vnějšku volány (jako kdyby byly například v Javě deklarovány jako private), čehož je docíleno pomocí takzvané uzávěry – při spuštění kódu modulu se spustí funkce, uvnitř které jsou definovány metody modulu, ale jen některé z nich jsou uloženy do objektu, který je ze spuštěné funkce vrácen a uložen do proměnné, která reprezentuje modul. Díky tomu mohou být tyto metody z vnějšku volány na objektu modulu, kdežto zevnitř modulu je možné volat i všechny ostatní metody. Na obrázku 3.2 je zjednodušeně znázorněn kód souboru modulu (konkrétně prezentačního).

Všechny moduly jsou připraveny ukládat si konfigurační a stavová data. Podle knihy Single Page Web Applications [19] jsem každému modulu vytvořil objekt zapouzdřující konfigurační proměnné a objekt zapouzdřující stavové proměnné. Také jsem do front-endu zavedl mechanismus pro nastavování konfigurovatelných proměnných. Do konfigurace spadají proměnné, jejichž hodnota je známa při inicializaci modulu a jejichž hodnota se po inicializaci nikdy nezmění. Stavové proměnné naopak mohou být na začátku nedefinované (nebo mají hodnotu null) a mohou se za běhu aplikace měnit. Konfigurační pro-

```
surmon.palette = (function () {

    var configMap = {
        /* konfigurační data */
    };

    var stateMap = {
        /* stavová data */
    };

    var jqueryMap = {
        /* reference na části DOMu */
    };

    /*
     *
     * definice metod
     *
     */

    return {
        /* metody, které mají být veřejné (mohou být volány zvenku) */
    };

}());
```

Obrázek 3.2: Na tomto obrázku je vidět zjednodušený kód souboru „surmon.palette.js“.

měnné jsou nastaveny buď přímo v modulu, anebo jsou nastaveny rodičem, který modul inicializuje. Takto bývají nastavovány typicky funkce zpětného volání, díky kterým mohou moduly volat metody definované ve svých rodičích.

3.3.1 Životní cyklus modulů

Všechny moduly mají svůj životní cyklus, který je rozdělen do tří po sobě jdoucích kroků, a ty vypadají takto:

1. **Vytvoření modulu** – Modul vzniká po dokončení spuštění kódu javascriptového souboru, ve kterém je modul definován. Poté je možné volat funkce na objektu modulu, ale modul zatím není schopen samostatně pracovat, protože ještě nemá nastaveny konfigurační proměnné potřebné pro správný běh a nemá zaregistrovány funkce pro zpracování vyvolávaných událostí.

2. **Konfigurace** – Rodičovský modul, který má nějaké potomky, je zodpovědný za jejich konfiguraci. Ta je realizována zavoláním funkce modulu potomka, do které se předají parametry, které potomek potřebuje k běhu a ke komunikaci s rodičem nebo ostatními moduly skrze rodiče. Komunikace se sourozeneckými moduly je konfigurací umožněna právě tak, že rodičovský modul předá potomkovi metodu jeho sourozence jako parametr konfigurace a potomek si metodu v sobě uloží. Poté může tuto metodu normálně volat. Po konfiguraci jsou moduly plně funkční, jen stále nereagují na žádné události.

3. **Inicializace** – V této fázi se vykreslí část uživatelského rozhraní, kterou modul spravuje (pokud nějakou spravuje), konfigurují a inicializují se potomci modulu a nakonec se zaregistují posluchače na události. Z tohoto popisu vyplývá, že aby se mohl modul dostat do fáze inicializace, musí se do této fáze napřed dostat jeho rodič. Nejvyšším rodičem je modul shell, jehož inicializace je zavolána po načtení všech javascriptových souborů. Po inicializaci už modul normálně funguje a jeho funkcionalitu mohou používat ostatní moduly.

Inicializaci u datových modulů jsem implementoval mírně odlišně. Místo hierarchické inicializace jsou všechny moduly inicializovány přímo z nejvyššího datového modulu. Vztahy mezi moduly jsou stále hierarchické, jen inicializace modulů je „plochá“. Důvodem k takovéto implementaci je spíše určitá logika, než optimalizace nějakého typu. Prezentační moduly zodpovídají za část uživatelského rozhraní, které bude vždy zobrazeno uvnitř rodičovské části uživatelského rozhraní. Pokud by totiž byl nejdříve inicializován modul potomka, došlo by k chybě, minimálně při vykreslování odpovídající části uživatelského rozhraní. Datové moduly oproti tomu nejsou takto svázány, proto není potřeba je inicializovat hierarchicky.

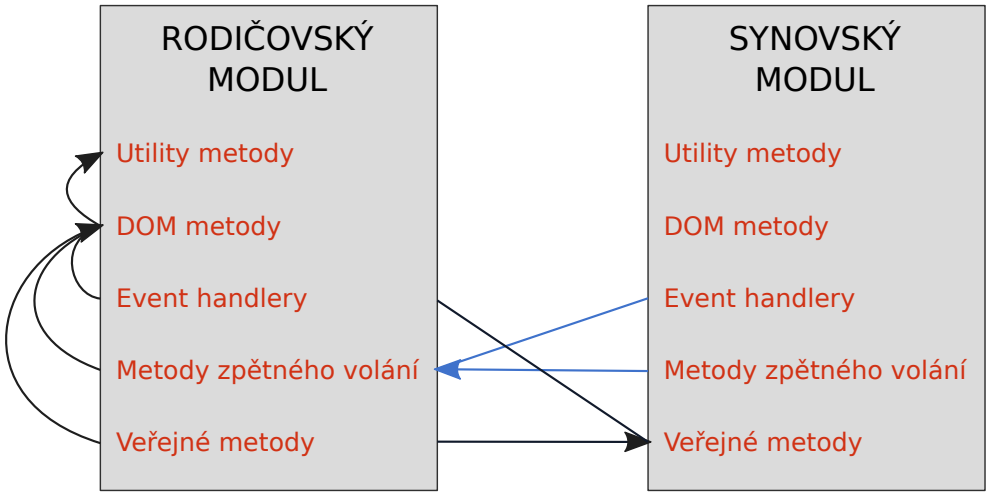
Jakmile je modul inicializován, už se nikdy nevrátí do žádného předchozího stavu. To proto, že ve webovém rozhraní aplikace Surmon není potřeba implementovat restartování modulů. V každém případě by bylo možné vytvořit funkci, která by vrátila inicializaci modulu zpět zrušením registrace posluchačů událostí, smazáním vykreslené části uživatelského rozhraní a vrácením konfiguračních hodnot do původního stavu. Dokonce by bylo možné celý modul úplně zrušit, nicméně v tomto webovém rozhraní si vystačím s životním cyklem, který jsem v této sekci popisoval.

3.3.2 Prezentační moduly

Prezentační moduly jsou hlavní součástí webového rozhraní, neboť řídí aktualizace modelu a jsou zodpovědné za vykreslování částí uživatelského rozhraní. Tyto moduly obsahují různé typy metod pro specifické účely, které zde budu popisovat. Hlavně se však zaměřím na práci prezentačních modulů s DOMem a také na popis samotných HTML struktur, ze kterých jsou vytvářeny části uživatelského rozhraní, a jejichž odpovídající DOM strukturu moduly vykreslují a aktualizují.

3.3.2.1 Typy metod prezentačních modulů

Metody prezentačních modulů jsou rozděleny do pěti kategorií podle své úlohy. V kódu jsou metody podle těchto kategorií uspořádány a pro každou kategorii je před jejími funkcemi komentář nadepisující danou kategorii. Metody jsou děleny do těchto kategorií:



Obrázek 3.3: Na obrázku je znázorněn typický model volání metod uvnitř prezentačního modulu a mezi moduly rodiče a potomka pomocí šipek.

- **Utility metody** – Jsou to pomocné metody, které nepracují s DOMem. Tyto metody mohou například provádět nějaký pomocný výpočet potřebný pro vykreslování hran schématu (konkrétně třeba zjištění kvadrantu, ve kterém se nachází cílový bod hrany vzhledem k počátečnímu).
- **DOM metody** – Jsou zodpovědné za tvorbu a aktualizaci prezentace uživateli. Tyto metody jsou typicky volány z metod zpracovávající události (event handlery).
- **Event handlery** – Metody sloužící ke zpracovávání událostí vyvolaných uživatelem nebo datovými moduly po dokončení asynchronní komunikace se serverem. Tyto metody jsou klíčové pro umožnění interakce uživatele s uživatelským rozhraním, a také pro reflektování výsledků komunikace se serverem v uživatelském rozhraní.
- **Metody zpětného volání** – Toto jsou metody, které rodičovský modul předává svým potomkům ve fázi konfigurace, aby mohli potomci tyto metody „zpětně“ volat.
- **Veřejné metody** – Jedná se o metody, které jsou modulem zvěřejněny, a tím pádem mohou být volány rodičovskými moduly, pod které spadají.

Tyto metody různých typů se navzájem volají, což je znázorněno šipkami na obrázku 3.3. Tyto znázorněné směry volání jsou typické, ale nemusí být nutně pouze tak, jak je to vidět na obrázku, neboť například některé DOM metody mohou být taktéž veřejné. Tím pádem by mohla na obrázku vést šipka skoro ze všech typů metod do veřejných metod.

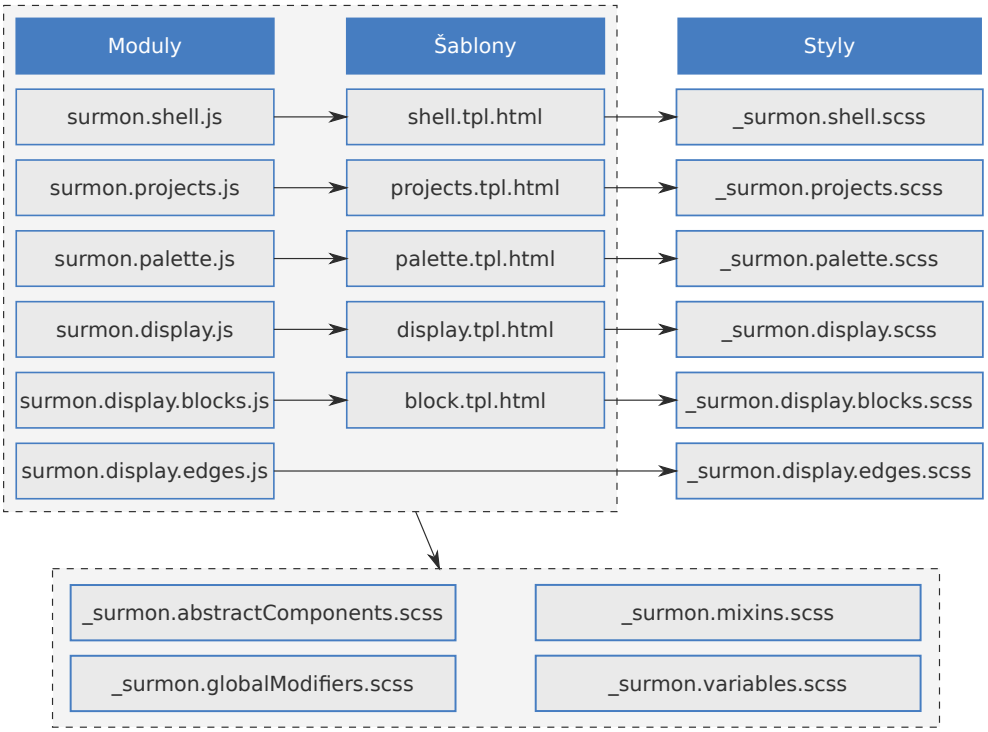
3.3.2.2 Vykreslování uživatelského rozhraní

Pro jednotlivé prezentační moduly je potřeba vytvořit HTML struktury různých komponent uživatelského rozhraní, se kterými moduly pracují. Vzhledem k tomu, že jsem nechtěl vytvářet HTML přímo v kódu Javascriptu, protože je to časově náročnější a není možné využít nástrojů programovacího prostředí pro psaní kódu HTML, rozhodl jsem se, že vytvořím samostatné HTML soubory a poté je pomocí nějakého nástroje přetransformuji do textového řetězce, který bude uložen do javascriptové proměnné. Pro tento účel jsem nechtěl použít žádný kompletní šablonovací systém jako je například Handlebars [52], protože tyto systémy se také samy starají o překreslování uživatelského rozhraní, což typicky provádí neefektivně. Raději jsem použil modul do Gruntu, který po spuštění přemění vytvořený HTML dokument na textový řetězec a uloží ho do proměnné v kódu Javascriptu. Konkrétně se řetězce ukládají jako hodnota vlastnosti objektu, jejíž název je shodný s názvem HTML šablony.

Vytvořil jsem tedy několik HTML dokumentů, které reprezentují určitou část uživatelského rozhraní. Tyto HTML dokumenty (nebo také šablony) mohou reprezentovat buď celou část uživatelského rozhraní, kterou některý z modulů má na starost, anebo se může jednat o nějakou menší část, třeba o samotný blok, kterých je v uživatelském rozhraní více. Když se při inicializaci prezentačního modulu vykresluje komponenta uživatelského rozhraní reprezentovaná určitým HTML dokumentem převedeným do textového řetězce, je potřeba DOM vytvořený z daného HTML fragmentu vložit do již existujícího DOMu dokumentu. Z tohoto důvodu jsem si vytvořil takový mechanismus, že do HTML šablony odpovídající rodičovskému modulu vložím element s názvem například **<dummy-palette/>**, který indikuje pozici, kam má být do DOMu vložen DOM HTML fragmentu potomka (v tomto případě DOM palety stavebních bloků). Pak na začátku inicializace modulu potomka tento element nahradím DOMem HTML fragmentu potomka.

Na obrázku 3.4 je diagram vizualizující vztahy mezi soubory modulů, HTML šablonami a SASS dokumenty. Šipky směrem od modulů k šablonám znamenají, že modul pracuje s částí uživatelského rozhraní, kterou šablona reprezentuje. Šipky od šablon k SASS dokumentům ukazují, ze kterého SASS dokumentu využívají elementy v šabloně styly. Pokud vede šipka od modulu přímo k SASS dokumentu, znamená to, že kód tohoto modulu vytváří DOM elementy bez použití HTML šablony, a tyto elementy využívají styly daného SASS dokumentu. V podstatě by každý modul mohl mít šipku navíc přímo k SASS dokumentům. Nakonec jsou v diagramu zobrazeny i ostatní dokumenty definující styly, které mohou být využity z jakéhokoliv modulu a v jakékoliv HTML šabloně.

DOM elementy vytvářené v modulech přímo bez použití HTML šablon jsou typicky nějaké položky seznamu, kterých je předem neznámý počet a které jsou spojeny s nějakým objektem z datového modelu. Například v levém panelu se vykresluje seznam položek otevřených projektů a každá položka se



Obrázek 3.4: Diagram znázorňující vztahy mezi zdrojovými soubory modulů, HTML šablon a CSS stylů.

„odkazuje“ na konkrétní objekt projektu, který reprezentuje, pomocí uchovávání si jeho identifikátoru v konkrétním HTML atributu. Když pak proběhne akce ze strany uživatele, která se týká položky seznamu otevřených projektů, vezme se identifikátor z HTML atributu položky a podle něj se vyhledá příslušný objekt projektu, se kterým se pracuje dále.

Prezentační moduly vykreslují data, která jsou uložena v datovém modelu (tedy ve všech datových modulech). Vykreslovanými daty jsou:

- otevřené projekty
- schémata všech otevřených projektů
- bloky otevřeného schématu a tedy i všechny jejich piny a hrany, které bloky přes piny propojují
- všechny definice bloků, které jsou umístěny v paletě v pravém panelu

Webové rozhraní je zodpovědné za aktualizaci zobrazených dat v případě jejich změny v datovém modelu. Tyto změny mohou mít jednu ze tří podob. Může se jednat o přidání nové položky do modelu nebo o změnu či smazání

existující položky (například projektu). Všechny typy změn by se daly implementovat tak, že by se po změně překreslila celá komponenta uživatelského rozhraní (například panel projektů), které se změna týká, včetně všech jejích položek. Toto je neefektivní přístup, jelikož vyžaduje mnoho operací s DOMem, které jsou časově náročné. Proto jsem se rozhodl implementovat tyto změny zvlášť, aby se vždy provedly jen ty operace s DOMem, které jsou nutné.

V prezentačních modulech se oproti ostatním modulům nachází objekt, do kterého jsou na začátku inicializace modulu uloženy reference na různé části DOMu, které má modul ve správě. Jsou to části, ke kterým bude v budoucnu potřeba přistupovat a manipulovat s nimi. Díky ukládání referencí během inicializace pak už není nutné znovu a znovu získávat reference na části DOMu v případech, kdy je třeba s nimi pracovat. Jedná se tedy o optimalizační mechanismus. Reference na části DOMu jsou uchovávány v podobě jQuery kolekcí (jsou to objekty obalující reference na DOM).

3.3.3 Moduly datového modelu

Zde se budu zabývat popisem vytvořených datových modulů – konkrétně tím, jakým způsobem fungují, co umí a jak vytváří instance objektů, se kterými se dále ve webovém rozhraní pracuje a které jsou zobrazovány v rámci vykreslování a aktualizace uživatelského rozhraní. Více do detailu se budu zabývat vytvářením instancí bloků. Nakonec také zmíním možnosti implementace schémat jako superbloků.

Moduly datového modelu mají odlišnou funkci oproti prezentačním modulům. Stahují data ze serveru, uchovávají je a poskytují k nim přístup ostatním modulům. Jednotlivé instance objektů jsou v modulech uchovávané jednak v globálních kolekcích modulů, kam jsou vkládány úplně všechny instance daného typu objektu, a pak jsou také instance ukládány do svých rodičovských objektů (například objekt schématu má kolekci svých bloků). Také jsem uvnitř datových modulů vytvořil některé stavové proměnné, které mají jako svou hodnotu instanci daného typu objektu (například aktivní blok). Pro vytváření kolekcí objektů jsem použil knihovnu TaffyDB a definoval jsem v modulech funkce pro získání uchovávaných objektů podle různých jejich atributů. Tyto funkce jsou volány především prezentačními moduly, které sbírají potřebná data k vykreslování.

Hlavními atributy, podle kterých jsou uchovávané objekty získávány, jsou identifikátoy UUID a UUCID. UUID je globálně unikátní identifikátor, který by měl být unikátní v rámci celého serveru. UUCID se liší v tom, že je unikátní pouze v rámci dat na front-endu. Některé objekty identifikátor UUID ani nemají – jsou jimi piny, hrany a definice. Ve webovém rozhraní používám hlavně UUCID, který by teoreticky nebylo nutné vůbec používat, protože jsem komunikaci se serverem navrhl tak, že všechna data ve webovém rozhraní mají vygenerováno UUID ještě před tím, než je možné s nimi pracovat. Idea UUCID byla právě v tom, že se s ním bude pracovat, dokud se ze serveru

nezíská UUID. Nicméně díky UUCID by se dalo uživatelům částečně umožnit pracovat s webovým rozhraní i bez připojení k internetu, kdy není možné získat UUID ze serveru.

UUID se normálně generuje na serveru při vytváření nových objektů. Výjimku do tohoto pravidla zavádí bloky, jejichž UUID je generováno na straně front-endu, aby nebylo nutné při vytvoření každého bloku komunikovat se serverem. Vytváření UUID ve front-endu je přijatelné díky tomu, že data na serveru nebudou ukládána do relační databáze a stačí, když bude UUID unikátní alespoň v rámci daného schématu, aby mohla fungovat aktivace schématu.

3.3.3.1 Vytváření instancí objektů

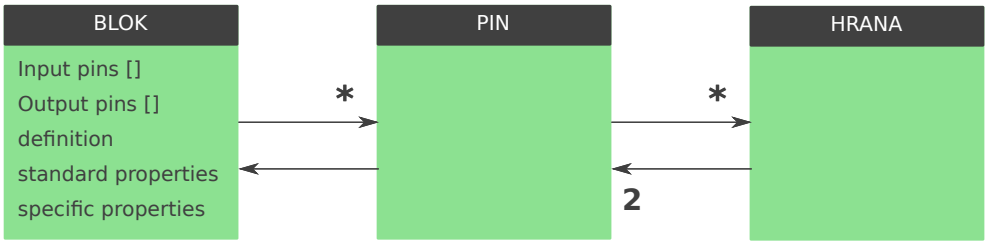
Vytváření instancí objektů je důležitou součástí každého datového modulu. Budu zde popisovat, jak takovéto vytváření instancí probíhá a jak se vytvořené instance liší od podoby objektů, které jsou vyměňovány v rámci komunikace se serverem. Na závěr popíši, jak se liší vytváření instancí bloků od vytváření instancí ostatních typů objektů kvůli potřebě práce s definicemi bloků.

Každý datový modul má v sobě metodu s názvem „make“, která slouží jako továrna (factory), ve které se vytváří nové instance objektu s daným prototypem a poté se inicializují vlastnosti objektu podle předaných parametrů. V rámci této tovární funkce je objekt také uložen do lokální kolekce (TaffyDB databáze), kde jsou uloženy všechny instance tohoto typu objektu. U některých modulů jsem implementoval více továrních funkcí, ale princip je vždy takový, že se v rámci těchto funkcí provede nějaký úkon navíc a pak se zavolá základní tovární funkce „make“.

Při vytváření instancí objektů se vytvoří i instance jejich podobjektů. Celý řetězec vytváření instancí vypadá tak, že se vytvoří instance projektu a pokud má v sobě nějaká schémata, spustí se tovární funkce pro tato schémata. V nich se zase vytvoří instance bloků, dále v blocích instance pinů a nakonec se vytváří hrany. Z tohoto řetězce vyplývá, že při vytváření instance nějakého typu objektu v sobě musí předané parametry tovární funkce obsahovat také informace o všech podobjektech, které do objektu patří, a to na jakékoliv úrovni v hierarchii.

V kapitole o návrhu jsem popisoval vztahy mezi typy objektů. Tyto vztahy jsem se rozhodl pro účely implementace rozšířit, aby se objekty snáze používaly. Datový model tak jak jsem ho implementoval, je vidět na obrázku 3.5. Rozdíl oproti původně navrženému modelu je ten, že hrany mají reference na oba koncové piny a piny mají referenci na svůj rodičovský blok. Díky těmto novým vztahům je možné jednoduše traverzovat napříč schématem přes hrany a bloky v jakémkoliv směru. Také jsem neimplementoval dědičnost pinů, zejména kvůli velmi lehkým implementačním rozdílům mezi vstupními a výstupními piny.

Objekty mohou mít dvojí podobu. První podoba, kterou nazývám „zdrojová“, je ta, která se využívá pro výměnu dat mezi front-endem a back-endem. Tato



Obrázek 3.5: Na tomto diagramu je zobrazena část implementačního datového modelu, která tento model odlišuje od původně navrženého modelu.

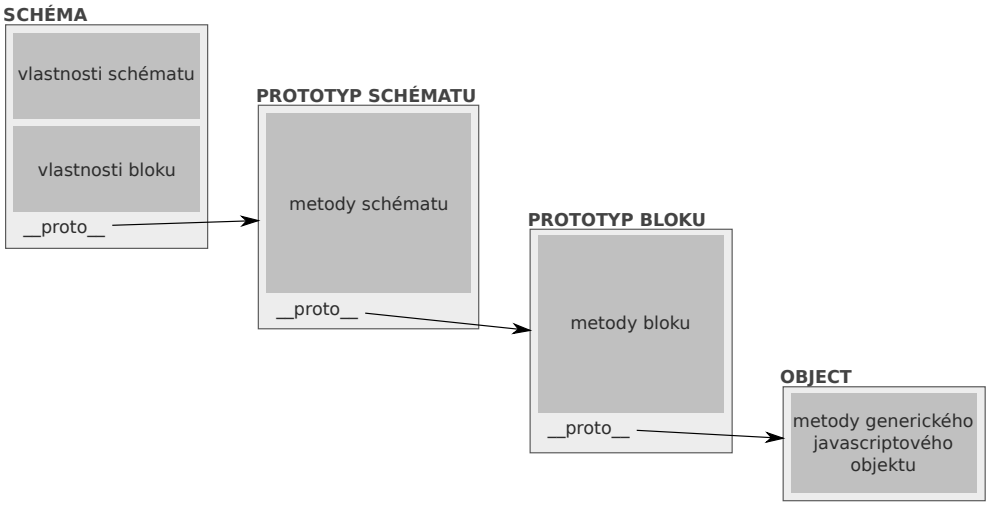
podoba odpovídá datovému modelu, který jsem uvedl v kapitole návrhu. Data v této podobě posílá server front-endu, a proto je potřeba je na front-endu napřed zpracovat, aby odpovídala implementačnímu datovému modelu (přidáním nových vztahů). Objekty ve zdrojové podobě (nazývám jí také mapa objektu) jsou hlavními parametry do již zmíněné funkce make. V této funkci se vytvoří nový objekt s daným prototypem a poté se na něj aplikují vlastnosti z mapy objektu. Některé pomocné vlastnosti a reference na další objekty se přidají navíc. Tak vznikne „implementační“ podoba objektu.

Zvláštním případem z hlediska vytváření objektů je blok. Bloky se vytváří za přítomnosti nejen mapy bloku, ale i příslušné definice bloku. Nejprve se vytvoří základní struktura bloku podle vlastností jeho definice. Mapa bloku není povinným parametrem, ale pokud je předána, tak se poté nahradí hodnoty vlastností základní struktury bloku hodnotami stejnojmenných vlastností z mapy bloku. Když se vytváří nový blok, použije se pouze definice, ale pokud jde o nějaký již existující blok získaný typicky ze serveru, předá funkci „make“ se i mapa bloku.

Každý vytvořený blok si uchovává referenci na svou definici, která poskytuje třeba informace o tom, jakého jsou určité vlastnosti typu, což se hodí při vykreslování kontrolek pro editaci vlastností. Bloky mají v implementační podobě (na rozdíl od zdrojové podoby) rozděleny vlastnosti do skupin implementovaných jako vnitřní objekty v bloku. Jedna skupina zahrnuje standardní vlastnosti společné pro všechny bloky. Druhá skupina má v sobě specifické vlastnosti daného bloku, které jsou ještě dále rozděleny na vstupní a výstupní vlastnosti. Vstupní vlastnosti se dají upravovat uživatelem, ale výstupní slouží pouze jako informace podané uživateli a vzniknou aktivací schématu.

3.3.3.2 Schémata jako superbloky

Během řešení datového modelu webového rozhraní je potřeba se vypořádat s implementací schémat jako speciálních bloků, tedy superbloků. Schémata, jelikož mají zároveň roli superbloků, musí nějakým způsobem mít také funkcionalitu normálních bloků. To jsem implementoval pomocí řetězce prototypů tak, že prototyp schémat má také svůj vlastní prototyp, a tím je právě pro-



Obrázek 3.6: Diagram zobrazující řetězec prototypů u instancí schémat. Je vidět, že objekty prototypů obsahují pouze metody, nikoliv vlastnosti.

totyp bloků. Se schématy se tedy dá zacházet jako s bloky a jejich „bloková“ část je také inicializována stejným způsobem jako bloky. Dědičnost (řetězení prototypů) u schémat je vyobrazena na obrázku 3.6. Superbloky jsou jedním z aspektů řešené problematiky, jehož implementace není v této práci prioritou. Proto bylo mým cílem implementaci superbloků spíše jen připravit, aby bylo možné jí v budoucnu dodělat.

U superbloků je třeba se potýkat s tím, že každé schéma může být vloženo do více jiných schémat nebo dokonce vícekrát do stejného schématu. Tím pádem je vlastně potřeba mít více kopií jednoho schématu, jednu za každý superblok. To je možno realizovat tak, že po přidání schématu do jiného schématu se vkládané schéma zduplikuje a jeho kopie bude mít referenci na bloky originálního schématu, aby nebylo potřeba uchovávat kopie všech bloků schématu. Alternativou je způsob řešení, který by zahrnoval vytvoření nového typu objektu „superblok“. Instance tohoto typu objektu by dědily vlastnosti bloků, a navíc by obsahovaly referenci na schéma, které reprezentují. Vykreslené bloky schémat na plátně by pak reprezentovaly instance superbloků. Tyto postupy jsem zde uvedl jako návrhy na implementaci, ale žádný z nich jsem v rámci této práce neimplementoval.

3.4 Implementace práce se schématem

V této části se zaměřím na popis svého postupu při implementaci centrální části webového rozhraní, kterou je vytváření schématu a práce se schématem. Uvedu zde jednotlivé kroky v chronologické posloupnosti, které vedly k výsledné realizaci zmíněné části webového rozhraní. Mezi tyto kroky patří

například zavedení propojování bloků pomocí hran, umožnění přidávání bloků z palety na plátno nebo editace vlastností bloků. U jednotlivých kroků zmiňuji komponenty knihovny jQuery UI, které jsem použil.

Začal jsem tím, že jsem v HTML kódu umístil na plátno pár statických bloků a nejprve jsem implementoval tažení bloků. To bylo snadné, jelikož jsem měl k dispozici knihovnu jQuery UI, ze které jsem použil komponentu „draggable“. Všechny bloky jsem učinil posouvatelnými zavoláním funkce „draggable“ na jQuery kolekce bloků s vhodnými parametry. Díky této komponentě je také možné definovat funkce, které se volají na začátku, na konci nebo během tažení bloku.

Když už byly bloky pohyblivé, nastal čas umožnit propojování jejich pinů hranami. Jak jsem v kapitole návrhu zmínil, propojování bloků a vykreslování hran jsem si implementoval sám. Implementace je zatím jednoduchá a hrany jsou zcela rovné a nijak se nevyhýbají objektům na plátně. Hrany jsem implementoval obecně tak, že mohou propojovat jakékoliv dva DOM elementy, nicméně jimi vždy propojuji pouze piny. Pro prezentaci hran jsem využil SVG element „path“, který umožňuje vykreslení hran s různými zakřiveními a zlomy. Aby se hrany mohly aktualizovat s pohybem bloků, přidal jsem do nastavení komponenty „draggable“ u bloků funkci, která při každém pohybu bloku a také po skončení pohybu změní atributy hrany tak, aby její průběh odpovídal nové pozici jejích koncových pinů.

Piny bloků jsou propojovány tažením hrany pomocí levého tlačítka myši z výstupního pinu jednoho bloku nad vstupní pin druhého bloku a následným uvolněním stisknutí tlačítka myši. Tažení hrany z výstupního pinu na vstupní pin jsem implementoval pomocí neviditelného pomocného pinu. Po kliknutí na výstupní pin nějakého bloku se přemístí pomocný pin na pozici ukazatele myši a vytvoří se hrana mezi výstupním a pomocným pinem. Pak se až do uvolnění tažení hrany pohybuje pomocný pin s ukazatelem myši a jelikož je tento pin neviditelný, vypadá to, jako že je hrana tažena myší. Při skončení tažení se tažená hrana smaže a pokud se ukazatel myši nachází nad vstupním pinem nějkého bloku, vytvoří se hrana mezi výstupním pinem a tímto vstupním pinem.

Poté jsem přesunul svou pozornost na paletu bloků. Vytvořil jsem pár definic bloků, kterými jsem zaplnil paletu. Opět jsem se uchýlil k použití jQuery UI, a to abych mohl jednoduše implementovat přijímání tažených bloků z palety do plátna. Na jQuery kolekci plátna se zavolá funkce „droppable“, která umožňuje definovat, jaký typ elementů bude plátno přijímat a co se má stát po uvolnění tažení přijímaného elementu nad plátnem.

Aby se daly bloky a hrany z plátna mazat nebo aby bylo možné provádět s nimi nějaké jiné operace, vytvořil jsem pro bloky i hrany kontextové menu. Toto kontextové menu se objeví po kliknutí na blok či hranu (tedy po označení bloku nebo hrany). U bloků je menu umístěné přímo dovnitř jejich obsahu, takže se automaticky pohybuje spolu s blokem při přesouvání bloku. U hran je kontextové menu posazeno doprostřed hrany a jeho pozice musí být samo-

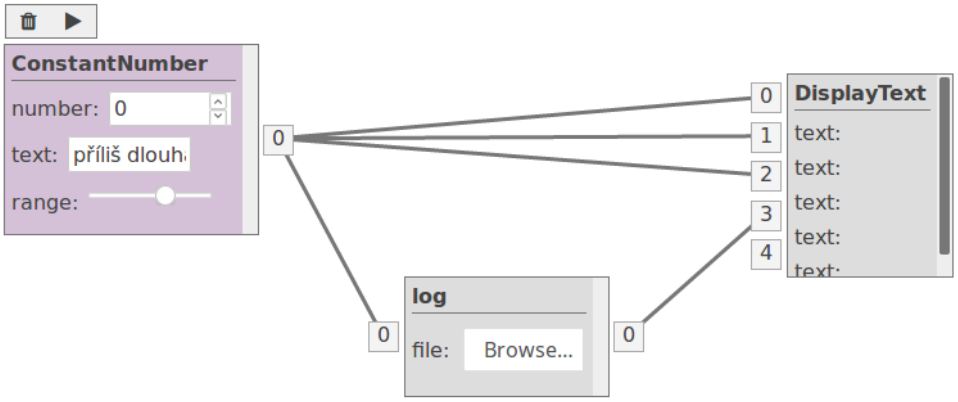
statně aktualizována při každém posunutí hrany (menu není vloženo do DOM elementu hrany).

Dále jsem se soustředil na umožnění upravování vlastností bloků. „Sumon.display.blocks“ modul má kromě zobrazování bloků na starost i zobrazování jejich vlastností v pravém panelu v komponentě panelu vlastností, ke které má modul přístup. V této komponentě se vykresluje seznam vlastností označeného bloku v definovaném pořadí. Nejprve se zobrazují standardní vlastnosti – název bloku, třída (označuje Java třídu použitou na back-endu pro daný typ bloku), UUID a tooltip. Pak jsou zobrazeny specifické vlastnosti daného bloku. Ty vlastnosti, které jsou editovatelné, mají k sobě zobrazen widget pro editaci. Při zobrazování vlastností jsem se potýkal s délkou jejich hodnot. Některé hodnoty (hlavně UUID) se do jim připraveného místa nevejdou. Zatím jsem tento problém vyřešil nastavením tooltipů k hodnotám vlastností bloku a hodnotu vlastnosti třída jsem zkrátil tak, že je z ní zobrazen jen samotný název třídy bez namespacu. Alternativním řešením by mohlo být po najetí myši na vlastnost roztáhnout celý řádek s názvem i hodnotou vlastnosti tak, aby byla poté vidět celá hodnota vlastnosti.

Vlastnosti bloků se mají zobrazovat nejen v pravém panelu, ale i v samotných blocích. V blocích jsou však zobrazeny pouze specifické vlastnosti. Velikost bloku na plátně musí být zobrazeným vlastnostem přizpůsobena, aby nebylo potřeba v obsahu bloku scrollovat. Při implementaci editovatelných vlastností bloku je důležité, aby se aktualizoval datový model při změně hodnoty nějaké vlastnosti. Aktualizovat se však musí i prezentace, jelikož editace hodnot vlastností se musí projevit na všech místech, kde je vlastnost zobrazena nebo je její hodnota nějak používána při vykreslování uživatelského rozhraní. Když jsou tedy vlastnosti bloku současně zobrazeny v pravém panelu i v bloku samotném, je nutné jejich zobrazené hodnoty synchronizovat, aby nedošlo k nekonzistencím. Při úpravě hodnoty vlastnosti se identifikuje název změněné vlastnosti, který je uložen v atributu „data-prop“ elementu reprezentující vlastnost. Díky tomu může být v datovém modelu aktualizována hodnota dané vlastnosti bloku. Poté se najdou místa v uživatelském rozhraní, jejichž obsah závisí na hodnotě upravené vlastnosti, aby jejich obsah mohl aktualizován. Všechna tato místa se dají nalézt díky tomu, že mají na svém elementu také nastaven atribut „data-prop“ na název dané vlastnosti.

Jelikož by bloky měly být škálovatelné, znovu jsem použil knihovnu jQuery, která poskytuje komponentu „resizable“. Ta umožňuje k vybranému elementu přidat úchyty pro jeho roztažení či zmenšení. Přidal jsem použitím této komponenty škálovací úchyty na všechny strany bloku i do všech rohů bloku. Tyto úchyty jsou v normálním stavu skryté, ale po najetí myši nad ně se zviditelní. Bloky nelze zmenšit pod danou mez, jelikož by se jinak úchyty překrývaly, nicméně by stejně neměl být důvod bloky do takovéto míry zmenšovat.

Abych vylepšil práci s bloky schématu, zavedl jsem řazení bloků a hran na pomylsné ose „z“, což určuje, jak se bloky a hrany navzájem překrývají. Překrývání jsem implementoval tím způsobem, že označený blok je vždy nejvýše,



Obrázek 3.7: Na tomto obrázku je zachyceno schéma, kde je několik propojených testovacích bloků.

pak jsou bloky, které s ním sousedí, pak zase jejich sousedé, a takto až do nejkrajnějších bloků. Nejníže jsou bloky, které nejsou na označený blok nijak napojené (ani přes jiné bloky). Tato implementace má nevýhodu, že je celkem složitá, ale přesto funguje poměrně rychle. Alternativou by mohlo být to, že by se pořadí bloků určovalo historií označování bloků (nedávno označené bloky by byly výše).

Po implementaci všech těchto funkcionalit byl modul **surmon.display.blocks** příliš obsáhlý, a proto jsem se rozhodl vyčlenit z něj nový modul, který by převzal část jeho funkcionality. Nazval jsem ho **surmon.display.blocks.props**, a svěřil jsem mu veškeré operace a interakce s vlastnostmi bloků, ať už uvnitř samotných bloků nebo v panelu vlastností. Kvůli tomuto novému modulu jsem také vytvořil novou HTML šablonu „blockProps.tpl.html“ obsahující HTML strukturu panelu vlastností. S touto šablonou a s její odpovídající DOM strukturou pak props modul pracuje.

Na obrázku 3.7 je ukázka podoby schémat po dokončení implementace práce se schématem. Na obrázku je vidět propojení bloků, kontextové menu nad označeným blokem a zobrazení vlastností uvnitř bloků.

3.5 Komunikace mezi moduly

Moduly spolu musí komunikovat, aby bylo možné synchronizovat stavy částí uživatelského rozhraní, které moduly spravují, a aby mohly navzájem využívat své funkcionality. V následujícím seznamu jsou vypsány konkrétní implementované případy komunikace mezi moduly. Inicializace modulů a komunikace mezi datovými moduly nejsou do tohoto seznamu zahrnuty. Šipky mezi názvy modulů znamenají, že modul na levé straně volá funkce modulu na pravé straně.

- **surmon.shell → surmon.display** – Shell modul při změně velikosti postranních panelů volá funkci display modulu pro změnu velikosti plochy mezi postranními panely, aby nadále zabírala veškerý prostor mezi panely.

- **surmon.palette → surmon.display** – Paleta musí předat display modulu elementy reprezentující definice bloků z palety bloků. To kvůli tomu, aby se v display modulu mohla inicializovat plocha pro přijímání definic bloků z palety pomocí jQuery UI komponenty „droppable“. Po každé, když se přidají nějaké položky definic bloků do palety, je potřeba předat elementy těchto nových definic display modulu, aby mohly tyto definice být vkládány do plochy pro tvorbu schématu.

- **surmon.projects → surmon.display** – Tato komunikace probíhá skrze shell modul, který musí při inicializaci předat projects modulu funkce z display modulu. Projects modul komunikuje s display modulem ve dvou případech. První případ se týká otevření a zobrazení schématu na plátně při kliknutí na položku schématu z panelu projektů. Vzhledem k tomu, že plocha pro zobrazení schématu je součástí display modulu, musí mezi moduly proběhnout komunikace, aby se mohlo schéma zobrazit a aby se vytvořila jeho záložka (pokud ještě není vytvořena). Druhý případ je podobný a jedná se o smazání schématu použitím akce v panelu projektů. Display modul v tomto případě musí zrušit záložku daného schématu a musí schéma zavřít, pokud je zobrazené.

- **surmon.display → surmon.projects** – Implementoval jsem i opačnou komunikaci, a to z display do projects modulu. Aby panel projektů podával uživateli zpětnou vazbu konzistentní se zobrazenými záložkami, musí být projects modul informován o označování a zavírání záložek, které patří do display modulu. Díky tomu pak projects modul může zvýraznit správnou položku v panelu projektů.

- **surmon.display → surmon.display.blocks** – Display modul v mnoha případech volá funkce blocks modulu. Tato komunikace nastává, když je potřeba provést nějakou úpravu schématu, jelikož bloky jsou důležitou součástí schématu. Příklady této komunikace zahrnují volání vykreslení nebo smazání bloků schématu, vytvoření nového bloku z bloku přetaženého do plátna nebo deaktivace aktivního bloku při kliku někam na plátno.

- **surmon.display → surmon.display.edges** – Tato komunikace je podobná jako v předchozím případě. Display modul komunikuje s edges modulem za účelem vytváření, mazání nebo deaktivace hran.

- **surmon.display.blocks → surmon.display.edges** – Co se týče komunikace, je blocks modul dosti propojen s edges modulem. Při práci

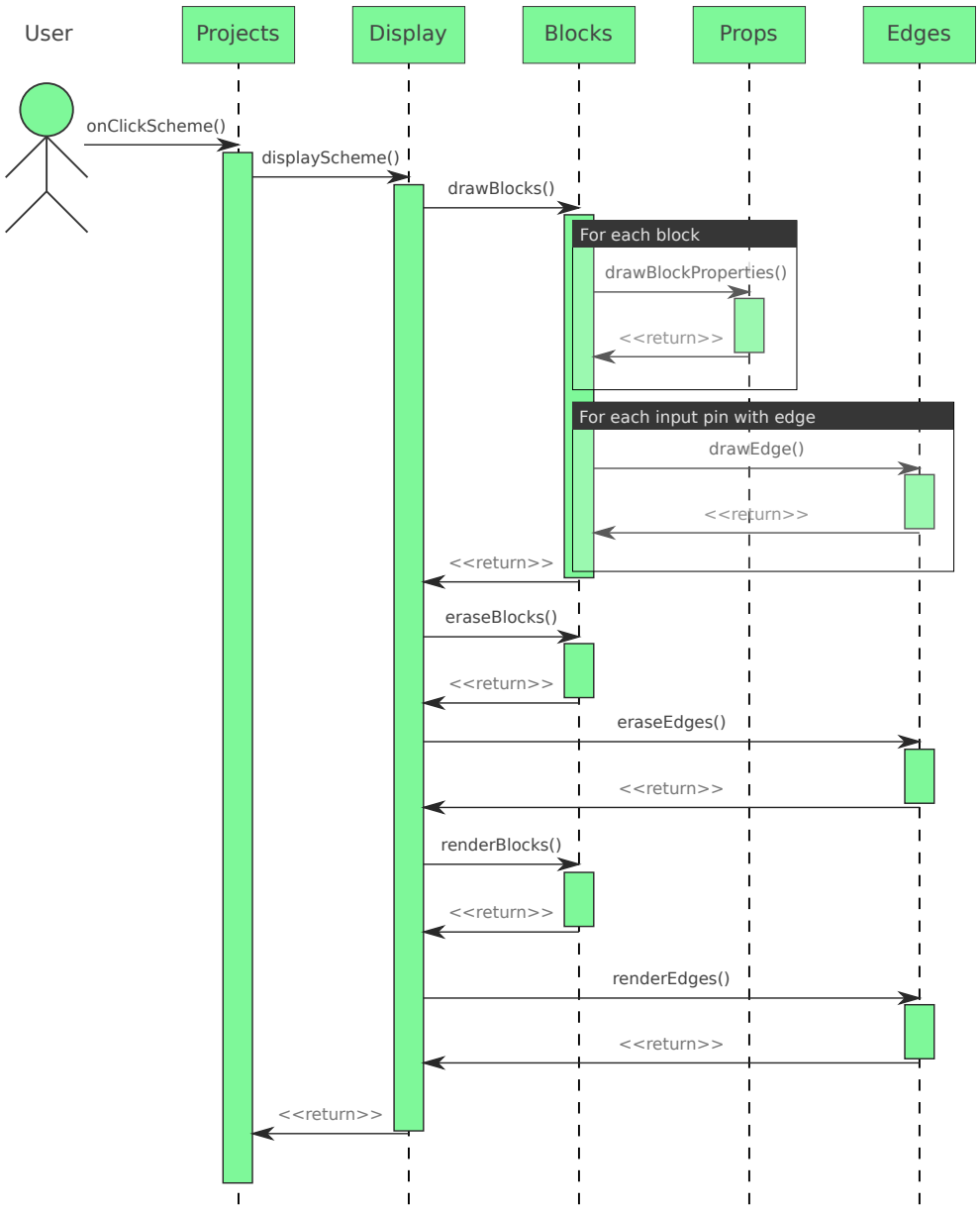
s bloky je totiž často potřeba upravovat hrany, které jsou s bloky spojeny. Například je nutné edges modulu signalizovat potřebu propojení dvou pinů bloků, rozpojení hran kvůli smazání bloku nebo deaktivace označené hrany po označení bloku.

- **surmon.display.edges → surmon.display.blocks** – Edges modul komunikuje s blocks modulem, když je potřeba deaktivovat označený blok po označení hrany nebo když dojde k rozpojení hrany, a v důsledku toho je nutné smazat koncový pin hrany, který vznikl duplikací.
- **surmon.display.blocks → surmon.display.blocks.props** – Tato komunikace probíhá tehdy, když se mají zobrazit vlastnosti nějakého bloku buď uvnitř něj nebo samostatně v pravém panelu. Toto nastává, když je samotný blok vykreslován nebo když je označen.
- **surmon.display.blocks.props → surmon.display.blocks** – Props modul potřebuje v některých případech vědět, jaký blok je v uživatelském rozhraní zrovna označen, což zjišťuje voláním funkce blocks modulu, která mu vrátí jQuery kolekci označeného bloku. Dále signalizuje potřebu překreslení hran u bloků, které se zvětší následkem aktivace kvůli načtení obrázku do bloku jako výsledku aktivace (obrázek je výstupní vlastností bloku). Tato komunikace jde tedy přes modul bloků do modulu hran, kde se jednotlivé hrany překreslí.

Pomocí diagramu na obrázku 3.8 jsem znázornil ukázku komunikace mezi moduly na příkladu zobrazení schématu kliknutím na jeho položku v panelu projektů (počítá se s tím, že schéma není ještě otevřeno, tedy nemá svou záložku). V diagramu nejsou zahrnuty datové moduly ani konkrétní instance objektů, pouze samotné prezentační moduly, jinak by byl diagram příliš složitý. Na diagramu je znázorněno, že se nejprve nakreslí otevírané schéma (drawBlocks()), pak se smaže právě zobrazené schéma, a nakonec se otevírané schéma vyrenderuje. Rozdíl mezi nakreslením a vyrenderováním je takový, že při nakreslení vznikne instance příslušného DOM elementu a nastaví se jí neměnné hodnoty, kdežto při renderování se element vloží do DOMu dokumentu a nastaví se mu hodnoty, které se mohou při práci s aplikací měnit. Kdyby se mělo zobrazit schéma, které je již otevřené (má svoji záložku), volalo by se při vykreslování schématu pouze renderBlocks() a renderEdges() zobrazovaného schématu.

3.6 Komunikace se serverem

Webové rozhraní potřebuje mít přístup k nějakému zdroji, odkud bude brát data, aby s nimi mohl uživatel skrze uživatelské rozhraní pracovat. Jelikož zatím není back-end Surmonu hotov, potřeboval jsem vytvořit nějaký zdroj testovacích dat. V této sekci se budu zabývat právě popisem datových zdrojů,



Obrázek 3.8: Tento sekvenční diagram zachycuje komunikaci mezi prezentačními moduly při otevření schématu z panelu projektů, které ještě není otevřeno.

které jsem implementoval a použil. Kromě toho uvedu některé detaily ohledně komunikace s testovacím webovým serverem, který jsem pro účely testování implementoval.

Za účelem získávání dat jsem zpočátku nepoužíval žádný server, ale takzvaný „fake“ modul, v němž byla v proměnných uložena testovací data. Data ve fake modulu měla stejnou podobu, jakou by měla, kdyby byla získávána ze serveru, takže mohl tento modul dobře simulovat skutečný server. Fake modul jsem používal hlavně v počátečních fázích, kdy jsem se zabýval základními akcemi v rámci webového rozhraní, zejména tvorbou schématu. V uživatelském rozhraní v těchto fázích nebyly žádné projekty ani další schémata a veškerá má implementační činnost se soustředila na plátno, kde se schéma vytváří.

Když už jsem začal více pracovat s objekty, které mají být získávány ze serveru, nahradil jsem fake modul data modulem a v něm jsem definoval funkce, v nichž se volají AJAX požadavky na server. Kvůli tomu jsem si musel naimplementovat vlastní server a v něm přijímání požadavků na základě REST architektury. Mnou vytvořený server je napsán v jazyce Javascript kvůli jednoduchosti práce s JSON dokumenty a jednotnému programovacímu jazyku. Zpracovávání požadavků jsem samozřejmě implementoval zjednodušeně, hlavně pak aktivaci schématu, u které server posílá v odpovědi všem výstupním blokům stejná data k zobrazení.

Kvůli tomu, že data se kterými server pracuje odpovídají původnímu datovému modelu (z kapitoly návrhu), je třeba před odesláním dat na server data upravit. Zaprvé je vhodné odstranit pomocné vlastnosti, které jsou využívány pouze na straně front-endu. Kvůli některým takovýmto vlastnostem vznikají cyklické struktury, které se nedají převést do formátu JSON a poslat na server. Také je důležité převést do tohoto formátu všechny podobjekty, které se nachází uvnitř daného objektu. Pro tyto účely jsem u většiny objektů implementoval metodu, která vygeneruje zdrojovou podobu toho objektu, a ta může být poslána na server. Tato metoda se volá postupně na všechny podobjekty v hierarchické struktuře posílaného objektu.

Komunikace se serverem probíhá i při importu schématu z disku. V tomto případě se pošle požadavek, který je stejný jako při vytvoření nového schématu, ale přiložená data jsou trochu jiná. Pošle se standardně objekt s informacemi o nově vytvářeném schématu, ale objekt místo jednotlivých informací o schématu obsahuje zdrojovou podobu celého importovaného schématu. Objekt má ještě jednu svou položkou, kterou serveru signalizuje, že se jedná o celé schéma a nikoliv o omezenou množinu informací o schématu. Server po přijetí požadavku vygeneruje nová UUID pro schéma a všechny bloky v něm, aby nedošlo ke kolizi identifikátorů v případě, že by uživatel exportoval schéma a importoval ho do stejného projektu.

Obecně platí, že uživatel může během komunikace se serverem pracovat s webovým rozhraním (požadavky směřující na server jsou asynchronní). Nicméně při vytváření nových projektů a schémat se nejdříve pošle požadavek na server, který v odpovědi vrátí JSON instanci daného typu objektu.

Poté se vytvoří instance objektu na front-endu a až po tom všem se projekt či schéma konečně zobrazí uživateli, takže do té doby nemůže uživatel se těmito novými objekty nijak interagovat. Některé požadavky se také mohou spustit až po dokončení jiných. Jelikož jsem implementoval vytváření bloků pomocí jejich definic, musí se ze serveru nejprve získat všechny definice bloků, a pak je teprve možné stáhnout uložené projekty a schémata, neboť se při vytváření instancí schémat na front-endu vytváří instance jeho bloků podle definic.

3.7 Optimalizace

Složitější webová rozhraní mají typicky problémy s rychlostí, protože častěji komunikují se serverem a pracují s velikým množstvím elementů, které navíc mají nastaveno mnoho událostí, aby se s nimi dalo interagovat. Problematickým prvkem v aplikacích umožňujících tvorbu diagramů jsou také hrany, jejichž vykreslování je komplikovanější. Z těchto důvodů jsem se snažil implementovat webové rozhraní tak, aby fungovalo efektivně. Popíši zde, jakým způsobem jsem optimalizoval webové rozhraní z hlediska rychlosti a také uvedu některé návrhy na optimalizaci do budoucna.

Snažil jsem se zajistit, aby nemusela příliš často probíhat komunikace se serverem. Proto jsem implementoval webové rozhraní a REST API serveru tak, aby v jedné odpovědi získalo webové rozhraní co nejvíce relevantních dat najednou. Proto jsou při načítání projektů ze serveru získány nejen samotné projekty, ale i všechna jejich schémata současně. Stejně tak jsou při načítání webového rozhraní staženy všechny definice bloků, takže poté už není potřeba získávat další definice při otevírání jednotlivých kategorií. Jedním typem komunikace, kterou jsem implementoval a která by mohla být optimalizována, je uložení schématu nebo projektu na server při jejich vytvoření. Díky této komunikaci se na serveru mohou tyto objekty uložit a mohou jim být vygenerována UUID, ale jedná se o další požadavek zasílaný na server. Schémata a projekty by bylo možné ukládat až při explicitním úkonu uživatele, což by také podporovalo offline práci s webovým rozhraním.

Pro rychlost webového rozhraní je také nutné efektivně pracovat s DOMem. Z tohoto důvodu si ukládám jQuery kolekce důležitých částí DOMu, aby k nim byl v budoucnu rychlý přístup. Dále se při změně datového modelu snažím překreslovat jen tu část uživatelského rozhraní, které se změna modelu týká. Jedním aspektem práce webového rozhraní s DOMem, který by byl potřeba optimalizovat, je vykreslování již otevřených schémat (těch, co jsou v záložkách). V současné době je toto implementováno tak, že se při kliknutí na záložku znovu přidají na plátno DOM elementy všech bloků a hran daného schématu (nevytváří se, jen se znovu vykreslí na plátno). Tento proces by bylo vhodné optimalizovat tak, že by každé otevřené schéma mělo své vlastní plátno, tato plátna by se navzájem „překrývala“ a bylo by vidět pouze plátno právě zobrazeného schématu. Ostatní plátna by byla skryta pomocí CSS stylů.

Tento přístup by mohl být mnohem efektivnější a je možné ho implementovat více způsoby – nastavováním vlastnosti „display“ nebo „visibility“ u pláten pro jejich skrývání a zviditelňování. Obecně je efektivnější používat vlastnost visibility, protože její změna nemění rozměry elementu, takže se nespustí aktualizace rozmístění elementů na stránce, což je náročná operace [53].

Velmi složitou operací webového rozhraní je vykreslování zdrojové podoby schématu. Když jsem toto implementoval pomocí jQuery, bylo vykreslování pomalé, protože jsem v kódu vytvářel elementy pomocí zápisu **$(' ')**, což je samo o sobě rychlé, ale při vykreslování stovek elementů je vidět příliš veliké zpomalení. Proto jsem nahradil veškeré použití jQuery ve vykreslování zdrojové podoby schématu čistým javascriptovým kódem a výsledná rychlost byla několikanásobně větší. Každopádně u velmi rozsáhlých schémat může stále být vykreslování delší, a proto se provádí pouze při zobrazení schématu, před zobrazením zdrojové podoby schématu nebo před exportem schématu.

3.8 Kompatiblita

Zde se budu zabývat popisem průběhu a výsledků testování webového rozhraní na různých prohlížečích. Testoval jsem ty přohlížeče, které jsem si zvolil v kapitole návrhu s výjimkou Google chrome, jelikož on i Opera, kterou jsem testoval, mají stejné jádro, takže závěry z testování na Opeře lze vztáhnout i na Google chrome. Zajistit kompatibilitu nebylo díky knihovně jQuery tak náročné, nicméně testování mi i tak poskytlo hodně poznatků ohledně schopností verzí jednotlivých prohlížečů. Při hledání příčin nedostatků kompatibility jsem využíval web „Can I Use“ [16].

Opera 38 byl první prohlížeč, který jsem testoval. Některé nedostatky v kompatibilitě nebyly tak vážné, jako například vlastnost Opery a Google chromu nespouštět událost „hover“, když je stisknuté tlačítko myši. Kvůli tomu nedochází ke zvětšování vstupních pinů, když na ně uživatel najede myší při tažení hrany. Na další problém jsem narazil při zkoumání funkčnosti editovatelných vlastností bloků. Aby se spouštěly události při úpravě těchto vlastností, musel jsem k ovládacím prvkům vlastností zaregistrovat novou událost typu „onChange“, jelikož původní „onInput“ událost se u některých typů vlastností nespouštěla. Také jsem musel zavést resetování ovládacího prvku pro nahrání obrázku, protože v něm zůstával první nahraný obrázek.

Dále jsem testoval **IE**, u kterého jsem očekával největší nutnost úprav. Původně jsem chtěl, aby bylo webové rozhraní funkční i na IE 9, takže jsem zpočátku ladil kompatibilitu IE v této verzi. Zde nebylo příliš mnoho funkčních problémů, které by se daly opravit, spíše několik funkcionalit vůbec nebylo v IE 9 dostupných. Mezi ně patřilo nahrávání souborů, CSS přechody, validace formulářových polí, a další. Zjistil jsem, že webové rozhraní se v IE 9 nenačte, pokud obsahuje v kódu použití vlastnosti „console“ objektu „window“, protože tato vlastnost je dostupná pouze, když jsou v prohlížeči otevřené vývojářské

nástroje.

V IE 10 pak už mnoho chyb nebylo. Tato verze IE má oproti ostatním testovaným prohlížečům určité odlišnosti ve zpracovávání událostí týkající se úpravy vstupních polí. Každopádně závažnější chybou, kterou jsem opravil byla nefunkčnost exportu schématu, jelikož v IE 10 není zaveden HTML atribut „download“, pomocí kterého jsem export implementoval. Opravu jsem provedl pomocí funkce „msSaveOrOpenBlob“, která je specifická pro IE, a s jejíž pomocí lze zprovoznit export v IE. Díky této funkci jsem rozšířil metodu implementující export tak, aby použila funkci „msSaveOrOpenBlob“, pokud je v prohlížeči dostupná, čímž jsem umožnil exportování schémat v IE 10. Další větší chyba se týkala označování hran. Hrany se nedaly označit, protože se při kliku objevilo laso pro označování bloků, které přestože mělo nastavenou nulovou velikost, tak překáželo v kliknutí na hranu. Zobrazením lasa až při tažení myši jsem tento problém vyřešil.

Zkoušel jsem funkcionalitu webového rozhraní i na **Edge 14** (nižší verze není dostupná). Nenašel jsem zde žádné další problémy, kromě jedné maličkosti, která se opět týkala exportování schémat. Schémata se exportovala pod očividně strojově vygenerovanými názvy, takže bylo nutné schémata ručně přejmenovávat. Tuto chybu jsem však odstranil stejným způsobem, kterým jsem v IE 10 zprovoznil export (tedy použitím funkce „msSaveOrOpenBlob“).

Nakonec jsem testoval **Firefox 40**. Jelikož jsem během vývoje používal Firefox pro ladění webového rozhraní, je kód front-endu tomuto prohlížeči nejvíce přizpůsobem. Nalezl jsem pouze jedinou chybu, na kterou jsem narazil i v IE 10, a tou je konflikt mnou implementovaného scrollování plátna pomocí prostředního tlačítka myši s totožnou funkcionalitou nativně implementovanou v těchto prohlížečích. Pro spravení této chyby stačilo pouze v kódu zastavit nativní funkcionalitu.

Během většiny testování jsem nemusel použít postup graceful degradation, jelikož většina funkcionalit je plně dostupná ve všech cílových verzích prohlížečů. Pouze při testování IE 9 jsem přidal do kódu podmínky, kvůli kterým se určité ovládací prvky uživatelského rozhraní nezobrazí, pokud daný prohlížeč nesplňuje konkrétní podmínky (například nepodporuje nahrávání souborů). Úpravy kódu, které jsem během testování dělal, se hlavně týkaly doplňování částí kódu tak, aby byla funkcionalita dostupná na všech cílových verzích prohlížečů.

3.9 Finalizace uživatelského rozhraní

Tato sekce má za úkol popsat, jak jsem vytvářel výslednou podobu uživatelského rozhraní. Konkrétně zde budu psát o barvách, které jsem využil a také o vzhledu některých komponent uživatelského rozhraní. Část této sekce jsem vymezil pro popis ikon, které jsem zvolil pro reprezentaci různých akcí.

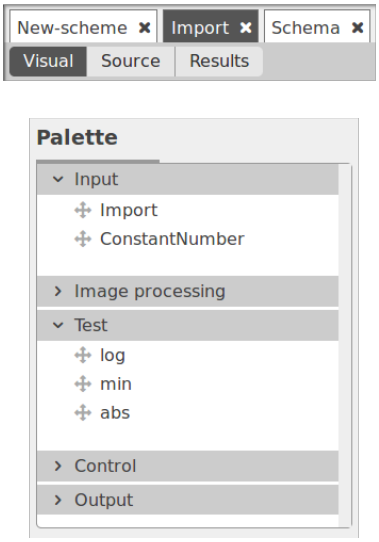
V poslední části se budu zabývat uživatelským testováním, jehož výsledky zde zmíním a na jehož základě upravím uživatelské rozhraní do finální podoby.

V uživatelském rozhraní vystupuje spoustu prvků s různými významy. Proto jsem pomocí SASS proměnných definoval barvy, z nichž každá se týká nějakého typu komponent nebo stavu komponenty. Snažil jsem se používat barvy konzistentně napříč celým uživatelským rozhraním, aby bylo uživateli vždy rychle zřejmé, co která barva naznačuje. Definoval jsem například barvy pro normální text, text na tmavém pozadí, zvýrazněný text, barvu pro pozadí aktivní komponenty, barvu pro oddělovače a mnoho dalších. Tyto zmíněné barvy mohou být použity kdekoliv v uživatelském rozhraní, ale definoval jsem ještě barvy specifické pro bloky a hrany, jelikož ty také mají své vlastní komponenty a stavy.

Barev jsem tedy nevyužil pouze pro grafické účely, ale hlavně pro účely podání zpětné vazby uživateli. Barvy jsem použil například pro signalizaci aktivního schématu, projektu či bloku nebo pro zvýzaznění ovládacích prvků, nad kterými se nachází kurzor myši. Při tažení nové hrany se piny, které mohou hranu přijmout zabarví, aby bylo uživateli zjevné, že jsou piny kompatibilní a mohou být propojeny. Při tažení hrany nad kompatibilním pinem se pin zvětší, což má signalizovat, že uživatel může pustit hranu, přičemž se hrana na tomto pinu „uchytí“. Užil jsem i různých tvarů kurzoru myši po názornější signalizaci akce, která právě probíhá nebo kterou může uživatel provést s elementem, nad kterým se kurzor myši nachází.

V uživatelském rozhraní jsem použil některé standardní podoby komponent, na které jsou uživatelé zvyklí a rozumí jim. Jednou z těchto komponent jsou záložky, které jsou velmi rozšířenou komponentou pro zobrazování různých obsahů bez nutnosti načítání celé webové stránky. Dále jsem definoval podobu přepínače pohledu na schéma a to tak, aby to vypadalo, jako že jsou všechny jeho tlačítka součástí jednoho celku, a že tedy jen jedno z nich může být aktivní, zatímco ostatní jsou neaktivní. Na obrázku 3.9 je ukázka zmíněných komponent.

Další z větších komponent jsou akordióny v paletě a panelu projektů, které jsem se rozhodl implementovat podle doporučení vyplývajících z jednoho výzkumu použitelnosti akordiónů [54]. Ten říká, že ikona u každé položky akordiónu by měla být umístěná vlevo hned před názvem položky, jelikož se pak uživatelům daří rychleji akordión používat. Co se týče podoby ikony, tak nejlépe na tom je ikona „+“ pro rozbalení položky a „x“ pro její sbalení. Nicméně vzhledem k tomu, že chci k položkám projektů přidat akci pro zavření projektu, která bude mít právě ikonu „x“, zvolil jsem pro ikony položek akordiónu šipky, jak je vidět na obrázku 3.9. Akordióny mohou fungovat tak, že vždy může být rozbalená jen jedna položka nebo více položek. Druhou možnost považuji za výhodnější, jelikož například u palety bude uživatel chtít přidávat do schématu bloky z několika konkrétních kategorií, které by takto mohly být zobrazeny najednou.



Obrázek 3.9: Na tomto obrázku jsou podoby komponent akordiónu, záložek a přepínače, jak jsem je implementoval ve webovém rozhraní.

3.9.1 Ikonografie

Dlouho jsem během realizace webového rozhraní pracoval se zástupnými ikonami, které měly alespoň trochu připomínat jimi reprezentované akce. Všechny ikony jsem získal ze stránky Fontello [44], ale ne všechny jsem vyhodnotil jako vhodné pro daný účel. Jelikož webové rozhraní projde uživatelským testováním, mělo by napřed obsahovat co nejvíce přesné ikony. Proto jsem nejprve zkoumal, jaké ikony bych mohl z Fontella vybrat pro dané akce, a jak bych je mohl případně upravit, budou-li ne zcela vyhovující. Nakonec jsem vybral tyto ikony:

- ikona odkazu na manuál
- vytvoření nového projektu
- vytvoření nového schématu
- zobrazení seznamu uložených projektů uživatele, ze kterého se dají otevřít nebo smazat; v seznamu jsou pouze projekty, které nejsou otevřené
- uložení projektu nebo schématu na server
- přejmenování projektu nebo schématu
- smazání projektu, schématu nebo vnitřku schématu (všechny bloky a hrany)

- zavření projektu
- import schématu
- export schématu
- aktivace schématu
- tato ikona naznačuje, že se dá element přetáhnout myší na plátno
- ikona sbalené položky akordiónu
- ikona rozbalené položky akordiónu

Tyto ikony mají konzistentní význam napříč celým uživatelským rozhraním, takže například ikona pro vytvoření schématu je naprosto stejná jak v hlavním menu, tak u položky konkrétního projektu v panelu projektů. Některé ikony však mohou částečně měnit význam podle kontextu. Například ikona symbolizující ukládání se může vztahovat k projektům i schématům – záleží na tom, kde se ikona nachází. Podle toho se při kliknutí na ikonu uloží schéma nebo projekt, nicméně tato ikona vždy symbolizuje funkci ukládání.

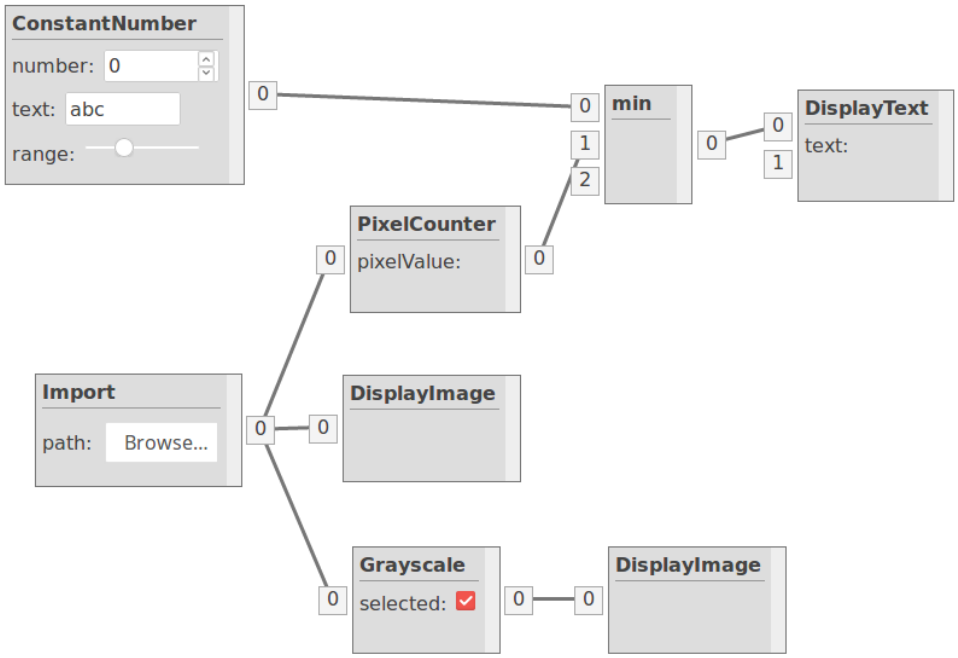
Pouze dvě ikony jsem si musel vytvořit sám, jelikož jsem na stránce Fontella nenašel žádné již hotové vhodné ikony. Těmito dvěmi ikonami jsou vytvoření nového projektu a import. Ikony jsem vytvořil modifikací existujících ikon, k čemuž jsem využil program Inkscape. Existující ikony jsem získal jejich extrakcí z SVG písma staženého ze stránky Fontella. K extrakci jsem použil program FontForge [55].

3.9.2 Uživtelské testování

Po dokončení implementace webového rozhraní jsem se rozhodl provést uživatelské testování, jelikož je důležitou součástí tvorby uživatelského rozhraní, a také proto, že se v uživatelském rozhraní nacházely některé prvky, o jejichž použitelnosti jsem měl určité pochybnosti, které jsem si chtěl ověřit. V této části práce popíši, jak jsem testování prováděl a uvedu některá podezření na nedostatky uživatelského rozhraní, která jsem před testováním měl. Na závěr uvedu souhrn výsledků uživatelského testování. Více informací o samotném testování je v UI specifikaci (viz přiložené CD).

Nejprve jsem se zaměřil na vymyšlení scénářů pro testování. Chtěl jsem, aby tyto scénáře pokryly všechna možná použití webového rozhraní. Vymyslel jsem tyto scénáře, které budou uživatelé při testování provádět v určitém pořadí:

1. Uživatel má za úkol vytvořit konkrétní schéma, které je znázorněno na obrázku 3.10. Potom má uživatel nahrát do schématu obrázek, zadat vstupní číslo a schéma aktivovat.



Obrázek 3.10: Na tomto obrázku je schéma, které měli uživatelé za úkol vytvořit a aktivovat v rámci uživatelského testování.

2. Dalším úkolem je importovat schéma z disku, změnit ho smazáním bloku a hrany, upravit velikosti bloků a nakonec toto upravené schéma exportovat.

3. Vytvoření více projektů a schémat v nich, přičemž v tomto scénáři jsem se hlavně zaměřoval na orientaci uživatele ve vytvořených schématech a na to, jak porozumí tomu, který projekt a schéma jsou právě aktivní.

Během testování jsem kladl uživatelům otázky, abych zjistil, proč zrovna provádí určitou akci. Také jsem během i po testování zadával doplňující menší úlohy – například jsem zkoušel, jak uživatelé rozumí ikonám či určitým částem uživatelského rozhraní, zejména panelu pro zobrazení vlastností označeného bloku nebo úchytům pro zvětšování bloků.

Kromě scénářů jsem si vytvořil poznámky o jistých aspektech uživatelského rozhraní, u kterých jsem si chtěl ověřit, zdali jsou v uživatelském rozhraní úspěšně realizovány či nikoliv. Tyto aspekty jsou uvedeny v následujícím seznamu.

- srozumitelnost použitých ikon
- označování projektů v panelu projektů (projekty je možné označovat pouze zobrazením některého jeho schématu)

- práce s akordiónem
- efektivita práce s bloky – přesouvání, zvětšování, úprava vlastností
- srozumitelnost duplikace pinů
- objevitelnost záložek schémat
- objevitelnost akce importu
- označování bloků ve schématu (označení proběhne až po uvolnění stisku tlačítka myši při kliknutí na blok místo toho, aby proběhlo už při stisku tlačítka)

Následně jsem potřeboval alespoň pět uživatelů, abych mohl testování provést a získal dostatečně relevantní výsledky [56]. Nechtěl jsem vybírat uživatele, kteří už znají desktopovou verzi aplikace Surmon, protože mnou vytvářená webová verze je desktopové verzi částečně podobná, takže by uživatelé lépe rozuměli uživatelskému rozhraní a nezískal bych tak cenné výsledky. Vybral jsem si tedy pět uživatelů, kteří mají zkušenosti s prací s desktopovými i webovými aplikacemi, ale nikdy nepracovali s aplikací Surmon. Tyto uživatele jsem vybral tak, aby měli různé úrovně technického nadání a různé úrovně zkušeností s prací s aplikacemi – od uživatele, který nemá příliš kladný vztah k učení se nových aplikací až po uživatele, který neustále pracuje s novými aplikacemi.

Během testování jsem pořizoval zvukové záznamy a záznamy z obrazovky. Díky nim jsem se mohl během testování lépe soustředit na to, co uživatel dělá, aniž bych si musel dělat mnoho poznámek a až po skončení testování jsem prošel vytvořené záznamy, ze kterých jsem získla potřebné informace. Průběhy testování, zejména momenty, kdy měli uživatelé problémy provést nějaký úkon, jsou uvedeny v UI specifikaci na přiloženém CD.

3.9.3 Výsledky uživtelského testování

Během testování bylo zjištěno mnoho nedostatků, z nichž některé však byly objeveny pouze jedním uživatelem. Na takovéto nedostatky jsem však při shrnutí výsledků testování nebral zřetel a zaměřil jsem se na ty nedostatky, které byly indikovány více uživateli nebo byly závažnější. Po zpracování výsledků ze všech testování bylo vidět, že se zjištěné nedostatky začínají dosti opakovat, a u poslední testované osoby bylo objeveno velmi málo nových problémů. V následující tabulce jsou nedostatky shrnuty a jsou k nim uvedena jejich možná řešení. V tabulce nejsou zmíněna ověření kladných aspektů uživatelského rozhraní (co není zmíněno v tabulce, je podle testování v pořádku) a jsou v ní pouze vážnější nedostatky. Více výsledků je opět v UI specifikaci.

Na základě všech zjištěným nedostatků je po testování potřeba webové rozhraní upravit. Některé problémy lze vyřešit velmi snadno, ale u jiných by

Tabulka 3.1: Shrnutí vážnějších nedostatků vyplývajících z uživatelského testování a návrhy na jejich opravu.

Nedostatek	Návrhy na opravu
Nejsou zcela jasné funkce ikon, nejvíce v hlavním menu.	Přidat tooltipy ke všem ikonám reprezentující akce.
Uživatelé často zpočátku na bloky v paletě klikají.	Zobrazit blok na volném místě na ploše po kliknutí na blok v paletě. Přetahování bloků by nadále fungovalo.
Projekt po vytvoření není označen, takže není možné do něj vložit nově vytvořené schéma z hlavního menu.	Automaticky označit nově vytvořený projekt nebo zrušit možnost vytváření nového schématu z hlavního menu.
Čísla v pinech jsou matoucí a vypadají jako data, se kterými schéma pracuje.	Smazat čísla z pinů a místo nich třeba zobrazit nějakou ikonu, viz předchozí bod.
Většina uživatelů vytvářela nové schéma kvůli importu schématu.	Změnit importování tak, aby skutečně bylo potřeba vytvořit nové schéma nebo spíše k možnosti importu z hlavního menu přidat možnost importu z dialogu pro vytvoření nového schématu (byla by tam ikona importu i slovní popis).
Import je ve webovém rozhraní těžko objevitelný.	Nahradit input pro nahrání souboru ikonou pro import.
Manuál pravděpodobně není příliš přehledný.	Přidat do navigace v manuálu odkazy na konkrétní úkony místo na sekce, ve kterých se různé úkony nachází. Velmi užitečné by mohlo být vytvoření videa demonstrující práci s aplikací. Také by se hodilo vyhledávání v manuálu.
Uživatelé by chtěli mít možnost označit projekt kliknutím na jeho název.	Přidat tuto funkcionalitu, přičemž by pak rozbalování a sbalování schémat projektu bylo prováděno dvojklikem na název projektu. Bylo by také potřeba zrušit podobu akordiónu u panelu projektů, protože by panel fungoval jinak.
Neexistuje historie a funkce undo a redo.	Tuto funkcionalitu přidělat.
Hrany se dají vytvářet pouze taháním za výstupní piny bloků.	Umožnit vytváření hran i ze vstupních bloků.

mohla implementace trvat velmi dlouho – například přidání historie editací schémat nebo rozšíření logiky, podle které jsou duplikovány piny v různých blocích. Problémy mohou být i provázané, takže vyřešením jednoho už nemusí být nutné řešit nějaký jiný. Na základě poznatků z testování jsem upravil webové rozhraní implementací řešení převážně jednodušších problémů, některé z nichž však mohou řešit i závažnější problémy. Opravy, které jsem provedl zahrnují použití lepších ikon, smazání čísel z pinů, označení nově vytvořeného projektu nebo posunutí úchytů pro škálování bloku více směrem dovnitř bloku.

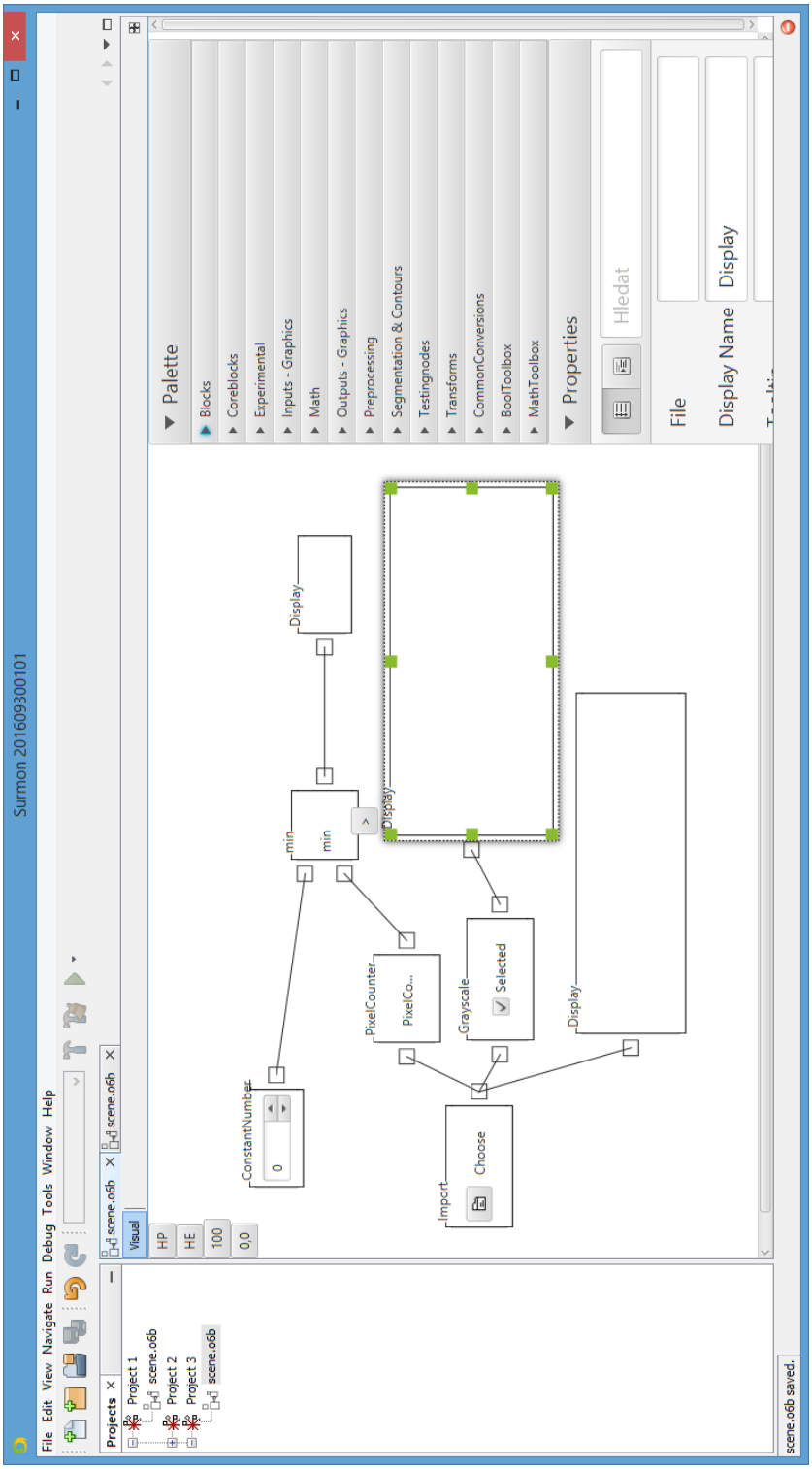
3.10 Porovnání vytvořeného webového rozhraní s desktopovou verzí

Provedu zde porovnání mnou implementovaného webového rozhraní pro aplikaci Surmon s existující desktopovou aplikací. Uvedu zde, do jaké míry jsou si obě verze Surmonu podobné a jak jsou spolu kompatibilní. Zaměřím se také na prvky, ve kterých se obě podoby Surmonu liší a zmíním funkcionalitu, kterou má každá z těchto verzí navíc oproti té druhé. Pro ukázku jsou zde obrázky obou verzí aplikace se stejným zobrazeným schématem (3.11 a 3.12), aby bylo možné si prohlédnout některé rozdíly v uživatelském rozhraní a grafice.

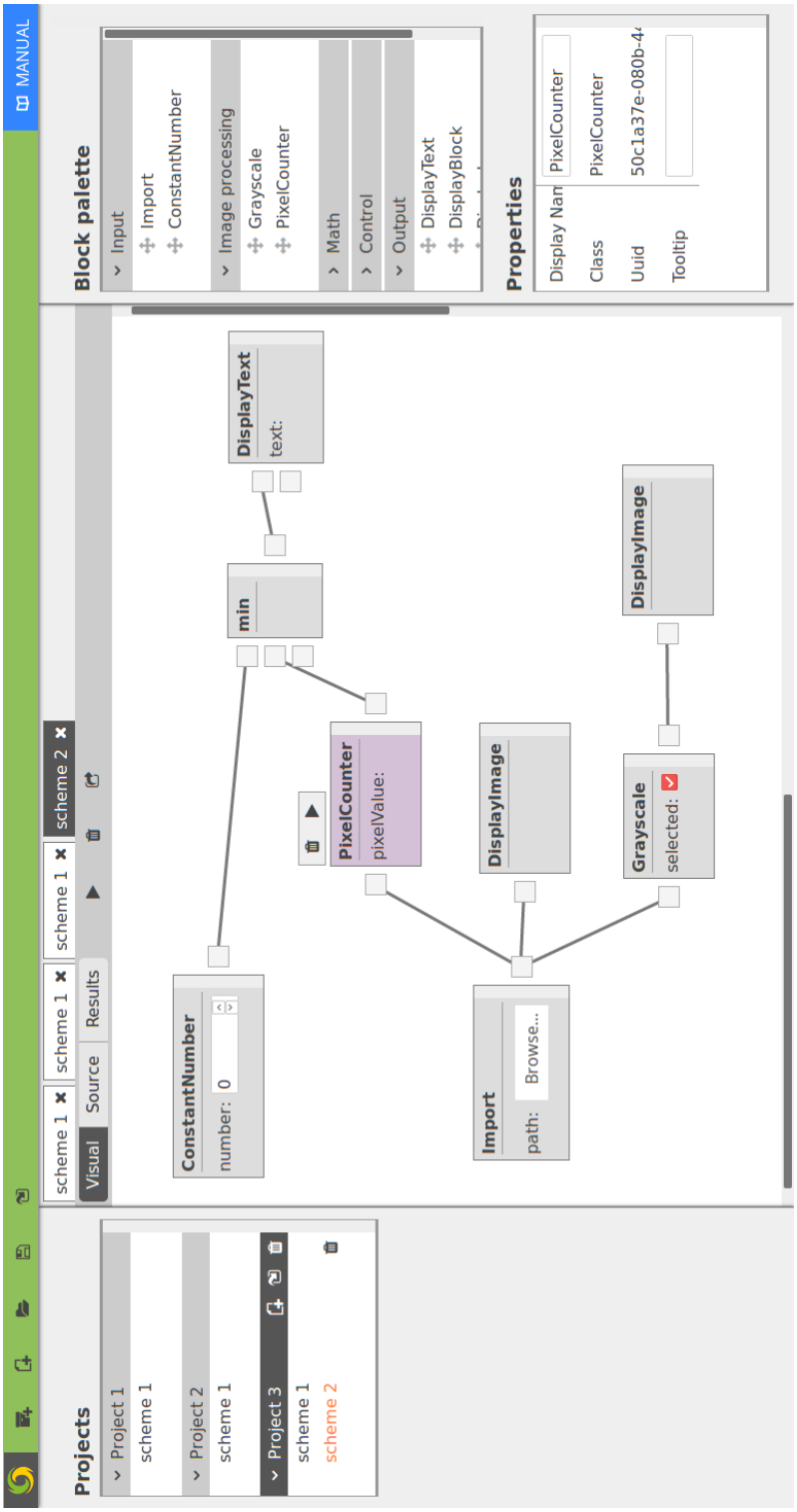
Webové rozhraní jsem vytvářel na základě znalosti desktopové verze. Snažil jsem se, aby webové rozhraní poskytovalo stejnou funkcionalitu a aby bylo s desktopovou verzí kompatibilní. Z tohoto důvodu si jsou obě verze dosti podobné, alespoň z hlediska funkcionality. Tato podobnost je záměrná, jelikož se v podstatě jedná o tu samou aplikaci, takže by pro uživatele nebylo dobré, kdyby bylo webové rozhraní příliš odlišné.

Obě verze Surmonu pracují se stejnou podobou zdrojové reprezentace schémat, která je ve formátu JSON. Webové rozhraní je díky tomu schopné importovat schémata vytvořená desktopovou aplikací, ale tato schémata se musí skládat z bloků, které webové rozhraní podporuje. Dalším společným znakem je z veliké části stejné rozmístění komponent uživatelského rozhraní, jak je na obrázcích vidět. Vlevo je umístěn panel projektů, vpravo paleta bloků a vlastnosti zobrazeného bloku a uprostřed plátno pro tvorbu schémat, nad kterým jsou záložky s otevřenými schématy. Vytváření schématu a interakce se schématem jsou také převážně stejné.

V čem se však tyto verze liší, je hlavně podoba uživatelského rozhraní nebo trochu jiné postupy při provádění některých úkonů. Desktopová verze je vytvořena na platformě Netbeans, což velmi ovlivňuje podobu uživatelského rozhraní. Grafická podoba je spíše strohá a nepoužívá mnoho barev. Oproti tomu grafická podoba webového rozhraní je pestřejší – používá více barev pro zvýrazňování obsahu a také využívá barvy a animace jako zpětnou vazbu pro uživatele. Webové rozhraní se také liší v tom, že nenabízí uživateli žádné menu v záhlaví aplikace, jehož položky by obsahovaly seznamy akcí, ale všechny akce



Obrázek 3.11: Obrázek je screenshotem celého okna desktopové verze aplikace Surmon.



Obrázek 3.12: Obrázek je screenshotem celého okna webového rozhraní aplikace Surmon.

jsou prezentovány uživateli přímo v různých menu (všechny akce jsou vždy reprezentovány pouze ikonami).

Odlišností v rámci uživatelského rozhraní je také realizace panelu projektů. Webové rozhraní panel implementuje jako akordión, ve kterém se obsah položek zobrazujea skrývá jedním klikem, kdežto u desktopové verze fungují položky panelu projektů jako složky, které se otevírají dvojklikem. Na principu akordiónu jsou v obou verzích založeny palety, ale ve webovém rozhraní je navíc možné mít rozbaleno více kategorií bloků najednou. Webové rozhraní oproti desktopové verzi zobrazuje akce týkající se projektů a schémat přímo v odpovídajících položkách v panelu projektů. Dalším prvkem, který v desktopové verzi není, je možnost provádět veškeré interakce s blokem (posun, škálování, úprava vlastností) bez nutnosti nejprve na blok kliknout.

Mnoho funkcionalit, které jsou v desktopové verzi přítomny, ve webovém rozhraní ještě chybí. Některé z nich jsou velmi důležité a jiné by zase mohly sloužit jako vhodný doplněk pro jednodušší používání aplikace. Tyto funkcionality jsou vypsány v následujícím seznamu.

- zobrazení pomocného menu při kliknutí pravým tlačítkem myši (u webových aplikací toto není zcela běžné)
- uchovávání historie práce s aplikací a poskytnutí funkcí undo a redo
- označování více bloků najednou
- aktivace samotného bloku
- používání superbloků
- zobrazování výsledků aktivace v samostatných záložkách
- vyhledávání ve vlastnostech označeného bloku

3.11 Rozšíření do budoucna

Na závěr kapitoly bych chtěl ještě uvést určité možnosti pro rozšíření nebo optimalizaci webového rozhraní Surmonu. Je zřejmé, že je mnoho funkcionalit, které webové rozhraní zatím nepodporuje, ale které by byly důležité nebo alespoň nějak užitečné pro uživatele aplikace Surmon. Nejprve zde uvedu některé funkcionality, které bude nezbytné nebo velmi potřebné v budoucnu do front-endu přidat a poté se zaměřím na méně důležité funkcionality a potenciální optimalizace.

Jak už jsem párkrát zmiňoval, back-end Surmonu právě vzniká a není možné ho zatím používat. Z tohoto důvodu jsem pracoval s testovacími daty, hlavně pak s definicemi bloků – vybral jsem si pár definic bloků a zbylé jsem v rámci této práce neřešil. Proto webové rozhraní není připravené pracovat se všemi možnými typy bloků a typy jejich vstupních či výstupních vlastností.

Stávající implementace, která je připravena pro práci s vybrannými definicemi bloků, by měla pokrýt velikou část zbývajících definic, protože jsou často podobné a implementace by měla být pro tyto účely dostatečně abstraktní. V žádném případě však implementace zatím nepokrývá úplně všechny definice, jejich části a typy vstupních či výstupních dat.

Jedním z vážnějších nedostatků webového rozhraní v současném stavu je absence uživatelských účtů. Ty by v budoucnu neměly chybět, protože bez nich není možné spolehlivě určit, které projekty a schémata na serveru patří kterému uživateli. Uživatel může nyní využívat existující funkce pro import a export schémat jako náhradu za ukládání dat na server, což samozřejmě zdaleka není tak efektivní a příjemné. Ve větší části této práce jsem s uživatelskými účty nepočítal, ale zahrnul jsem do návrhu uživatelského rozhraní wireframy, které s účty souvisí.

Výhodou zavedení uživatelských účtů by byla i možnost ukládat na server určitá nastavení nebo metadata o používání webového rozhraní. Například by se na server mohla posílat data o tom, jaké projekty měl uživatel naposledy otevřené, jakou záložku měl naposledy otevřenou nebo které kategorie bloků v paletě měl rozbalené. Toto vše určitě není nezbytné, ale uživatelům by to mohlo ušetřit čas. V návrhu REST API jsem uváděl zdroj, který poskytuje naposledy otevřené projekty, ale v současném stavu bez uživatelských účtů vrací API při požadavku na tento zdroj všechny projekty uložené na serveru.

Další zatím chybějící součástí webového rozhraní jsou superbloky. Ty jsem sice částečně implementoval, ale ještě je potřeba přidat možnost vkládání schémat jako superbloků na plátno a také vytvořit různé mechanismy, které se superbloků týkají (například definování rozhraní superbloku pomocí bloků vstupního a výstupního rozhraní). Superbloky nejsou nezbytným prvkem aplikace, ale mohou práci s ním uživatelům velice usnadnit.

Je ještě spoustu dalších, převážně menších potenciálních vylepšení, z nichž některá jsou vyjmenovaná a zjednodušeně popsaná v následujícím seznamu. Nejsou v něm uvedena vylepšení, která jsou vypsána v UI specifikaci jako návrhy pro řešení problémů vyplývajících z uživatelského testování a ani v něm nejsou zmíněny nedostatky oproti desktopové verzi (ty jsou uvedeny v předchozí sekci).

- implementace definic rozmístění vlastností uvnitř bloku
- zavedení potvrzovacích dialogů při provádění nenávratných akcí (například smazání schématu)
- automatické zarovnávání bloků podle ostatních bloků nebo jejich přichytávání k mřížce
- realizace klávesových zkratek (některé už implementovány jsou)
- přidání třetího módu zobrazení schématu, kde by byly v záložkách zobrazeny výsledky aktivace schématu

- podpora práce s aplikací offline – například přidáním možnosti ukládat projekty a schémata do lokálního úložiště prohlížeče a zavedením mechanismu pro automatické nahrání projektů a schémat, které byly vytvořeny v offline stavu, na server
- efektivnější implementace zobrazování již otevřených schémat (viz sekce optimalizace)
- při tažení bloku nepřesouvat samotný blok i s jeho hranami, ale přesouvat pouze zástupný element a po jeho puštění přesunout blok na jeho místo (až poté se přepočítají hrany)

Během celé realizace jsem pracoval s „vývojářskou“ verzí implementace webového rozhraní, což samozřejmě není zcela vhodná verze pro použití uživateli. Předtím, než půjde webové rozhraní Surmonu do produkce, bude potřeba převést vývojářskou verzi do produkční verze, která bude mít menší nároky na komunikaci se serverem díky spojeným a komprimovaným zdrojovým souborům. Ve výsledné produkční verzi tedy bude načítání webového rozhraní o něco rychlejší.

Závěr

Cílem práce bylo vytvořit webové rozhraní pro aplikaci Surmon, které by bylo schopné komunikovat se vznikajícím back-endem této aplikace a které by poskytovalo uživatelům podobnou funkcionalitu jako již existující desktopová verze aplikace. Součástí práce bylo také navrhnout a na závěr otestovat uživatelské rozhraní, vytvořit manuál pro uživatele a dokumentaci pro vývojáře.

Nejprve jsem prováděl analýzu a návrh různých aspektů webového rozhraní. Konkrétně jsem analyzoval uživatelské rozhraní čtyř podobných aplikací, jejichž některými prvky jsem se inspiroval při návrhu uživatelského rozhraní. Poté jsem na základě analýzy podílů desktopových verzí prohlížečů na trhu zvolil konkrétní verze prohlížečů, se kterými by mělo být výsledné webové rozhraní kompatibilní. Zabýval jsem se též architekturou webového rozhraní. Řešil jsem, jakým způsobem bude webové rozhraní rozčleněno do komunikujících komponent a také jsem navrhl rozhraní pro komunikaci se serverem. Před samotnou realizací jsem prostudoval dostupné technologie, knihovny a aplikace, z nichž jsem si vybral vhodné nástroje pro tvorbu webového rozhraní.

Realizace probíhala tak, že jsem nejdříve do adresářové strukturu projektu zavedl potřebné knihovny a nastavení. Před implementací modulů, do kterých je webové rozhraní rozčleněno, jsem vytvořil prototyp uživatelského rozhraní, ze kterého později vznikla výsledná podoba uživatelského rozhraní. Následně jsem vytvářel jednotlivé moduly a tím jsem přidával funkcionalitu webovému rozhraní. Důležitou součástí implementace byla práce se schématem, proto jsem tuto část v kapitole realizace detailněji popsal. Dále jsem uvedl, jak moduly komunikují mezi sebou a se serverem. K závěru jsem se zaměřil na testování kompatibility napříč zvolenými verzemi prohlížečů a na uživatelské testování. Nakonec jsem porovnal v rámci diplomové práce vytvořené webové rozhraní s existující desktopovou aplikací a uvedl jsem potenciální rozšíření do budoucna.

Výsledné webové rozhraní je funkční, ale zatím není možné ho využívat z důvodu, že není hotov back-end aplikace, se kterým by mohlo webové rozhraní reálně komunikovat. Webové rozhraní je možné použít k vytváření sché-

mat, ale v současném stavu mu chybí některé prvky, aby mohlo svojí funkcionalitou plně zastupovat existující desktopovou verzi aplikace Surmon. Nicméně je otestované z hlediska komunikace se serverem, kompatibility s podporovanými prohlížeči a také z hlediska použitelnosti pro uživatele. Webové rozhraní je tedy z veliké části již připraveno na reálné používání.

Literatura

[1] Surmon, s.r.o.: Surmon. [software], [cit. 2017-02-25]. Dostupné z: http://31.31.75.152/tiki/tiki-index.php

[2] RapidMiner, Inc.: Rapid Miner Studio. [software], [cit. 2017-03-05]. Dostupné z: https://rapidminer.com/

[3] IBM: IBM SPSS Modeler. [software], [cit. 2017-04-25]. Dostupné z: http://www-03.ibm.com/software/products/cs/spss-modeler/

[4] StatSoft: Statistica. [software], [cit. 2017-04-25]. Dostupné z: http://www.statsoft.cz/

[5] NetBeans: Netbeans Platform. [software], [cit. 2017-04-25]. Dostupné z: https://netbeans.org/features/platform/index.html

[6] //SEIBERT/MEDIA: Draw.io. [online], [cit. 2017-03-06]. Dostupné z: https://www.draw.io

[7] Atlassian: Confluence. [software], [cit. 2017-05-06]. Dostupné z: https://www.atlassian.com/software/confluence/

[8] Dropbox: Dropbox. [software], [cit. 2017-05-06]. Dostupné z: https://www.dropbox.com/

[9] Lucid Software Inc.: Flowchart Maker & Online Diagram Software. [online], 2017, [cit. 2017-03-06]. Dostupné z: https://www.lucidchart.com

[10] Loranger, H.: Accordions for Complex Website Content on Desktops. [online], [cit. 2017-04-25]. Dostupné z: https://www.nngroup.com/articles/accordions-complex-content/

[11] Nielsen, J.: 10 Heuristics for User Interface Design. [online], [cit. 2017-04-25]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[12] NetBeans: Netbeans IDE. [software], [cit. 2017-04-25]. Dostupné z: https://netbeans.org/

[13] Eclipse Foundation: Eclipse IDE. [software], [cit. 2017-04-25]. Dostupné z: https://eclipse.org/ide/

[14] WWW Consorcium: Graceful degradation versus progressive enhancement. [online], 2015, [cit. 2017-03-07]. Dostupné z: https://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement#Graceful_degradation_and_progressive_enhancement_in_a_nutshell

[15] Net Applications: Browser market share. [online], 2017, [cit. 2017-03-07]. Dostupné z: https://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustomd=0

[16] Deveria, A.: Can I Use. [online], [cit. 2017-03-07]. Dostupné z: http://caniuse.com

[17] Google: Google Analytics. [software], [cit. 2017-05-06]. Dostupné z: https://www.google.cz/intl/en/analytics/

[18] Jithin: What is MVC? Advantages and Disadvantages of MVC. [online], [cit. 2017-05-06]. Dostupné z: https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/

[19] Mikowski, M. S.; Powell, J. C.: *Single page Web Applications*. Manning Publications Co., 2014, ISBN 9781617290756.

[20] HTTP Status Codes. [online], [cit. 2017-04-25]. Dostupné z: http://www.restapitutorial.com/httpstatuscodes.html

[21] Google: Angular. [software], [cit. 2017-04-25]. Dostupné z: https://angularjs.org/

[22] Facebook Inc.: React. [software], [cit. 2017-04-25]. Dostupné z: https://facebook.github.io/react/

[23] Sanderson, S.: Knockout. [software], [cit. 2017-04-25]. Dostupné z: http://knockoutjs.com/

[24] Mikowski, M. S.: Do you really want an SPA framework. [online], 2016, [cit. 2017-03-07]. Dostupné z: https://mmikowski.github.io/no-frameworks

[25] Krajka, B.: The difference between Virtual DOM and DOM. [online], [cit. 2017-04-25]. Dostupné z: http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

[26] Angelov, M.: 12 Awesome CSS3 Features. [online], [cit. 2017-05-07]. Dostupné z: http://tutorialzine.com/2013/10/12-awesome-css3-features-you-can-finally-use/

[27] Weizenbaum, N.: SASS. [software], [cit. 2017-02-25]. Dostupné z: http://sass-lang.com/

[28] Sellier, A.; aj.: LESS. [software], [cit. 2017-02-25]. Dostupné z: http://lesscss.org/

[29] Resig, J.: jQuery. [software], [cit. 2017-02-25]. Dostupné z: https://jquery.com/

[30] González, S.; Zaefferer, J.: jQuery UI. [software], [cit. 2017-02-25]. Dostupné z: https://jqueryui.com/

[31] client IO s.r.o.: JointJS. [software], [cit. 2017-04-25]. Dostupné z: https://www.jointjs.com/opensource/

[32] jsPlumb, Inc.: jsPlumb. [software], [cit. 2017-04-25]. Dostupné z: https://jsplumbtoolkit.com/

[33] Adeyemi, T.: Interact.js. [software], [cit. 2017-04-25]. Dostupné z: http://interactjs.io/

[34] jQuery Connecting line. [software], [cit. 2017-04-25]. Dostupné z: http://www.jqueryscript.net/other/jQuery-Plugin-To-Connect-Two-Html-Elements-with-A-Line.html

[35] HTML SVG Connect. [software], [cit. 2017-04-25]. Dostupné z: http://www.jqueryscript.net/other/jQuery-Plugin-To-Connect-HTML-Elements-with-Lines-HTML-SVG-Connect.html

[36] Mikowski, M. S.: jquery.event.gevent. [software], [cit. 2017-04-25]. Dostupné z: https://github.com/mmikowski/jquery.event.gevent/

[37] Smith, I.: TaffyDB. [software], [cit. 2017-04-25]. Dostupné z: http://taffydb.com/

[38] Mikowski, M. S.: uriAnchor. [software], [cit. 2017-04-25]. Dostupné z: https://github.com/mmikowski/urianchor/

[39] HubSpot: Vex. [software], [cit. 2017-04-25]. Dostupné z: http://github.hubspot.com/vex/docs/welcome/

[40] Harborow, B.; aj.: Bootstrap. [software], [cit. 2017-04-25]. Dostupné z: http://getbootstrap.com/

[41] ZURB, Inc.: Foundation. [software], [cit. 2017-04-25]. Dostupné z: http://foundation.zurb.com/

[42] Google: Material Design Lite. [software], [cit. 2017-04-25]. Dostupné z: https://getmdl.io/index.html

[43] Yahoo! Inc.: Pure.css. [software], [cit. 2017-03-07]. Dostupné z: https://purecss.io

[44] Shmelev, R.; aj.: Fontello. [software], [cit. 2017-04-25]. Dostupné z: http://fontello.com/

[45] Gandy, D.: Font Awesome. [software], [cit. 2017-04-25]. Dostupné z: http://fontawesome.io/

[46] The Inkscape Project: Inkscape. [software], [cit. 2017-04-25]. Dostupné z: https://inkscape.org/en/

[47] Evolus: Pencil project. [software], [cit. 2017-04-25]. Dostupné z: http://pencil.evolus.vn/

[48] Twitter: Bower. [software], [cit. 2017-02-25]. Dostupné z: https://bower.io/

[49] Alman, B.: Grunt. [software], [cit. 2017-02-25]. Dostupné z: https://gruntjs.com/

[50] Williams, J.: JSDoc. [software], [cit. 2017-05-01]. Dostupné z: https://github.com/jsdoc3/jsdoc/

[51] Giraudel, H.: What Nobody Told You About Sass's @extend. [online], [cit. 2017-04-25]. Dostupné z: https://www.sitepoint.com/sass-extend-nobody-told-you/

[52] Decker, K.: Handlebars. [software], [cit. 2017-04-27]. Dostupné z: http://handlebarsjs.com/

[53] Sullivan, N.: Reflows & Repaints: CSS Performance making your JavaScript slow? [online], [cit. 2017-04-25]. Dostupné z: http://www.stubbornella.org/content/2009/03/27/reflows-repaints-css-performance-making-your-javascript-slow/

[54] Gutin, L.: Testing Accordion Menu Designs & Iconography. [online], [cit. 2017-04-25]. Dostupné z: https://www.viget.com/articles/testing-accordion-menu-designs-iconography/

[55] Williams, G.; aj.: FontForge. [software], [cit. 2017-04-25]. Dostupné z: https://fontforge.github.io/en-US/

[56] Nielsen, J.: Why You Only Need to Test with 5 Users. [online], [cit. 2017-05-01]. Dostupné z: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Příloha **A**

Seznam použitých zkratek

API Application interface

XML Extensible markup language

JSON Javascript object notation

UI User interface

SPA Single page application

HTML HyperText markup language

CSS Cascading style sheets

SASS Syntactically awesome style sheets

DOM Document object model

AJAX Asynchronous Javascript and XML

SVG Scalable vector graphics

IE Internet Explorer

OS Operation system

MVC Model-View-Controller

REST Representational state transfer

URI Uniform resource identifier

UUID Univarsaly unique identificator

UUCID Univarsaly unique client identificator

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `doc` ..... adresář s dokumentacemi
  - `dev` ..... dokumentace zdrojových kódů
  - `UI` ..... specifikace uživatelského rozhraní
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF