VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

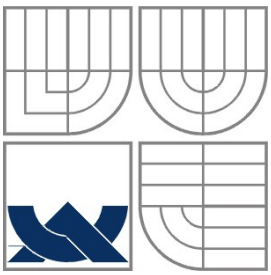
UŽIVATELSKÉ ROZHRANÍ PRO INTERAKTIVNÍ WEBOVÉ STRÁNKY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

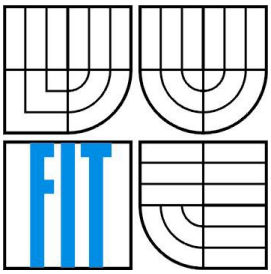
AUTOR PRÁCE
AUTHOR

JIŘÍ MAŇÁSEK

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

UŽIVATELSKÉ ROZHRANÍ PRO INTERAKTIVNÍ WEBOVÉ STRÁNKY

USER INTERFACE FOR INTERACTIVE WEB PAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ MAŇÁSEK

VEDOUCÍ PRÁCE
SUPERVISOR

BERAN VÍTĚZSLAV, Ing., Ph.D.

BRNO 2013

Abstrakt

Cílem práce je vytvořit webový nástroj usnadňující a urychlující vytváření dynamických webových stránek umísťováním widgetů libovolně do prostoru stránky. Každý tento widget zobrazuje konkrétní obsah. Taky je možné vytvářet nové či upravovat stávající podle potřeby nebo pouze využít již existujících. V tomto dokumentu je popsáno, jakým způsobem byla tato aplikace vytvářena a jak vypadá výsledek.

Abstract

Goal of this thesis is to create web based tool which simplifies and accelerates the process of making dynamic web pages by placing widgets arbitrarily into the page area. Every widget can display specific content. It is also possible to develop new widgets, modify them or just use the ones that already exists. This document describes how this application was created and the result of that process.

Klíčová slova

web, webová aplikace, framework, tvorba webových stránek, widgety, Web 2.0

Keywords

web, web application, framework, web page creation, widgets, Web 2.0

Citace

Jiří Maňásek: Uživatelské rozhraní pro interaktivní webové stránky, bakalářská práce, Brno, FIT VUT v Brně, 2013

Uživatelské rozhraní pro interaktivní webové stránky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vítězslava Berana, PhD.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Maňásek
13.5.2013

Poděkování

Chtěl bych poděkovat Ing. Vítězslavu Beranovi, PhD., vedoucímu této bakalářské práce, za poskytnutí velmi dobrých pracovních podmínek a pomoc při zpracování celého projektu.

Obsah

1 Úvod

Obor informační technologie se dnes velmi rychle vyvíjí. Ze dne na den vznikají nové technologie, postupy, nástroje a požadavky. To platí i pro vývoj webových stránek. Stránky se málokdy vytvářejí z ničeho. Využívají se nejrůznější modely, knihovny, frameworky, šablony a doba vývoje se zkracuje na minimum.

Podle úrovně svých znalostí v oboru tvorby webových stránek si pak uživatel vybírá jakou cestou se vydá. Pro velmi pokročilé není problém vytvořit jednoduchou stránku rychle a efektivně, pro složitější stránky si pak většina vývojářů vybírá nástroje, které za ně vytvoří alespoň základ jejich práce. Nezkušený uživatel ale musí sáhnout po co nejjednodušší variantě, která mu umožní vytvářet stránky i s minimálními znalostmi v tomto oboru. Využívá pak nejrůznější webové šablony a generátory, poskytující kompletní webovou aplikaci, kde uživatel poté jenom spravuje svoje příspěvky a obsah.

Tato práce se bude zabývat vytvořením nástroje pro usnadnění či urychlení vývoje interaktivní webové stránky. Uživatel si navrhne, jak bude jeho stránka rozvržená a umístí jednotlivé stavební kameny – widgety. Každý takovýto widget bude plnit určitou úlohu a zobrazovat konkrétní obsah. Rychlým a snadným postupem bez nutnosti znalosti webových technologií tak vzniká webová stránka, která se dá kdykoliv editovat a je snadné zachovat aktuálnost jak obsahovou tak vzhledovou.

První kroky práce se zabývají dostupnými technologiemi a postupy, které dnešní webový vývojář používá. Jedná se o základní i pokročilé webové technologie a knihovny. Z těchto získaných znalostí a přehledu byl potřeba udělat výběr nejvíce vhodných. Poté, když znám, co je možné využít a jak, je vytvořen návrh aplikace a popsán způsob, jakým budou jednotlivé její části spolupracovat. Předimplementační návrh ale nemusí být ještě úplně spolehlivý a i technologie, se kterými počítal, se můžou nakonec prokázat jako nevhodné. Z těchto důvodů proběhly experimentální implementace, různé testy a výsledky těchto procesů byly vyhodnoceny za cílem vytvoření finálního návrhu aplikace. Na konci této části je známá a z větší části i vytvořena kompletní aplikace a její struktura už bude neměnná. Posledním krokem bylo spojení všech experimentálních implementací, jejich odladění a vytvoření kompletní funkční implementace.

Na konci tohoto dokumentu by měl čtenář chápat, jak výsledná aplikace funguje, jakým způsobem a za jakým účelem byla vytvořena a jaké postupy a technologie k tomu využity byly, ale i proč některé dostupné tyto technologie byly zamítnuty a v aplikaci nevyužity.

2 Teorie

Při vytváření podkladů pro realizaci této aplikace bylo potřeba nastudovat několik základních věcí. Jelikož se pohybujeme v oblasti tvorby webových stránek, první kroky vedly ke klasickým nástrojům pro tvorbu webových aplikací jako je HTML, CSS, PHP. S tímto balíkem ovšem nelze vystačit při tvorbě stránek s komplikovanými dynamickými prvky, ukládáním dat, asynchronní komunikací s databází a dalšími. Proto bylo potřeba sáhnout po dalších technologiích [18]. Jedná se zejména o skriptovací jazyk JavaScript, databázi MySQL, asynchronní komunikaci Ajax ve formátu XML a knihovny a frameworky dále rozšiřující funkčnost.

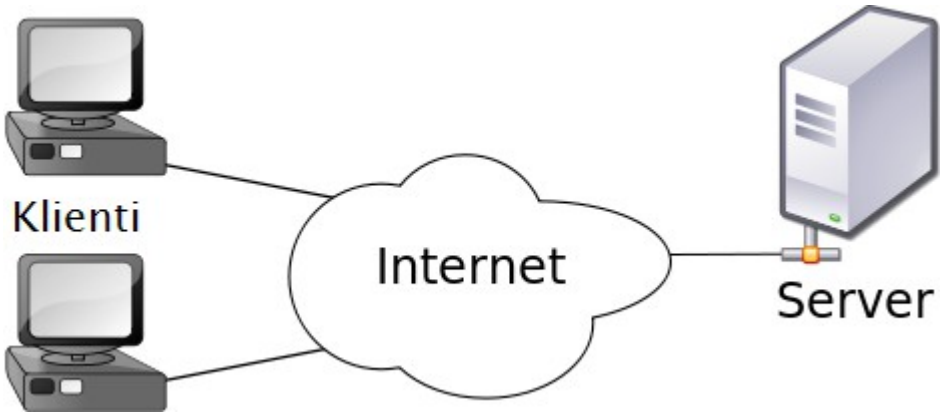
Tímto je přibližně pokryta programová stránka aplikace. Ovšem jak tohle všechno co nejlépe spojit dohromady? Existuje velké množství návrhových vzorů a postupů zodpovídajících tuto otázku. Dnešní webové frameworky, knihovny a další přístupy většinou staví na návrhovém vzoru MVC(Model-View-Controller) [15] a jeho příbuzných. Zároveň je už několik stránek, které se více či méně podobají této práci a dá se v nich najít mnoho odpovědí.

Tato kapitola popisuje, co jednotlivé technologie a postupy umožňují, co se s nimi dá řešit a taky jakým způsobem se dnes využívají.

2.1 Klient – server model

Většina dnešních internetových aplikací je založena na modelu klient-server [17], včetně právě této. Tento model nám říká, že v síti se vyskytují dva druhy počítačů a sice klient a server. Server je počítač, který nabízí klientům nějaké služby, klient je počítač, který tyto služby serveru využívá.

Komunikace pak vypadá jako série zpráv mezi serverem a klientem ve tvaru požadavek-odpověď.



Obr. 1: Klient-server [17]

Pro tyto zprávy musí být vytvořen komunikační protokol, který zajistí, že server a klient si navzájem budou rozumět, pokud dodrží komunikační protokol.

Obr. 1 ukazuje uspořádání serveru a jeho klientů. Pokud chce klient využít služeb serveru, kontaktuje ho přes síť internet a server mu tyto služby může poskytnout. Většina serverů je schopná naráz obsluhovat více klientů, ale není to pravidlem. Někdy je využit výlučný přístup, kdy klienti čekají, než server dokončí obsluhu aktuálního požadavku.

Typy klientských aplikací
Aplikace pracující na principu klient-server se pak může lišit v tom, kde je vykonávána jaká činnost, zda na serveru nebo u klienta:

- obsáhlý klient
  Obsáhlý klient provádí všechny požadované operace sám a nemusí spoléhat na server.
- tenký klient
  Tenký klient využívá hlavně zdrojů z hostitelského počítače. Zaměstnáním takového klienta je víceméně jen graficky zobrazovat data, poskytované aplikačním serverem, který vykonává převážnou část všech vyžadovaných operací. Výhodou takového klienta je vysoká ovladatelnost a pružnost.
- hybridní klient
  Jedná se o kombinaci dvou výše zmíněných typů. Takovýto klient dokáže vykonávat nějakou část činnosti nezávisle na serveru, ale server k tomu potřebuje, nejčastěji jako uložiště dat

Rozdělení typů klientských aplikací a jejich definice byly převzaty z [20].

2.2 Návrhové vzory

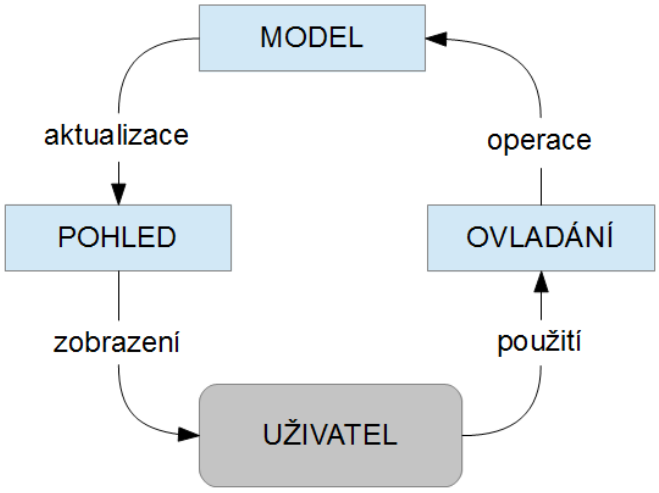
Návrhový vzor představuje obecný postup, kterým se řeší určitý, již dříve vyřešený problém. V našem případě jde o návrh webové aplikace a jejich podčástí.

2.2.1 Model-View-Controller

Model-View-Controller [15], zkráceně MVC, nám říká, jaká by měla být architektura naší aplikace. Zde budeme používat české pojmy pro tento návrhový vzor. MVC definuje tři základní komponenty:

- **model** (model) - reprezentace dat (informací)
- **pohled** (view) – převádí informace z modelu do podoby, vhodné pro zobrazení uživateli
- **ovládání** (controller) – provádí změny v komponentě **model** a tím nepřímo určuje, co bude zobrazeno v komponentě **pohled**

Takovéto rozdělení pak zajišťuje možnost změn v jednotlivých komponentách bez toho, aniž by tyto změny měly větší dopad na ostatní komponenty. Např. pokud provedeme změny v **model**, nemělo by to nijak zásadně ovlivnit činnost komponent **ovládání** a **pohled**.



Obr. 2: Komunikace v MVC

Operace prováděné v MVC (viz *obr. 2)*:

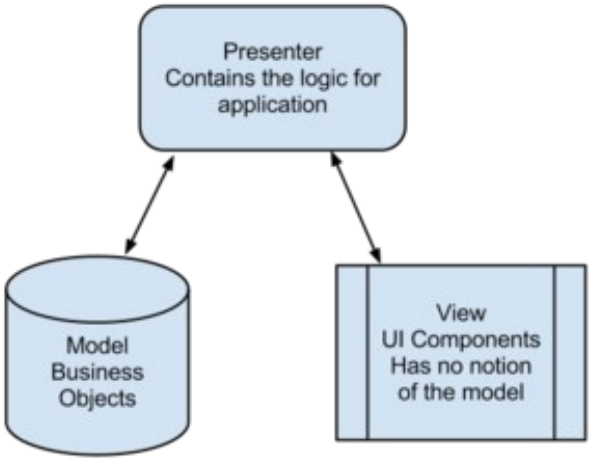
- *aktualizace* - **pohled** je aktualizován informacemi z **modelu**
- *zobrazení* - uživatel vidí zobrazené informace pomocí **pohledu**
- *použití* - uživatel ovládá aplikaci pomocí **ovladače**
- *operace* - podle požadavků uživatele interpretovaných **ovládáním,** jsou upravovány data uložené v **modelu**

2.2.2 Model-View-Presenter

Model-View-Presenter [16], zkráceně MVP, je návrhový vzor vycházející z výše zmíněného MVC.

MVP definuje tři základní komponenty v architektuře aplikace:

- **model –** rozhraní pro manipulaci a přístup k datům
- **view -** zobrazuje informace a směruje uživatelské požadavky na komponentu **presenter**
- **presenter –** formátuje data z komponenty **model** srozumitelně pro komponentu **view** a vykonává požadavky uživatele na komponentou **model**



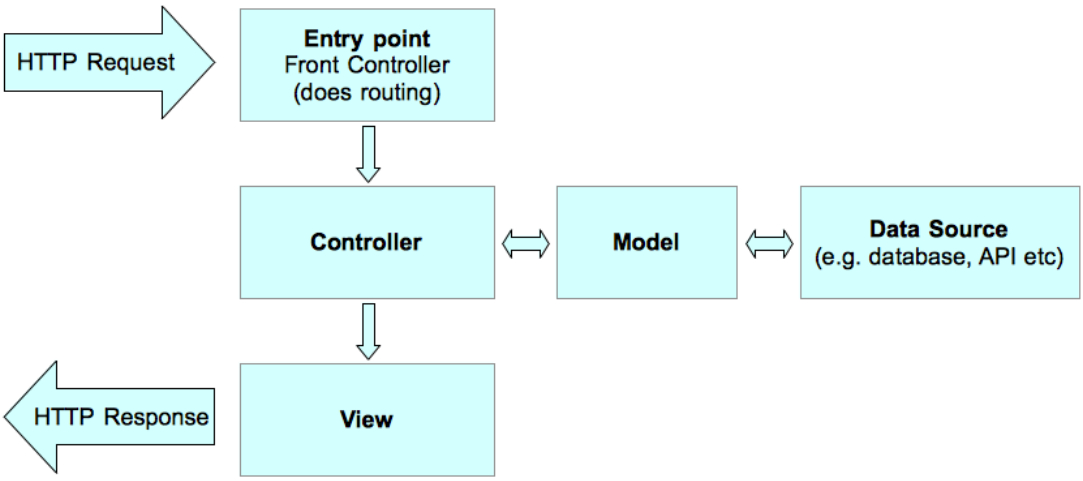
Obr. 3: Komunikace v MVP [16]

Z obr. 3 je vidět, že komponenta **presenter** v tomto vzoru hraje jakousi roli prostředníka v komunikaci mezi komponentami **model** a **view**. Komponenta **view** tedy nemá ponětí, jak jsou uložena data v **model** a naopak komponenta **model** netuší, jak se její data budou zobrazovat uživateli.

MVP opravdu efektivně rozděluje jednotlivé klíčové části aplikace a umožňuje tak vývojáři manipulaci s ním bez toho, aniž by potřeboval zasahovat do zbylých částí. Ovšem tento vzor je náročnější na aplikační logiku, umístěnou v komponentě **presenter**.

2.2.3 Front Controller

Při použití MVC návrhového vzoru [15] u vývoje webové stránky nám pak vzniká typicky struktura Front Controller. Front Controller, funguje tak, že všechny HTTP požadavky jsou směřovány přes jeden bod (soubor; typicky index.php), ve kterém se vyhodnotí a jsou vyvolány potřebné akce. Na *obr. 4* je znázornění, jak probíhá v takové architektuře zpracování HTTP požadavku.



Obr. 4: Front Controller pattern (Zdroj: http://addyosmani.github.io/backbone-fundamentals/)

2.3 Technologie pro Web 2.0

Web 2.0 [18] je označení webové stránky, která využívá velké množství nejmodernějších webových technologií. Její obsah se aktualizuje asynchronně, jsou zde umístěny videa a animace, opomenuty nejsou ani prvky založené na zásuvných modulech a mnoho dalších. Web 2.0 komunikuje s uživatelem svižně, má pokročilé uživatelské rozhraní, zajímavou grafickou podobu. To vše je postaveno nad databází, v níž je uloženo vše od obsahu stránky až po uživatelskou kustomizaci webu a informace pro našeptávání u vyhledávání.

Technologie, o kterých zde bude řeč, jsou rozděleny do dvou kategorií (viz. kapitoly 2.3.1, 2.3.2).

2.3.1 Klasické webové stránky

V době, kdy ještě webové stránky nebyly plně dynamické a neměnili svůj obsah bez znovunačtení (aktualizace), se používali technologie HTML, CSS, PHP, MySQL. Ty pro všechny takové stránky stačily. Dnes je tomu už jinak, ale tyto technologie pořád tvoří základ, bez něhož se neobejdeme.

HTML a CSS

Dvojice, umožňující zobrazení informací v prohlížeči koncovému uživateli. Díky CSS jsou HTML prvky jakkoliv pozicovatelné, nastavitelné, barevné, skryté nebo zobrazené a mnoho dalšího. U HTML nás bude později zajímat hlavně jeho reprezentace v podobě DOM modelu, se kterým se dá efektivně pracovat pomocí skriptovacího jazyku JavaScript. HTML a CSS jsou standardizovány skupinou W3C [12] a přehledně popsány na stránce [13].

PHP a MySQL

Představuje datový backend aplikace umístěný na webovém serveru. Umožňuje nám ukládáni, úpravu, filtrování a výběr dat, které jsou následovně zpracovány a odeslány klientovi, ať už v podobě

webové stránky nebo jako XML data. Autoři PHP i MySQL poskytují velmi kvalitní dokumentaci, ze kterých byly často čerpány informace [4][11], doplněné knihou [3] a webovou stránkou [12].

2.3.2 Vysoce interaktivní webové stránky

Javascript a Ajax
JS a Ajax jsou technologie, které umožňují práci se samotným HTML. Spouští se na straně uživatele. Pomocí Javascriptu můžeme téměř neomezeně měnit stránku zobrazenou uživateli bez nutnosti ji znovu načítat. Ajax zase dovoluje komunikovat ze serverem na pozadí opět bez nutnosti načítat nějakou stránku. Dohromady tyto technologie vytváří velmi mocný nástroj pro tvorbu dynamických interaktivních stránek. [6][7][8]

jQuery
Knihovna postavená nad JavaScriptem. Definuje mnoho metod pro práci s DOM modelem nejenom HTML stránek a jeho událostí. Obsahuje taky podknihovnu jQuery UI, obsahující množství prvků pro tvorbu uživatelských rozhraní, metody pro animace, umožňuje manipulaci s rozměry a umístěním prvků HTML stránky další funkce. Pravděpodobně nejdůležitější část a taky úplný základ jQuery jsou selektory. Díky tomuto nástroji je možné velice jednoduše přistupovat k libovolnému prvku stránky podle jeho ID, třídy, jména atd. [9]

jQuery-UI
Volné rozšíření předchozí knihovny jQuery, které nabízí mnoho funkcí, projevujících se na stránce graficky. Je zde paleta nástrojů, umožňujících jednoduché manipulace s prvky DOM (pohyblivost, změna rozměrů, animace), ale taky složité prvky, realizující celé miniaplikace umístitelné do stránky (kalendář, hodiny, poznámky). Pro tuto aplikaci nás zajímá zejména manipulace s DOM modelem. [10]

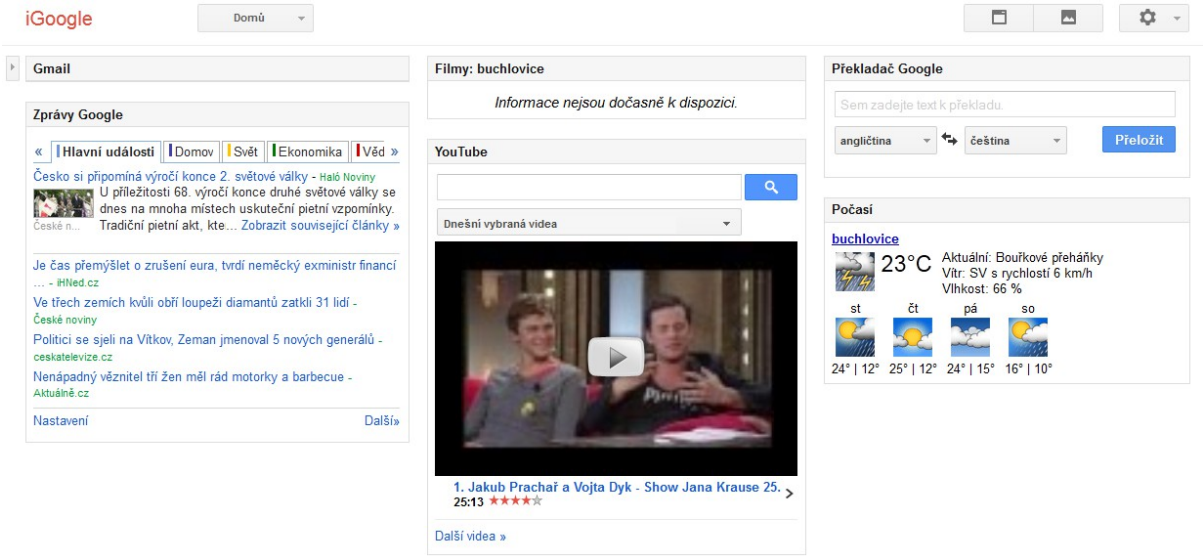
Nette
PHP framework, umožňující vývoj webových stránek bez bezpečnostních rizik a se zaměřením na znovupoužitelnost kódu. Podporuje návrhový vzor MVC (respektive MVP). Obsahuje šablonovací systém *latte*. Obsahuje nástroje pro využití Ajaxu, pro práci s databází, pro ladění našeho webu, pro zajištění bezpečnosti a další. (Zdroj: http://nette.org)

Backbone.js
Technologie pro usnadnění či spíše realizaci synchronizace uložených dat a dat zobrazených na webové stránce. Backbone.js předvádí vlastní formu návrhového vzoru MVC. Jeho účelem je udržet data z komponenty *model* synchronizované s daty zobrazenými uživateli na komponentě *view*. Toto je realizováno napojením jednotlivých prvků HTML stránky na model. Pokud dojde k nějaké události měnící data na některé straně, tyto data jsou synchronizovány. [1]

Underscore.js
Soubor nástrojů a funkcí, který usnadňuje práci s objekty, seznamy, poli, událostmi. Technologie je primárně určena pro použití s *Backbone.js* a *jQuery*. [2]

2.4 Podobná řešení

iGoogle
Společnost Google nabízí libovolně přizpůsobitelnou stránku. Uživatel si dle vlastního výběru přidává widgety z velmi široké nabídky (od her po finanční zprávy), podle toho co chce vidět. Projekt bohužel není otevřený pro vývoj komunitou a je plně řízen společností Google. [23]



Obr. 5: Ukázka vzhledu iGoogle[23]

webnode.cz
Tato aplikace přistupuje k vytváření webových stránek cestou obrovského množství šablon. Uživatel si některou vybere a poté přizpůsobuje obsah podle potřeb. Šablony jsou rozděleny do třech základních kategorií – osobní prezentace, firemní prezentace, internetový obchod. V každé je pak mnoho dalších podkategorií. [24]

wordpress.org, drupal.org
Klasické CMS (Content Management Systémy), umožňující jednoduchou správu webových stránek. Jedná se o projekty s přístupným zdrojových kódem, což umožňuje vývoj komunitou, který je velice znát. Pro oba systémy existuje nepřeberné množství šablon a vzhledů a ještě je vždy možné si vytvořit vlastní. Wordpress je velmi rozšířený systém (na 15% všech webových stránek). [21][22]

netvibes.com
Opět podobné řešení jako iGoogle. Uživatel dostává do rukou interaktivní webové rozhraní pro umísťování widgetů a samozřejmě obrovské množství widgetů, ze kterých se dá vybírat. Opět je bohužel nedostupný zdrojový kód a není možná jakákoliv vlastní modifikace. [25]

iNettuts - How to Mimic the iGoogle Interface
Online JavaScript a Ajax tutoriál, demonstrující možnosti těchto technologií na vytvoření napodobeniny uživatelského rozhraní iGoogle. [26]

Na základě těchto podobných řešení bych chtěl zdůraznit, v čem se bude tato práce více či méně lišit od dostupných projektů. Bude se jednat o aplikaci s otevřeným zdrojovým kódem. Vzhled webové stránky bude vytvářen spíše než množstvím tak sestavením dílčích částí a šablony budou určovat minimum stránky. Nízká omezení widgetů umožňují realizovat nejdivočejší požadavky. Kód aplikace bude umístěn na vlastním serveru, nikoliv centralizovaně na jednom serveru, kde si uživatel založí svůj účet a zde se bude aplikace spouštět.

3 Návrh řešení

Zde je rozebírán cíl práce, specifikace aplikace a předimplementační návrh řešení. Tento návrh ukazuje jak bude vypadat, jak se bude ovládat a taky jakým způsobem bude aplikace implementována.

3.1 Rozbor cíle

Cíl této práce se dá rozebrat ze dvou pohledů. Jeden je vymyšlení nového uživatelského rozhraní, které bude nejvhodnější pro tuto aplikaci. Objevují se zde otázky:

- Jakým způsobem se budou widgety rozmísťovat po stránce, tedy jak konkrétně se bude aplikace ovládat za cílem vytvoření vlastní webové stránky?
- Jak se budou vybírat widgety, vytvářet jejich instance a měnit jejich nastavení?
- Na jaké operace uživatelské rozhraní rozdělit, aby bylo přehledné, jednoduché, ale funkční?

Druhý pohled je z hlediska technologického a implementačního. Jakmile najdu odpovědi na předchozí otázky, automaticky dostávám nové:

- Jak dané uživatelské rozhraní zrealizuji?
- Jaké technologie se mně zde budou hodit a které by práci spíše zatížily?
- Jaká bude struktura aplikace, aby co nejlépe využila použitých technologií, zachovala rozšiřitelnost a přehlednost aplikace a pořád ještě byla dostatečně rychlá?
- V jakém prostředí by měla být tato aplikace využita?
- Jak náročná (výkonově, paměťově, síťově, implementačně) bude tato aplikace?
- Jak se bude pracovat s daty, kde a jak se budou ukládat?

Na oba tyto pohledy a jejich otázky se budou snažit, ať přímo nebo nepřímo, odpovědět následující kapitoly. Některé otázky mohou zůstat otevřeny a ponechány až do fáze experimentování a přípravy, kde budou na skutečně implementovaných funkčních částech aplikace otestovány tyto otázky.

3.2 Specifikace aplikace

Tato kapitola do detailu rozebírá, jak by měla cílová aplikace vypadat a navrhnout jedno nebo více řešení daných problémů. Z těchto řešení bude pak vybráno to nejlepší nebo jich bude použito více a z nich sestavena výsledná aplikace.

3.2.1 Rozvržení obrazovky

Uživatel bude mít možnost nastavit rozložení obrazovky podle svých představ v několika režimech (volné dělení, vertikální/horizontální sloupce, volné rozmístění). Takovéto rozložení pak bude sloužit jako šablona pro umísťování widgetů na stránku. Díky technologické možnosti udělat stránku vysoce interaktivní, bude vytváření rozvržení obrazovky uživatelsky příjemné. Provedené změny se budou ihned zobrazovat před uživatelem a ten hned uvidí, jestli provádí se stránkou to, co chce. Pokud uživatel není spokojený se změnami, které provedl, může celou úpravu zahodit a vrátit se k původně uloženému rozvržení stránky.

Rozvržení volným dělením

Podstatou je vytváření polí pro widgety na stránce dělením. Pokud začínáme s vytvářením úplně nového rozvržení, vidíme pouze prázdnou stránku. Kliknutím do jejího prostoru ji rozdělíme na dvě nové pole, buď horizontálně nebo vertikálně. Tyto vzniklé pole je poté opět možno rozdělit horizontálně nebo vertikálně a taky přizpůsobovat jejich velikost tomu, co potřebujeme. Tímto způsobem rozdělíme stránku na tolik polí,, kolik chceme o libovolné velikosti a do nich umístíme widgety.

Rozvržení horizontální/vertikální sloupce
U tohoto rozvržení je nutné si rozmyslet, jestli chceme stránku organizovat po sloupcích horizontálních nebo vertikálních. Horizontální sloupce znamenají, že nám budou vznikat pole pro widgety se stejnou výškou, řazené v horizontálních sloupcích. Vertikální sloupce obsahují pole se stejnou šířkou, pod sebou umístěné v několika vertikálních sloupcích. V jednotlivých polích pak budou umístěny widgety a zobrazen jejich obsah.

Rovržení volným umístěním
Pro maximální svobodu při vytváření rozvržení stránky může uživatel použít tento typ. Kterýkoliv widget je možné umístit kamkoliv na stránku, libovolně ho zvětšovat, zmenšovat a dále přesunovat (i za hranice obrazovky) a taky nastavovat zIndex jednotlivých widgetů. zIndex určuje, jakým způsobem se budou případně pole překrývat pokud byly umístěné přes sebe. Vyšší hodnota zIndex znamená, že pole bude překrývat to s nižší hodnotou zIndex. Po uložení stránky je možné toto rozvržení kdykoliv začít znovu editovat a přizpůsobovat jej tak svým potřebám.

3.2.2 Widgety a galerie widgetů

Widgety tvoří část aplikace, která zodpovídá za vlastní obsah a vzhled stránky. Poté co si uživatel vytvoří rozvržení obrazovky podle svých potřeb, do něj začne umísťovat widgety a to tak, aby pokryl účel stránky. Každý widget je tedy vlastně menší část stránky a jejího obsahu. Společně dohromady tvoří celek. Jednotlivé tyto widgety mezi sebou mohou komunikovat, sdílet obsah a sloužit jeden pro druhého jako ovládací prvek.

Hlavní důvod použití widgetů (a taky jeden z důvodů, proč je tahle aplikace vyvíjena) je možnost jejich recyklace. Pokud někdo vytvoří kvalitní a dobře přizpůsobitelný widget, je pravděpodobné, že se bude hodit i někomu jinému. Různými formami sdílení se tedy může začít vytvářet komunita, která bude produkovat kvalitní widgety a kolekce widgetů, plnící nejrůznější účely na webových stránkách. Méně zkušení uživatelé, kteří nedokážou vytvářet své vlastní widgety, pak takovouto paletu použijí při tvorbě vlastní stránky a budou schopní vytvořit komplikovanou webovou interaktivní aplikaci bez nutnosti hluboké znalosti vývoje webových stránek.

Nainstalované a povolené widgety v aplikaci budou zobrazeny v galerii viditelné při vytváření rozvržení webu. Každý widget je možné z této galerie vzít a umístit do stránky. Při umístění na stránku se vytvoří instance widgetu, které obsahuje vlastní nastavení a obsah. Je tak možné, aby se jeden modul vyskytoval na stránce i vícekrát a měl různé vlastnosti.

Nastavení vlastností widgetu a jeho obsahu bude probíhat přímo na stránce editace bez nutnosti přecházet někam jinam. Každý widget definuje formulář pro svoje nastavení, které může být velmi komplexní nebo naopak nemusí nabízet vůbec žádné.

3.2.3 Nastavení

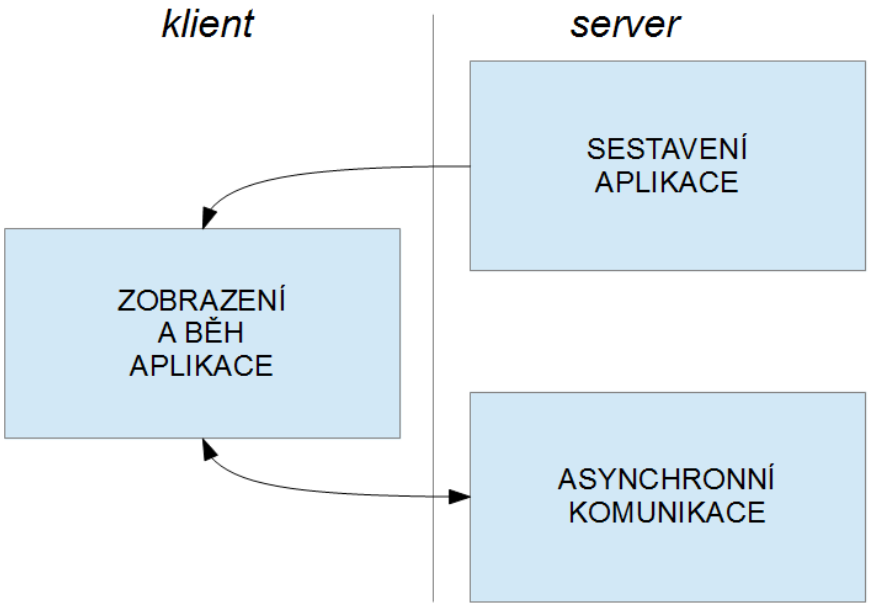
Stránka *nastavení* umožňuje nastavení vlastní aplikace, přehled a správu nainstalovaných widgetů.

3.3 Struktura aplikace

V kapitolách 3.2.1 – 3.2.4 nastíním, jakým způsobem bude strukturovaná implementace aplikace. V každé kapitole je blokové schéma, týkající se nějaké části aplikace a poté bude toto schéma dostatečně důkladně rozebráno, popsáno a zdůvodněno. Některé kapitoly upřesňují ty předchozí, a proto mohou být informace vypuštěny a umístěny až v kapitole upřesňující.

3.3.1 Top-level

Top-level je úroveň neobsahující žádné detaily aplikace a sloužící pro úvod do problematiky tohoto návrhu. Jednotlivé části v top-level budou dále upřesněny a podrobně rozebrány.



Obr. 6: Top-level architektura

Na *obr.6* je vidět blokové schéma nejstručněji popisující základní strukturu aplikace. Je rozdělena na dvě části *klient* a *server.* Tím je myšleno, že existují části aplikace běžící na serveru a taky části běžící u klienta.

Serverová část obsahuje dvě jednotky – *sestavení aplikace* a *asynchronní komunikace. Sestavení aplikace* má na starosti výběr a seskládání aplikace/stránky, která bude stažena a spuštěna na straně klienta v jeho prohlížeči. Pří sestavování se musí vyhodnotit, co vše klient potřebuje, vložit do stránky základní informace o rozložení obrazovky, widgetech a nastavení aplikace. Klient z těchto dat pak dodatečnými dotazy už dokáže sestavit funkční stránku.

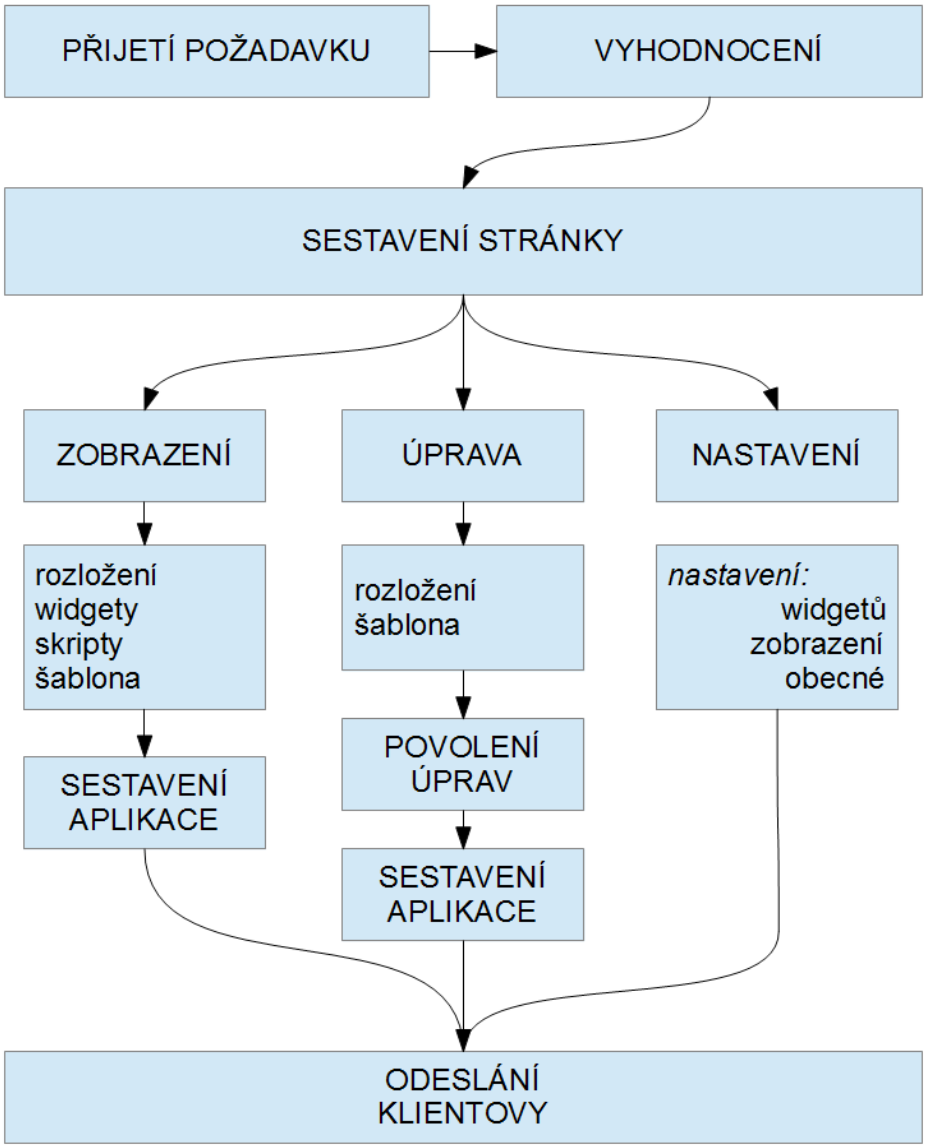
Asynchronní komunikace je rozhraní pro klienta, na které bude zasílat veškeré své dotazy. Přes toto rozhraní získá informace o uloženém rozvržení obrazovky, umístění widgetů, jejich nastavení a obsahu. Touto částí taky ukládá změny v nastavení aplikace, rozvržení obrazovky či nastavení widgetů do databáze.

Klientská část je, jak už bylo řečeno, část, která bude spuštěna v klientově prohlížeči. Obsahuje soubory skriptů realizujících zobrazení a obsluhu uživatelských požadavků. Jak bude zmíněno

v kapitole 3.2.2, obsluhují tři základní úlohy a sice zobrazení obsahu, úpravu rozvržení a widgetů a nastavení aplikace. Tato část je jedna dynamická webová stránka, která bude aktualizovat a měnit svůj obsah bez nutnosti znovunačtení celého obsahu, právě za pomoci komunikace se serverem přes dedikované rozhraní – *asynchronní komunikace.*

3.3.2 Sestavení stránky

Sestavení stránky je část aplikace, spouštěná na serveru. Má na starosti sesbírání všech potřebných informací, skriptů, knihoven a dalšího a jejich sestavení do webové stránky, která bude odeslána uživateli. Činnost se provádí na základě požadavku od uživatele, konkrétně otevření URL této webové aplikace. Podle typu požadavku generátor vytvoří adekvátní stránku a odešle.



Obr. 7: Průběh sestavení požadované stránky

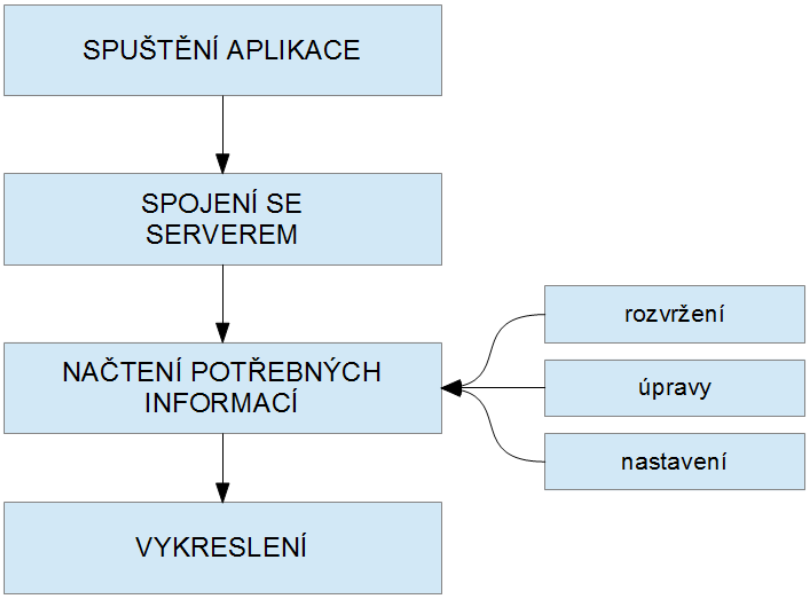
Na *obr. 7* vidíme, jak aplikace postupuje při sestavování stránky. Jakmile je rozpoznán uživatelský požadavek, postupuje se po jedné ze tří větví. Na konci každé vzniká jeden ze tří typů stránky – hlavní zobrazení, editace rozvržení, nastaveni – a tento je odeslán. Stránka musí mít všechno programové vybavení potřebné k činnosti, protože při svém běhu může žádat server pouze o uložení nebo získání informací z databáze.

Větev *zobrazení* vytvoří stránku zobrazující obsah, tedy výsledný vzhled webové stránky po dokončení veškerého nastavení (to co je cílem práce s touto aplikací). K tomuto je potřeba několik informací:

- zvolené rozvržení stránky a jeho typ – na základě tohoto bude vybrán soubor funkcí, zajišťujících správné vykreslení rozvržení, získávání informací ze serveru a další nastavení
- widgety umístěné na stránce – pouze orientační seznam; existuje jenom jeden soubor funkcí obsluhující vykreslování widgetů do stránky; informace o vlastních widgetech a další jejich nastavení si dodatečně stahuje stránka ze serveru
- skripty – soubor všech skriptů, které budou obsluhovat funkčnost stránky; některé jsou vybírány podmíněně na základě nastavení (viz. typ rozvržení stránky)
- šablona – ke stránce je připojen zvolený grafický vzhled prvků stránky

Poté, co jsou známy všechny informace a vybrány správné komponenty, je stránka sestavena do samostatně funkčního celku a tím je připravena pro spuštění u uživatele.

3.3.3 Zobrazení a běh



Obr. 8: Spuštění klientské aplikace

Jakmile uživatel získá od serveru funkční stránku, je nutné ji zavést. Stránka se musí přizpůsobit lokálnímu nastavení uživatelova prohlížeče, rozlišení jeho obrazovky, popřípadě preferovanému jazyku(*obr. 8).*

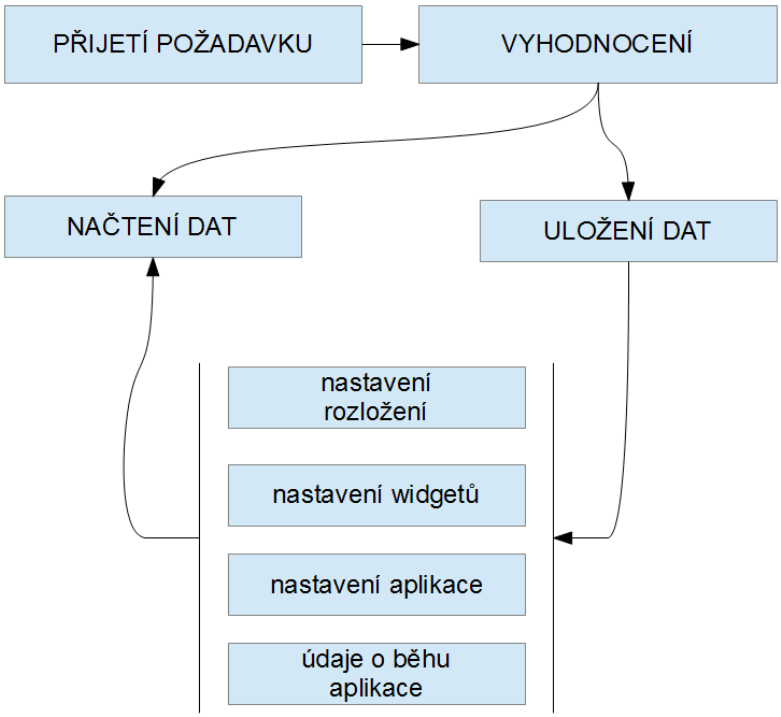
K vykreslení obsahu pořád chybí některé informace o umístění a zdrojích dat modulů. Toto získáme přes *asynchronní rozhraní* od serveru. Informace o obsazích se předají do widgetů a ty už se pak sami starají o aktualizaci těchto dat.

Pokud jsou všechny části nastaveny správně, stránku není potřeba aktualizovat – obsah se načítá automaticky či na požádání (hromadná aktualizace).

Spuštění stránky se neprovádí žádnou funkcí, ale pouze dokončením nastavení všech částí stránky.

3.3.4 Asynchronní komunikace

Asynchronní komunikace je rozhraní umožňující ukládat nebo načítat data z databáze za běhu aplikace u klienta bez toho, aniž by musel stránku celou znovu načítat.



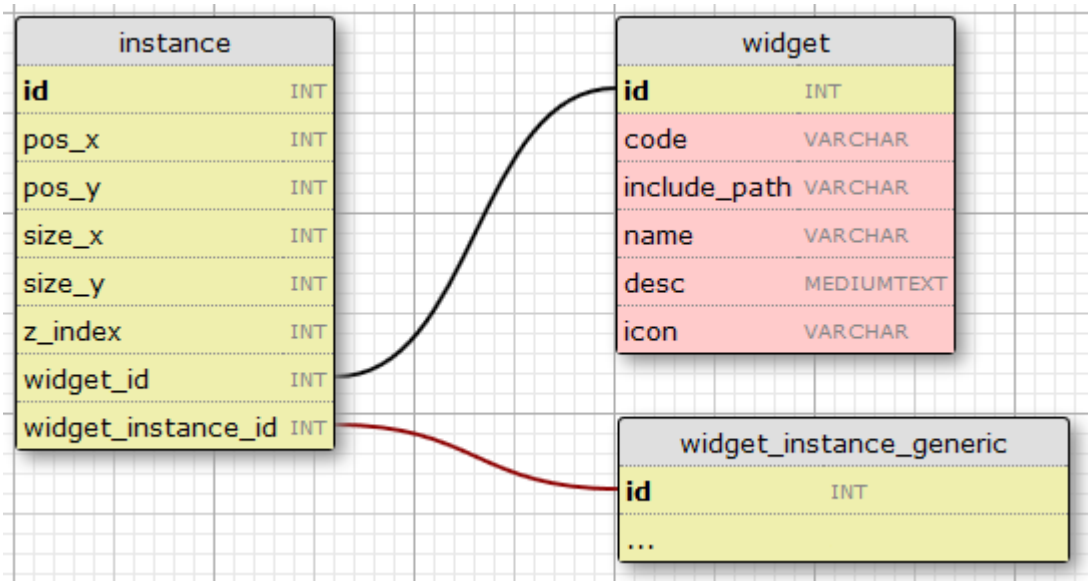
Obr. 9: Průběh asynchronní komunikace

Uživatelská aplikace zašle požadavek, ten je vyhodnocen a je provedena odpovídající akce – uložena přijatá data nebo vrácen výsledek s daty získanými z databáze.

3.4 Struktura databáze

Vlastní aplikace vyžaduje pouze jednoduchou strukturu obsahující 2 tabulky. Jediné, co je potřeba uložit, jsou informace o navrženém rozložení stránky, jaké widgety byly kde umístěny a informace o nainstalovaných widgetech.

Následující uvedené schéma databáze se týká pouze volného rozložení obrazovky. Pro ostatní typy je nutné strukturu rozšířit o jeden záznam. Tento typ rozložení byl zvolen pro finální implementaci, proto je zde uvedeno pouze toto schéma.



Obr. 10: Struktura databáze

Na *obr. 10* je znázorněno schéma databáze a její základní tabulky – *instance. widget* a *widget_instance_generic.* Tyto tři tabulky představují minimální databázovou strukturu, kterou aplikace vyžaduje pro svou funkčnost.

Tabulka *instance*
Data z této tabulky slouží pouze vlastní aplikaci pro uložení uživatelského rozložení stránky a widgetů na ní umístěné.

- **id –** unikátní identifikátor záznamu v tabulce
- **pos_x –** pozice X na stránce, na které začíná pole widgetu (horizontální odsazení od levého okraje)
- **pos_y** – pozice Y na stránce, na které začíná pole widgetu (vertikální odsazení od horního okraje)
- **size_x –** horizontální rozměr pole widgetu
- **size_y –** vertikální rozměr pole widgetu
- **z_index –** určuje pořadí widgetů při jejich vzájemném překrývání
- **widget_id** - označuje widget, který je v poli umístěn
- **widget_instance_id –** označuje instanci widgetu, která je v poli umístěna

Tabulka *widget*
Tabulka představuje seznam všech aktuálně nainstalovaných widgetů na serveru. Widgety, které zde nejsou uvedeny, ale mají na serveru nahrané svoje zdrojové soubory, nelze použít

- **id –** unikátní identifikátor záznamu v tabulce
- **code –** unikátní kódové označení widgetu
- **include_path –** cesta k souboru, který implementuje funkce widgetu
- **name –** plný název widgetu
- **desc –** popis widgetu
- **icon –** cesta k souboru s ikonou widget (není povinné)

Tabulka *widget_instance_generic*

Tato tabulka představuje pouze šablonu pro to, jak mají jednotlivé widgety ukládat svá data databázi. Není tedy definována žádná struktura, kterou musí dodržovat, pouze musí využít tuto tabulku (s libovolným unikátním názvem) a primárním klíčem **id**. Ostatní je už plně v režii widgetu.

- **id –** unikátní identifikátor záznamu v tabulce
- **…** - libovolné data potřebné pro widget(včetně dalších tabulek)

4 Příprava a experimenty

Před samotnou úplnou implementací jsem vyzkoušel vytvořit několik neúplných a nepřesných verzí pomáhajících při rozhodování, které nápady použít, jak by měla kompletní aplikace vypadá a jaké technologie se budou dát využít efektně a pomůžou při implementaci, a naopak vypustit ty které by práci spíše komplikovaly.

4.1 Použité technologie

Před vytvářením větších částí aplikace bylo na těch nejjednodušších útržcích vyzkoušeno, zda se konkrétní problémy dají řešit technologiemi, které byly v rámci této práce studovány a zda je nutné je vůbec do aplikace přidávat.

Příklady, na kterých byly technologie vyzkoušeny:

- rozvržení obrazovky
- drag-and-drop operace
- objekty v JavaScriptu a jejich vlastnosti
- experimentování s projektem TodoMVC [5]
- experimentování s frameworkem Nette (Zdroj: http://nette.org)

4.2 Typy rozvržení obrazovky

Volné dělení stránky

Vytvoření rozvržení tímto způsobem je velice jednoduché a rychlé. Ovšem uživatel je hodně omezen tím, jak je nucen využít prostor na stránce. Taky není možné vytvořit překrývání jednotlivých polí widgetů. Tento typ rozvržení byl implementován pomocí čistého Javascriptu i pomocí knihoven jQuery a jQuery-UI.

Horizontální/vertikální sloupce

Tento způsob byl implementován pomocí knihoven jQuery a jQuery-UI. Rozvržení omezovala uživatele při rozmísťování widgetů zvoleným počtem sloupců, protože nebyla dostupná možnost zobrazit jeden widget přes více sloupců. Dále vznikal problém při pokusu o přesun widgetů, které už na stránce byly umístěny.

Volné rozvržení

Tato varianta kompletně vyhovuje původnímu požadavku – nabídnout jiný přístup k sestavení stránky, než pomocí velkého množství šablon. Uživatel tedy může kamkoliv na stránku umístit kterýkoliv widget. Je zde několik nedostatků (přenositelnost mezi rozlišením, umístění obsahu na střed stránky), ty jsou ale převáženy výhodami (jednoduchost ovládání i implementace, přehlednost, svoboda v rozvržení, ...). Toto rozvržení bylo experimentálně implementováno (neúplná, nepřesná, neodladěná implementace čistě pro demonstraci funkčnosti) pomocí knihoven jQuery a jQuery-UI, které výrazně celý skript zjednodušily. Výsledek této implementace ověřil, že tato možnost rozvržení obrazovky je vhodnější než předchozí varianty v rámci této práce.

4.3 Částečná implementace

Následující tři verze aplikace ukazují rozdíly v použití jednotlivých technologií. V základu bylo vždy využito - na serverové straně – PHP, MySQL; na klientské straně – HTML, CSS, JS; pro asynchronní

komunikaci XML a Ajax. K těmto technologiím se pak přidávají další, popsané vždy u jednotlivých verzí.

Verze 1 – čistý Javascript
Tento pokus se ukázal být velice nešťastným. Celá aplikace je sice založena na JavaScriptu, ale bez použití knihoven a dalších technologií vznikal velice nepřehledný zdrojový kód, zbytečně složitý a neoptimalizovaný.

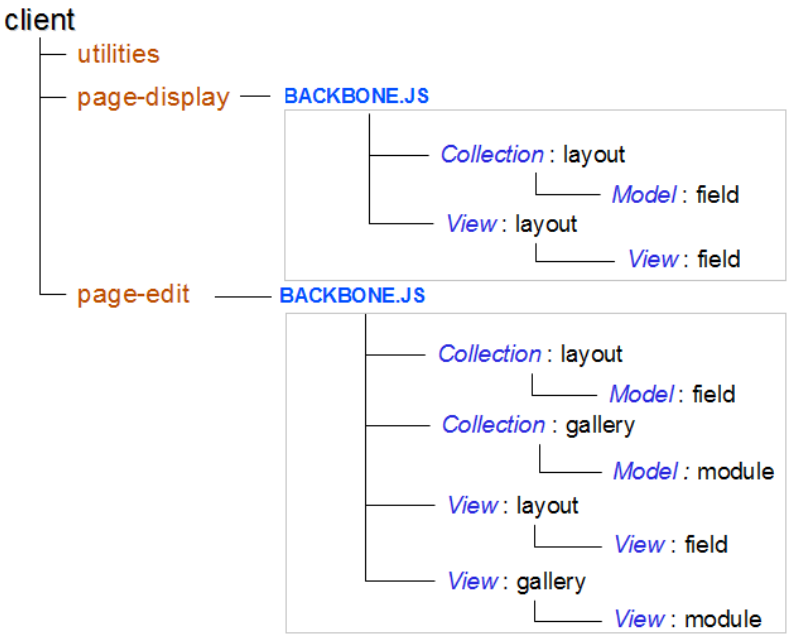
Například, neúplná implementace rozvržení obrazovky zabrala téměř 200 řádků kódu, což je zbytečně moc; jedná se o jednoduchou operaci.

Části aplikace, které byly implementovány v této verzi byly:
- rozvržení obrazovky
- synchronizace s databází

Další problém vznikal s přechodem na jiné prohlížeče. JavaScriptový kód, fungující v prohlížečích Mozilla Firefox a Opera, může dělat problémy Internet Explorer. Tyto problémy zanikají při použití technologií jako je jQuery.

Verze 2 – Underscore.js, Backbone.js, jQuery, jQuery-UI
Implementace této verze byla dotažena téměř do kompletního stavu. Chybělo několik menších detailů.

Backbone.js nutí programátora vytvářet strukturu, která svým způsobem vychází z návrhového vzoru MVC. Tato struktura má mnoho výhod (znovupoužitelnost, rozšiřovatelnost, jednoduché úpravy) a podporuje dobré programovací návyky.



Obr. 11: Struktura klientské aplikace založená na Backbone.js

Problém nastal při programování synchronizace mezi klientem a databází na serveru. Backbone.js definuje vlastní automatické metody, které zasílají nebo stahují data ze serveru. Tyto metody vyžadují

REST rozhraní [14] na straně serveru a k přenosu dat používají formát JSON. Toto se do aplikace příliš nehodí, hlavně kvůli již navrženému protokolu komunikace v XML formátu. Bylo tedy nutné všechny synchronizační metody, které Backbone.js definuje, přepsat tak, aby odpovídaly protokolu a komunikace se serverem fungovala. Komplexnost této knihovny také způsobovala, že aplikace běžela poněkud pomalu, nereagovala na uživatelské příkazy dostatečně rychle a byla náročná na výkon i spojení se serverem.

Verze 3 – jQuery, jQuery-UI
Verze 3 vychází ze struktury dané Backbone.js, upravené a přenesené do objektového návrhu pouze v JavaScriptu, za využití knihoven jQuery a jQuery-UI.

Opět byla implementována téměř celá funkčnost aplikace, za pomoci útržků kódu z verze 2. Tato verze poté volně přešla v úplnou implementaci, která bude dále podrobně rozepsána v *kapitole 5 Realizace*.

4.4 Vyhodnocení a návrh ideální cesty

Jako nejlepší cesta byla zvolena Verze 3, která se inspiruje strukturou Backbone.js, ale implementuje ji vlastním způsobem. Tato varianta se ukázala být nejrychlejší a nejjednodušší co se týče složitosti a přehlednosti kódu. Zároveň také díky objektově orientovanému návrhu je možné jednoduše rozšiřovat funkčnost aplikace.

V žádné z verzí nebylo nijak využito frameworku Nette. Byl pro aplikaci příliš obsáhlý a nepřinášel nijak zvláštní užitek. Tato práce má primárně demonstrovat trochu odlišný přístup k vytváření webových stránek, proto by bylo nadbytečné ji stavět nad tímto frameworkem, který se zaměřuje na opravdu velké množství věcí.

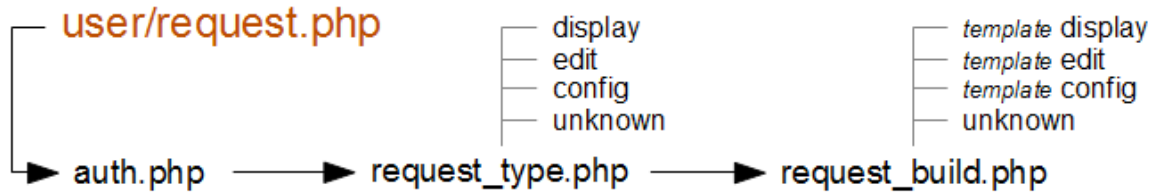
5 Realizace

Jak již bylo řečeno samotná implementace kompletní aplikace vychází z verze 3 (viz. *Příprava*). Bylo nutné sesbírat všechny úryvky aplikace a sestavit je dohromady v jeden funkční celek a také dodělat veškeré detaily, které při přípravě implementovány nebyly. Tato kapitola popisuje hotovou funkční aplikaci, vytvořenou při tomto procesu. Nebudou popisovány přímo jednotlivé soubory zdrojového kódu (ty jsou zakomentovány a popsány), ale spíše se zaměřím na to, jak aplikace pracuje jako celek.

Aplikace se dá pomyslně rozdělit na několik částí, které navzájem spolupracují. Každá tato část funguje samostatně a pro komunikaci s okolím využívá aplikačního komunikačního protokolu. Taková nezávislost umožňuje vnášení úprav bez nutnosti úpravy celé aplikace.

5.1 Serverová část

Tímto jsou myšleny všechny skripty, které jsou spouštěny a prováděny na straně serveru, nikoliv u klienta. Dále pod tento pojem patří také databázový server (v tomto případě MySQL) se všemi systémovými daty a případně i obrázky a další neskriptové soubory uložené přímo na serveru.



Obr. 12: Vyhodnocení přímého uživatelského požadavku

Výše uvedený obrázek zobrazuje postup skriptu, který vyhodnocuje přímý uživatelský požadavek, což je primárním úkolem této části aplikace. Přímým uživatelský požadavkem se rozumí pokus uživatele o načtení konkrétní stránky (URL) ze serveru.

Průběh vyhodnocování:

1. autentizace a autorizace uživatele
   Pokud je autentizace a autorizace povolena, server si postupně ověří stav uživatele. Vyhodnocuje se několik faktorů – uživatel už je/není přihlášen, obdržený požadavek vyžaduje/nevyžaduje autorizaci, uživateli již byl odepřen/povolen přístup na stránku. Po tomto vyhodnocení je provedena adekvátní akce – přesměrování na přihlášení, provedení požadavku, odepření přístupu.

2. vyhodnocení typu požadavku
   Přímé uživatelské požadavky jsou rozděleny na tři základní typy, doplněné o „neznámý“ požadavek.

   Typy uživatelský požadavků:
   - display – žádost o zobrazení stránky a vykreslení widgetů
   - edit – žádost o editaci stránky a widgetů
   - config – nastavení aplikace a správa widgetů

   Požadavek je na straně serveru rozlišen pomocí parametrů v GET proměnné. Pokud nejsou zadány žádné parametry, je automaticky proveden požadavek *display*.

3. sestavení odpovědi
   Po ukončení předchozího kroku známe uživatelský požadavek a nyní sestavíme a odešleme odpověď.

   Pro každý typ uživatelského požadavku, které jsou popsány výše, existuje šablona. Ta bude použita pro sestavení odpovědi.

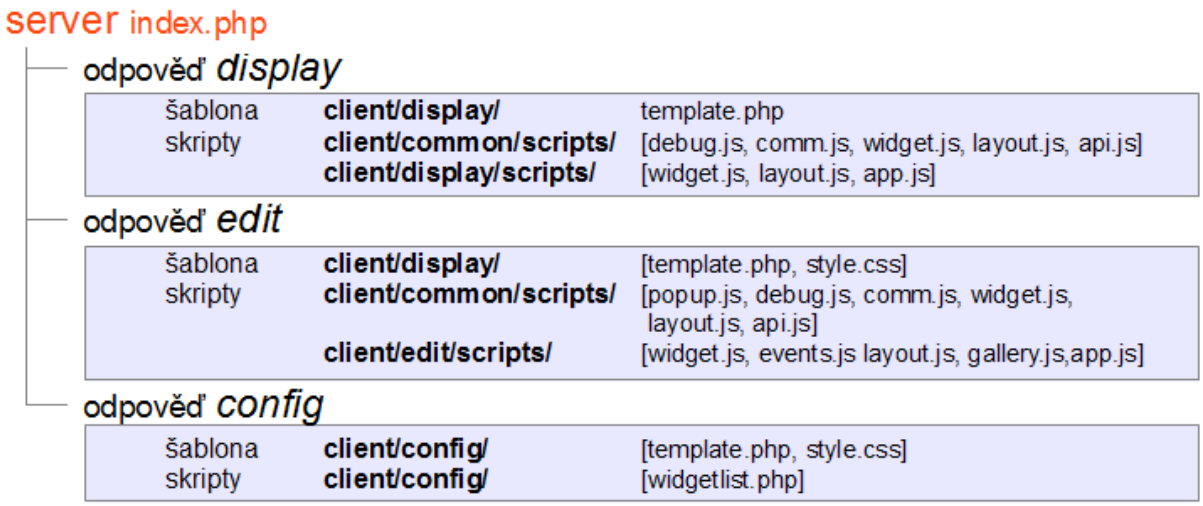
   Šablony:
   - požadavek *display* – šablona *client/display/template.php*
   - požadavek *edit* – šablona *client/edit/template.php*
   - požadavek *config* – šablona *client/config/template.php*

   Šablona je načtena a část aplikace pro klienta je připravena k odeslání.

4. ukončení činnosti skriptu na straně serveru
   Po tom, co klient obdrží aplikaci, už neprobíhá žádná komunikace ze serverovou částí až do okamžiku dalšího přímého klientského požadavku. Veškerá komunikace je už pouze asynchronní. Činnost této části aplikace je tedy dokončena.

5.2 Klientská část

Jedná se o veškerou činnost, které se vyhodnocuje pouze u klienta:
- zobrazení a vykreslení HTML
- CSS styly
- získání a zobrazení obrázků a dalších dodatečných souborů
- provedení Javascriptu
- odesílání a přijímání asynchronní komunikace



Obr. 13: Souborová struktura klientské části

Tento obrázek ukazuje soubory, které jsou odesílány klientovy na základě jednotlivých požadavků. Jedná se pouze o základní aplikaci, to znamená, že po spuštění mohou být stahovány ještě další obrázky, zdroje, skripty, soubory widgetů a další.

5.2.1 Obecné skripty

Definují kostry objektů, funkce a další informace, které jsou společné pro více částí aplikace. Jedná se tedy o jakýsi základ, na kterém se bude dále stavět. Cesta ke kořenové složce této části je *client/common/*.

Obsažené soubory:

- *./login.php* – formulář pro uživatelské přihlášení
- *./scripts/api.js* – základní rozhraní pro ovládání aplikace a widgetů
- *./scripts/comm.js* – objekt zjednodušující asynchronní komunikaci se serverem
- *./scirpts/debug.js* – jednoduché prostředí pro správu debugovacích výstupů aplikace
- *./scripts/popup.js* – funkce pro zobrazení/schování okna vykresleného přes stránku s libovolným obsahem
- *./scripts/layout.js* – základní implementace objektu, starajícího se o zobrazení stránky a správu widgetů
- *./scripts/widget.js* – základní implementace objektu instance widgetu, zajišťující jeho načtení, zobrazení a činnost

Obecné skripty samy o sobě nemají žádnou funkčnost a nijak se nespouští. Jsou ale využity pro další části aplikace, které na nich stavějí. Tato návaznost bude popsána dále.

5.2.2 Skripty pro zobrazení stránky

Slouží pro obsluhu přímého uživatelského požadavku „display.“ Cesta ke kořenové složce této části je *client/display/*.

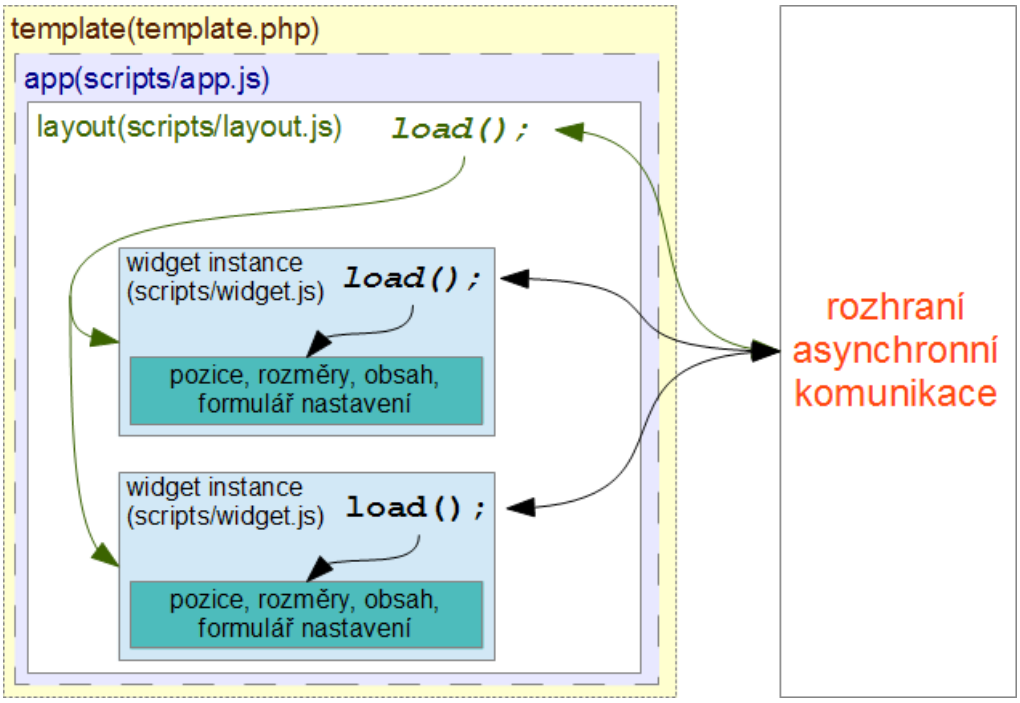
Obsažené soubory:

- *./template.php* – šablona pro vykreslení stránky; připojí všechny skriptové soubory a vytvoří základní HTML kostru, se kterou pak aplikace bude pracovat
- *./scripts/app.js* – kořenový skript; inicializuje veškerou činnost
- *./scripts/layout.js* – objekt pro zobrazení stránky; vychází z obecné implementace *common/scripts/layout.js*
- *./scripts/widget.js* – objekt instance widgetu; vychází z obecné implementace *common/scripts/widget.js*

Výše uvedené skripty pro svou činnost dále využívají *common/scripts/debug.js* a *common/scripts/comm.js*.

Průběh činnosti:

1. inicializaci provádí skript *app.js;* vytvoří globální proměnnou *app*, ve které bude umístěná celá aplikace
2. vytvoření nového objektu *layout*
3. objekt *layout* získá informace o uloženém rozložení stránky ze serveru v podobě seznamu instancí widgetů
4. pro každou instanci widgetu je vytvořen nový objekt *widget* a je uložen do pole v objektu *layout*
5. každý objekt *widget* zná ID instance widgetu, již má na stránce vykreslovat a podle tohoto ID si získá ze serveru všechny informace pro to, aby tak mohl učinit
6. jakmile všechny objekty mají svoje data stáhnuté ze serveru, vykreslí se do HTML kostry ze souboru *template.php*
7. ze strany aplikace je činnost dokončena a jednotlivé instance widgetů mohou zahájit svou vlastní činnost



Obr. 14: Schéma klientské aplikace(požadavek display)

Na *obr. 14* je vidět, jakým způsobem probíhá zobrazení stránky a jaké mají závislosti jednotlivé objekty. Na *rozhraní asynchronní komunikace* je přistupováno pomocí objektu *comm* (definovaného v souboru *client/common/scripts/comm.js*), který je obsažen v každé části využívající asynchronní komunikaci, na obrázku ve všech metodách *load()*. Tento objekt není ve schématu na *obr. 13* znázorněn.

5.2.3 Skripty pro editaci rozvržení stránky

Skripty obsahující přímý uživatelský požadavek „*edit.*“ Kořenová složka je *client/edit/*. Implementace této části volně vychází ze skriptů pro obsluhu požadavku *display*. Vykreslení stránky a widgetů probíhá stejně. Dále jsou zde skripty umožňující nastavení instancí widgetů, skripty realizující galerii dostupných widgetů a jednoduché menu.

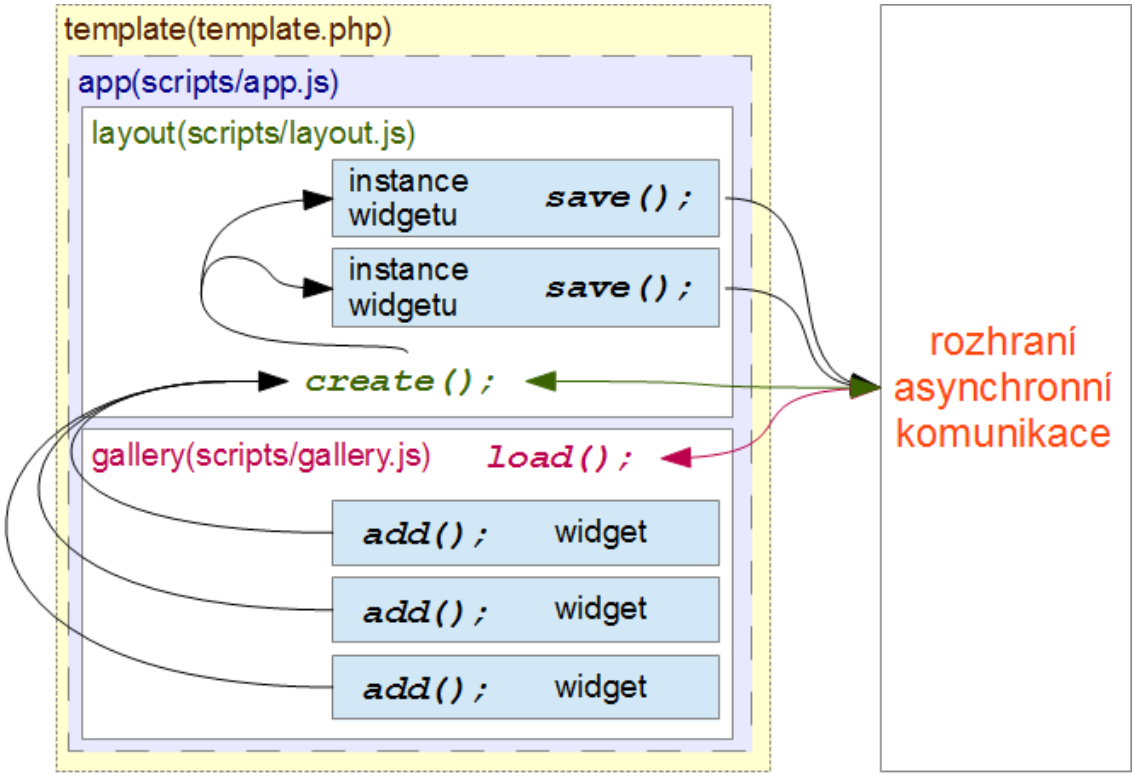
Obsažené soubory:

- *./template.php* – šablona pro vykreslení editační stránky, galerie modulů a menu; připojí všechny skriptové soubory, soubory CSS a vytvoří základní HTML kostru, se kterou pak aplikace bude pracovat
- *./style.css* – soubor CSS upravuje vzhled menu, galerie widgetů
- *./app.js* – kořenový skript; inicializuje veškerou činnost (důležitá část je vytvoření objektu *layout* – vykreslení editační stránky – a objektu *gallery* – seznam dostupných nainstalovaných widgetů)
- *./events.js* – připojuje funkce na DOM události tlačítek v menu
- *./gallery.js* – objekt starající se o načtení a zobrazení seznamu dostupných widgetů a umožňuje jejich přidání do stránky
- *./layout.js* – objekt pro zobrazení stránky; vychází z obecné implementace *common/scripts/layout.js;* obsahuje navíc metody pro vytvoření nové instance widgetu a uložení rozložení stránky do databáze
- *./widget.js* – objekt instance widgetu; vychází z obecné implementace *common/scripts/widget.js;* obsahuje navíc metody pro uložení nového nastavení do databáze,

odstranění instance a taky povoluje manipulaci – změna velikosti, změna pozice, změna zIndexu

Průběh činnosti:

1. inicializaci provádí skript *app.js;* vytvoří globální proměnnou *app*, ve které bude umístěná celá aplikace
2. vytvoření nového objektu *layout* a objektu *gallery*
3. funkce pro ovládání menu jsou připojeny na jednotlivé tlačítka pomocí událostí DOM
4. objekt *layout* získá informace o uloženém rozložení stránky ze serveru v podobě seznamu instancí widgetů
5. objekt *gallery* získá informace o dostupných widgetech ze serveru
6. pro každou instanci widgetu je vytvořen nový objekt *widget* a je uložen do pole v objektu *layout*
7. každý objekt *widget* zná ID instance widgetu, již má na stránce vykreslovat a podle tohoto ID si získá ze serveru všechny informace, pro to aby tak mohl učinit
8. jakmile mají všechny objekty svoje data stáhnuté ze serveru, vykreslí se do HTML kostry ze souboru *template.php*
9. po dokončení načítání a zobrazení je možná stránku začít editovat



Obr. 15: Schéma klientské aplikace(požadavek edit)

Obr. 15 znázorňuje strukturu aplikace při editaci stránky. Objekt *layout* má stejné vlastnosti jako ten, který je v *obr. 14* (ty zde nejsou pro zjednodušení uvedeny) a doplňuje je o metodu *create()*, tedy vytvoření nové instance widgetu. Tato metoda je vyvolána objektem *gallery* v případě, že uživatel provede akci přidání nového modulu na stránku (událost kliknutí na tlačíko, která vyvolá metodu *add();*). Tím vzniká nová instance, které později, pokud je uloženo celé rozvržení, je zaslána i do databáze a vytvořena permanentně.

5.2.4 Konfigurace aplikace

Na této stránce je možné zejména spravovat widgety. Každý widget, který byl na server nahrán, je nutné nejprve nainstalovat, aby ho bylo možné používat. Dostáváme seznam widgetů, jejich stav a možnost akce (nainstalovat nebo odebrat). Odebrání widgetů způsobí ztrátu veškerých dat, jež byla v průběhu jeho činnosti uložena do databáze.

5.2.5 Dědičnost v Javascriptu

V JavaScriptu oficiálně dědičnost není, nicméně v této aplikaci je dědičnost simulována. Tento krok nebyl nutný, ale pomáhá vytvářet velice přehlednou a logickou strukturu v kódu.

Konkrétně byla využita při vytváření objektů *layout (client/common/scripts/layout.js)* a *widget (client/common/scripts/widget.js).* Při obsluhování *display* i *edit* mají tyto objekty mnoho společného a liší se jen v několika metodách. Bylo by tedy zbytečné opisovat jejich společný kód kvůli drobným rozdílům a raději byla simulována dědičnost, která umožňuje jejich společné části sdílet a jednotlivé rozdíly vnášet až ve dodatečné specializaci těchto objektů.

```
1  // Layout object template
2  def.common.Layout = function(spec){
3        // 'this' variable for nested objects and functions
4        var self = this;
5        // object methods
6        this.load = function(){
7              ...
8        };
9        // return object
10       return this;
11 }
12 // Layout edit implementation
13 def.edit.Layout = function(){
14       // common.Layout() template
15       var self = new def.common.Layout(def.edit);
16       // object methods
17       self.create = function(widget_id,pos_x,pos_y){
18             ...
19       };
20       // return new object
21       return self;
22 }
```

Obr. 16: Příklad řešení dědičnosti v Javascriptu

Tento zdrojový kód demonstruje, jakým způsobem je dědičnost simulována.

Máme definovaný konstruktor *def.common.Layout(spec)*, který představuje obecnou implementaci objektu *layout.* Konstruktor pracuje s hodnotou *this* a tu po dokončení činnosti taky vrací, aby bylo možné použít operátor **new**. Definice obsahuje metodu *this.load()* a proměnnou *self.*

Dále máme konstruktor *def.edit.Layout().* Jedná se o rozšíření předchozího objektu *def.common.Layout.* Ve zdrojovém kódu vidíme na řádku 15 vytvoření nového obecného objektu *def.common.Layout()* a uložení do proměnné *self.* Konstruktoru je předán jako parametr odkaz na prostředí, se kterým má objekt *layout* pracovat (využívá další objekty definované v tomto prostředí,

jedná se tedy o jednoduchou šablonu, jakou známe například v C++). Konstruktor dále definuje metodu *self.create(...)*. Tato metoda v obecném konstruktoru nebyla. Po přidání všech rozdílů je vrácena proměnná *self,* nikoliv *this*. Dostáváme tedy objekt, obsahující všechny metody a proměnné definované konstruktorem *def.common.Layout(spec)* upravené a doplněné o metody a proměnné z konstruktoru *def.edit.Layout()*.

5.3 Asynchronní komunikace

Velká část dat aplikace je přenášena asynchronně. Děje se tak z důvodu, že se předem nedá určit, kdy budou potřeba a také tento přístup umožňuje vytvořit moderní stránku, kdy se obsah mění bez toho, aniž by bylo nutné načítat celou novou stránku. V této kapitole bude popsáno, jak probíhá asynchronní komunikace mezi serverem a klientem.

5.3.1 Rozhraní asynchronní komunikace

Aplikace implementuje jedno rozhraní pro příjem a odesílání zpráv asynchronní komunikace implementované souborem *comm.php*. Toto rozhraní je založené na návrhovém vzoru front-controller, tedy veškeré požadavky jsou směrovány pouze na tento jeden soubor a dále jsou větveny již podle samotného obsahu. Komunikace probíhá pouze ve formátu XML. Existuje několik vyjímek u zasílání odpovědí, které formát XML nedodržují. Požadavky, které ve formátu XML nebudou nebo nedodrží obecnou strukturu požadavku, budou zahozeny.

V souboru *comm.php* dochází ke zpracování a vyhodnocení příchozího XML požadavku, rozlišení na jednotlivé typy (jež jsou popsány v kapitole 5.3.3) a předání řízení adekvátním skriptům nebo obsluha jednoduchých požadavků včetně odeslání odpovědi.

5.3.2 Obecná struktura požadavku/odpovědi

Požadavek

Většina požadavků je ve formátu XML(od klientské části aplikace na server). Každý takovýto požadavek má základní strukturu, kterou musí dodržet:

```
1 <?xml version="1.0"?>
2 <request>
3       <data type="[request_type]" action="[request_action]">
4             ...
5       </data>
6 </request>
```

Obr. 17: Obecná struktura asynchronního požadavku

Na obrázku je vidět, jak požadavek vypadá. Kořenový se musí jmenovat „*request*“ a tag „*data*“ musí obalovat konkrétní data požadavku. Tag !*data*“ dále obsahuje parametry „*type*“, označující typ požadavku a „*action*“, označující konkrétní akci, která se bude vykonávat.

Odpověď

Struktura odpovědi nemá formu nijak definovanou, protože její obsah je doručen vždy přímo žadateli a není proto potřeba vyhodnocovat, kam patří a tedy nemusí být ani uveden typ nebo o jakou odpověď na který dotaz se jedná.

```xml
<?xml version="1.0"?>
<response>
	<data>
		...
	</data>
</response>
```

Obr. 18: Obecná struktura asynchronní odpovědi

Odpovědi zaslané vlastní aplikací (nikoliv widgety) mají vždy alespoň základní strukturu, která je uvedena na *obrázku 14*. Obsahuje kořenový tag <*response*> a poté samotná data odpovědi obalené tagem <*data*>.

Odpovědi zasílané widgety mohou tedy mít naprosto libovolnou strukturu a nijakým způsobem to neovlivní funkčnost aplikace. Není tedy ani nutné, aby byla odpověď ve formátu XML.

5.3.3 Rozdělení požadavků

Požadavky od klientské aplikace se dělí na dva základní typy, které dále obsahují detailnější informace. Jedná se o typy požadavků *instance* a *service*. Obecně, požadavek *instance* představuje obsluhu samotné aplikace a požadavek *service* patří widgetům.

Žádné další typy požadavků nejsou možné a aplikace je bude ignorovat. Widgety tedy musí využít pouze typ *service,* jinak nemohou využít rozhraní *comm.php*, což není doporučeno. Implementace vlastního rozhraní naruší strukturu celé aplikace a bude zbytečně vnášet zmatek do kódu.

5.3.4 Požadavky typu *instance*

Všechny požadavky tohoto typu zajišťují funkčnost samotné aplikace až do momentu, kdy je předáno řízení jednotlivým instancím widgetů.

Požadavky *instance* se dále dělí na 4 akce *CRUD (create, read, update, delete)* [19], doplněné o akce *list* (výpis všech uložených instancí widgetů) a *gallery* (výpis všech dostupných nainstalovaných widgetů).

CRUD akce se týkají výhradně obsluhy instancí widgetů (objekt *widget*), akce *list* využívá objekt rozvržení stránky *layout* a akce *gallery* patří objektu, obsluhujícímu seznam dostupných widgetů – objekt *gallery.*

```xml
<?xml version="1.0"?>
<request>
	<data type="instance" action="create">
		<instance>
			<widget_id value="1" />
			<pos_x value="787" /><pos_y value="141" />
			<size_x value="390" /><size_y value="218" />
		</instance>
	</data>
</request>
```

Obr. 19: Příklad asynchronního požadavku: Vytvoření instance widgetu

Tento XML dokument je ukázka toho, jak může vypadat požadavek pro vytvoření nové instance widgetu se zadanou velikostí a pozicí.

5.3.5 Požadavky typu *service*

Požadavky tohoto typu se týkají přímo samotného widgetu. Nějakým způsobem zajišťují jeho funkčnost. Pod tímto typem by měla být obsažena veškerá komunikace, která musí proběhnout mezi serverem a instancí widgetu. Widget by tedy neměl vytvářet vlastní architekturu klient-server, ale měl by využít tohoto rozhraní.

Ačkoliv všechny tyto požadavky by měly patřit obsluze mimo vlastní aplikaci, je zde několik výjimek. Akce, prováděné vlastní aplikací, patřící do typu *service*:

- požadavek *display* – aplikace posílá požadavek *display* jako typ *service*
- požadavek *display-config* – požadavek na zobrazení formuláře pro konfiguraci instance widgetu; opět zasílán vlastní aplikací; jde o nepovinný požadavek a jeho obsluha nemusí být implementována

Tyto dvě akce jsou tedy rezervované aplikací a nesmí být použity jako název akce požadavku typu *service.*

I přes volnost, kterou zde widgety dostávají při své komunikaci ze serverem, je nutné zachovat aspoň základní strukturu požadavku (viz. *Obecná struktura požadavku/odpovědi*) a taky uvést ID instance widgetu, které tuto komunikaci provádí.

```
1 <?xml version="1.0"?>
2 <request>
3       <data type="service" action="post">
4             <instance_id value="8" />
5             <timestamp value="1365963176136" />
6             <nickname><![CDATA[Jiří]]></nickname>
7             <message><![CDATA[Ahoj. Jak se máte?]]></message>
8       </data>
9 </request>
```

Obr. 20: Příklad asynchronního požadavku: Zaslání zprávy do chatu

Všechny ostatní akce budou předány přímo objektu instance widgetu a jeho metodě *service($xml),* jak je popsáno v *Obecná struktura widgetu.*

Toto je ukázka jednoho požadavku typu *service* od instance widgetu *Simple Chat.* Jedná se o zaslání nové zprávy do chatu. Požadavek vyvolává akci *post,* která je dále implementována právě ve funkci *service($xml).* V příkladu vidíme tedy základní strukturu požadavku včetně ID instance widgetu a poté samotná data, které instance widgetu na server zasílá.

5.4 Demonstrační widgety

Jako příklad toho, jak může být tento framework využit (a jak by měl být využit) bylo vytvořeno několik widgetů, které předvádějí možnosti a taky slouží jako návod pro tvorbu vlastních nových widgetů. Každý widget taky musí dodržovat určitý minimální formát, aby s ním mohla aplikace úspěšně komunikovat.

5.4.1 Obecná struktura widgetu

Všechny widgety musí být umístěny ve složce *modules/nazev_widgetu,* kde *nazev_widgetu* představuje libovolné unikátní kódové označení widgetu.

Následující body představují minimální požadavek na widget proto, aby vše správně fungovalo:

a) instalace a odinstalace

Každý widget musí obsahovat soubor *install.php* ve svojí složce. Tento soubor obsahuje informace potřebné k instalaci nebo odinstalaci widgetu a funkce, které zajistí vytvoření nebo smazání databázové struktury pro tento widget.

b) implementace rozhraní

Widget musí implementovat abstraktní třídu *widget_template* definovanou v souboru *modules/widget.php*. Tato třída vynucuje implementaci 4 metod:

- *static function create()* - vytvoří novou instanci widgetu
- *function remove()* - odstraní danou instanci widgetu se všemi uloženými daty
- *function display()* - zobrazí instanci widgetu(její výstup bude obsahem pole na stránce)
- *function service($xml)* – vnitřní komunikace widgetu; tato metoda je nepovinná (ovšem musí být definována)

Více detailněji je toto téma rozepsáno v příloze *Manuál pro tvorbu widgetů.*

5.4.2 Widget - Statické HTML

Tento widget umožňuje vkládat do stránky útržky HTML kódu, včetně skriptů, flashových objektů a dalších. Jedná se o pouze o útržek, to znamená, že v kódu nesmí být HTML hlavička, začátek a konec HTML souboru (<*html*>), tag pro titulek okna a další. Lze tedy použít vše, co lze použít ve standardním HTML souboru mezi tagy <*body*> a <*/body*>.

HTML kód se vkládá při editaci stránky, kde se po dvojkliku na instanci widgetu objeví okno s nastavením. V okně bude kolonka, do které se umísťuje požadovaný HTML kód a tlačítko pro uložení „Save.“ Změna obsahu se projeví až po aktualizaci celé stránky.

5.4.3 Widget – Statický obrázek

Umožňuje na stránce zobrazit jeden obrázek, který je určený zadaným URL. Widget sám o sobě neumožňuje nahrávání souboru na server. Je tedy nutné mít předem umístěný soubor obrázku někde na internetu a zadat jeho platné URL. Adresu souboru obrázku je možné upravit po dvojkliku na instanci widgetu.

Tento obrázek se pak zobrazí tak, aby vyplňoval co nejlépe pole instance widgetu (tzn. přizpůsobuje se šířce a výšce pole). Na *obr. 21* vidíme příklad tohoto přizpůsobení. V levé části je pole vyšší než obrázek, a proto byla upravena šířka obrázku. V pravé části je naopak pole širší než obrázek a byla upravena jeho výška. Vždy je tedy zachován původní poměr stran obrázku.

Obr. 21: Přizpůsobování obrázku poli

5.4.4 Widget – Jednoduchý chat

Chatovací okno, ve kterém uživatel vyplní svou přezdívku (bez nutnosti registrace a přihlašování), poté zadá zprávu a odešle. Zprávy se automaticky načítají ze serveru v intervalech 2,5 sec. Chat se automaticky přizpůsobuje velikosti pole, ve kterém je vykreslený (dle rozložení stránky).

5.5 Uživatelské zkušenosti

Výsledná aplikace měla představovat nový přístup k vytváření webových stránek. Bylo proto vhodné získat uživatelské hodnocení tohoto nástroje. Hodnocení bylo zaměřeno na dojem z uživatelského rozhraní samotného, nikoliv na technické zpracování a kvalitu implementace. Jak bylo zmíněno již v úvodu této práce, uživatel může mít různé zkušenosti s tvorbou webových stránek, což je nutné rozlišit při hodnocení zpětné vazby. Podle úrovně znalostí jsem uživatelské dojmy z vyzkoušení aplikace rozdělil do tří skupin: základní nebo žádné znalosti, mírně pokročilé znalosti, rozsáhlé nebo profesionální znalosti.

V následujících odstavcích jsem shrnul dojmy z použití aplikace podle úrovně znalostí uživatelů, kteří si přímo vyzkoušeli používání aplikace za účelem vytvoření jednoduchého obsahu na stránce. Jednalo se o několik nápisů nebo odstavců textu, dva nebo tři obrázky o různé velikosti a taky vyzkoušení widgetu *Jednoduchý chat*. Takovou stránku měli sestavit samostatně bez žádného předchozího obsahu, který by se dal upravovat.

Uživatelé se základními nebo žádnými znalostmi

Před vyzkoušením aplikace jim byl poskytnut manuál popisující ovládání aplikace. Bez něj bylo použití aplikace složité(nepochopení účelu widgetů; problém se vkládáním widgetů do stránky a s jejich nastavením). Po přečtení manuálu byly tito uživatelé schopni, pokud ne ihned, tak po pár pokusech začít přidávat widgety do stránky z galerie widgetů. Přesouvání jednotlivých instancí widgetů a změna jejich rozměrů nedělala problém nikomu. Problém se vyskytl u ukládání individuálních nastavení instancí widgetů, kde se změny ukládají tlačítkem *Save* ve vyskakovacím okně, nikoliv tlačítkem *Save* pro uložení rozvržení stránky. Někomu dělalo taky problém zavření vyskakovacího okna(zavření galerie, zrušení nastavování instance widgetu), kde je potřeba kliknout kdekoliv do zašedivělého prostoru mimo vyskakovací okno.

Mírně pokročilí uživatelé

Zde se ukázalo, že jsou schopní používat aplikaci i bez přečtení manuálu a opět maximálně na několik pokusů byli schopni provádět veškeré editační operace bez problémů. Téměř každý se, při vytváření nové instance widgetu, snažil zadat u pozice zrovna i velikost požadovaného prvku(jako při kreslení obdélníku v obrázkovém editoru). Uživatelé byly schopní do widgetu *Statické HTML* zadávat formátovací HTML značky a nezůstali pouze u textu(nikdo nezačal psát značky jako <html>,<body>

nebo <head>, které v tomto widgetu nesmějí být použity). Bylo taky vytýkáno nemožnost přímé změny pozadí celé stránky v nastavení aplikace, kde to očekávali.

Uživatelé s rozsáhlými nebo profesionálními znalostmi
Nevyskytl se žádný problém v ovládání aplikace, který by nedokázali po krátké chvíli zkoušení vyřešit a rychle sami bez slovního úvodu nebo manuálu pochopili jaký má aplikace účel. Dokázali i efektivně použít nastavení zIndexu u jednotlivých instancí widgetu a způsobit tak zajímavé překrývaní prvků.

Všem uživatelům se líbila možnost naprosto volného rozmisťování widgetů po stránce. Nikdo z nich, ale neprojevil nadměrné nadšení z konceptu práce, kdy někteří souhlasili s tím, že to bylo způsobeno nedostatečným množstvím dostupných widgetů, se kterýma by bylo možné pracovat. Možnost tvorby vlastních widgetů byla u některých mírně pokročilých uživatelů a u většiny velmi pokročilých vítána jako snadná možnost rozšiřitelnosti aplikace. Z hodnocení vyplynulo několik nedostatků v uživatelském rozhraní. Je tedy pořád prostor pro jeho zdokonalování. Většina uživatelů taky neměla zkušenosti s podobnou aplikací(pokročilý uživatelé znali iGoogle), a byl pro ně tento nástroj novým přístupem.

5.6 Instalace a nastavení aplikace

Nově nahraná aplikace na server ještě nemůže být funkční. Je potřeba nejprve nastavit údaje o připojení k databázi a nahrát základní databázovou strukturu.

Soubor *config.php* obsahuje všechny potřebné informace, které je nutné správně vyplnit před tím, než je zahájena instalace aplikace.

Souboru *install.php* provede všechny kroky, nutné pro správný běh aplikace.

Po instalaci je vhodné soubor *install.php* odstranit, přemístit nebo jakkoliv zamezit jeho znovuspuštění ať už administrátorem nebo uživatelem. Mohlo by dojít k poškození databáze nebo ztrátě dat.

Aby se dala aplikace využít, je ještě nutné nahrát požadované widgety a nainstalovat je. Jinak by nebylo co na webovou stránku umísťovat.

Prostředí pro běh aplikace vyžaduje webový server, na kterém je nainstalováno rozšíření PHP verze 5.4.8 a vyšší s knihovnou *mysqli*. Dále je potřeba přístupný MySQL databázový server verze 5.5 a vyšší, kde existuje přihlašovací účet, který má povolené všechny operace minimálně v rámci databáze určené pro tuto aplikaci.

6 Závěr

Cílem této práce bylo vytvořit webovou aplikaci, která bude usnadňovat tvorbu vlastních webových stránek. Práce je inspirována již existujícími řešeními, které fungují podobným způsobem, ale snaží se přistupovat k problému tvorby stránek vlastním způsobem. Existuje velké množství systémů a šablon pro ně využitelných pro nejrůznější požadavky. Tato aplikace se ale snaží o vypuštění využití obrovského množství šablon.

Uživatel dostává do rukou nástroj, který chápe webovou stránku jako soubor mnoha menších widgetů, kde každý takovýto widget zobrazuje jiný obsah a provádí jinou činnost, rozmístitelných libovolně po stránce jednoduchým vložením, přizpůsobením velikosti a přetažením na požadovanou pozici. Jednotlivé widgety jsou přístupné pomocí galerie, kde je zobrazen přehled těch dostupných. Aby aplikace co nejméně omezovala fantazii a potřeby uživatelů, je možné vytvářet vlastní widgety, jejichž činnost je jenom minimálně omezena ze strany aplikace.

Vytváření vlastních widgetů a také jejich následné zveřejňování a sdílení s dalšími uživateli, je základní myšlenka, která stojí za vznikem této aplikace. Pokud někdo nemá zkušenosti s tvorbou webových stránek, může jednoduše využít již hotové widgety od jiných uživatelů a vytvořit tak vlastní web bez nutnosti studovat technologie, ale zároveň není omezen šablonami.

Aplikace odpovídá definici *Web 2.0*. Máme jednu webovou stránku, na níž se obsah aktualizuje bez načtení celého webu znovu a je zde mnoho interaktivních prvků. Bylo tedy zapotřebí nastudovat mnoho technologií, které se dnes využívají při vývoji méně či více, zjistit na co jsou vhodné a které nástroje by realizaci spíše zkomplikovali. Po teoretické přípravě byly vybrány technologie *HTML, CSS, PHP, MySQL, Javascript, Ajax, jQuery, jQuery-UI, Backbone.js* a *XML*(jako datový nosič), ze kterých byla nakonec vypuštěna knihovna *Backbone.js*, ale její struktura byla použita pro návrh aplikace.

Podobná řešení, zabývající se podobným problémem, posloužila jako velmi dobrá inspirace. Stejně tak se tato aplikace snaží vyvarovat některých nedostatků, které se v těchto existujících nástrojích vyskytují. Hlavní inspirace vychází z aplikace iGoogle, ze které bylo čerpáno mnoho vlastností, následně přizpůsobených cílům této práce.

Backbone.js svým vlastním způsobem implementuje návrhový vzor *MVC*, který byl použit i v této práci společně s *Front controller* návrhovým vzorem. Z těchto základů pak vznikl návrh rozdělený podle modelu *klient-server* na tři hlavní části: *sestavení aplikace* (server), *zobrazení a běh aplikace* (klient), *asynchronní komunikace* (server), mezi kterými probíhá komunikace podle aplikačního protokolu za cílem vyhovění uživatelským požadavkům.

V průběhu realizace vznikly celkem tři neúplné verze aplikace, založené na odlišných kombinacích technologií. Tímto bylo zjištěno, které řešení bude optimální a to bylo poté implementováno do plné funkčnosti. Pro tuto práci to byla verze založená na struktuře *Backbone.js,* ale tato knihovna přímo použita nebyla. Byla vypuštěna kvůli zbytečné komplexnosti a absenci některých funkcí, které by musely být implementovány navíc mimo knihovnu (šlo o funkce synchronizující data u klienta s těmi na serveru).

Ve výsledné aplikaci jsou obsaženy demostrační widgety, na kterých lze vyzkoušet, jak tento nástroj funguje: *Statické HTML* (vkládání útržků HTML do stránky), *Statický obrázek* (zobrazení jednoho obrázku na stránce), *Jednoduchý chat* (umožňuje zasílání zpráv s přezdívkou všem připojeným uživatelům).

Tuto aplikaci si vyzkoušelo několik uživatelů, kteří se lišili svými znalostmi z oblasti tvorby webových stránek. Těm neznalým byl poskytnut manuál ovládání aplikace, po jehož přečtení téměř všichni dokázali použít všechny funkce poskytované aplikací, kromě možnosti vytvoření vlastních widgetů. Aplikace byla hodnocena kladně hlavně kvůli možnosti naprosto volného umístění widgetu na stránce, uložení a opětné editaci tohoto rozmístění.

Literatura

[1] Backbone.js. ASHKENAS, Jeremy. DOCUMENTCLOUD INC. *Backbone.js* [online]. 2010-2013 [cit. 2013-05-12]. Dostupné z: http://backbonejs.org/

[2] Underscore.js. ASHKENAS, Jeremy. DOCUMENTCLOUD INC. *Underscore.js* [online]. 2009-2013 [cit. 2013-05-12]. Dostupné z: http://underscorejs.org/

[3] BRÁZA, Jiří. *PHP 5: Začínáme programovat*. 1. vyd. Překlad Ondřej Baše, Ondřej Žižka. Praha: Grada Publishing, 2005, 244 s. ISBN 80-247-1146-X

[4] MySQL: MySQL 5.5 Reference Manual. ORACLE. *MySQL 5.5 Reference Manual* [online]. 1997-2013 [cit. 2013-05-12]. Dostupné z: http://dev.mysql.com/doc/refman/5.5/en/

[5] TodoMVC: Helping you select an MV* framework. OSMANI, Addy, Sindre SORHUS a Pascal HARTIG. [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://todomvc.com/

[6] PÍSEK, Slavoj. *JavaScript: efektní nástroj oživení www stránek*. 1. vyd. Praha: Grada, 2001, 231 s. ISBN 80-247-0014-X.

[7] RESIG, John. *JavaScript a Ajax: moderní programování webových aplikací*. Vyd. 1. Překlad Ondřej Baše, Ondřej Žižka. Brno: Computer Press, 2007, 360 s. ISBN 978-80-251-1824-5

[8] ŠKULTÉTY, Rastislav. COMPUTER PRESS. *JavaScript: programujeme internetové aplikace*. Vyd. 1. Praha: Computer Press, 2004, 224 s. ISBN 80-251-0144-4

[9] jQuery: Write less, do more. THE JQUERY FOUNDATION. j*Query* [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://jquery.com/

[10] jQuery: User interface. THE JQUERY FOUNDATION. j*Query* [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://jqueryui.com/.

[11] PHP: Hypertext Preprocessor. THE PHP GROUP. [online]. [cit. 2013-05-12]. Dostupné z: http://php.net/

[12] W3C. *World Wide Web Consortium* [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://www.w3.org/

[13] W3Schools Online Web Tutorial. REFSNES DATA. *W3Schools Online Web Tutorial* [online]. 1999-2013 [cit. 2013-05-12]. Dostupné z: http://www.w3schools.com/

[14] Wikipedia, the free encyclopedia. WIKIMEDIA FOUNDATION, Inc. *Representational state transfer* [online]. 2013 [cit. 2013-05-12]. Dostupné z: https://en.wikipedia.org/wiki/Representational_state_transfer

[15] Wikipedia, the free encyclopedia. WIKIMEDIA FOUNDATION, Inc. *Model–view–controller* [online]. 2013 [cit. 2013-05-12]. Dostupné z: https://en.wikipedia.org/wiki/Model–view–controller.

[16] Wikipedia, the free encyclopedia. WIKIMEDIA FOUNDATION, Inc. *Model–view–presenter* [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://en.wikipedia.org/wiki/Model–view–presenter

[17] Wikipedia, the free encyclopedia. WIKIMEDIA FOUNDATION, Inc. *Client–server model* [online]. 2013 [cit. 2013-05-12]. Dostupné z: https://en.wikipedia.org/wiki/Client–server_model

[18] Wikipedia, the free encyclopedia. WIKIMEDIA FOUNDATION, Inc. *Web 2.0* [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://en.wikipedia.org/wiki/Web_2.0

[19] Wikipedia, the free encyclopedia. WIKIMEDIA FOUNDATION, Inc. *Create, read, update and delete* [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

[20] Wikipedia, the free encyclopedia. WIKIMEDIA FOUNDATION, Inc. *Client (computing)* [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://en.wikipedia.org/wiki/Client_(computing)

[21] WordPress: Blog Tool, Publishing Platform, and CMS. [online]. [cit. 2013-05-12]. Dostupné z: http://wordpress.org/

[22] Drupal: Open Source CMS. [online]. [cit. 2013-05-12]. Dostupné z: http://drupal.org/

[23] iGoogle. GOOGLE. [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://www.google.com/ig

[24] Webnode.cz: Vytvořte si web zdarma. WEBNODE. [online]. 2013 [cit. 2013-05-12]. Dostupné z: http://www.webnode.cz/

[25] Netvibes: Social media monitoring, Analytics and Alerts Dashboard. NETVIBES. [online]. [cit. 2013-05-12]. Dostupné z: http://www.netvibes.com/

[26] Nettuts+: How to Mimic the iGoogle Interface. PADOLSEY, James. [online]. 2010 [cit. 2013-05-12]. Dostupné z: http://net.tutsplus.com/tutorials/javascript-ajax/inettuts/

Příloha A – Obsah CD

- zdrojový kód aplikace
- technická zpráva
- manuál ovládání aplikace a manuál tvorby vlastních widgetů
- plakát
- demonstrační video