Sem vložte zadání Vaší práce.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Tvorba webové aplikace pro návrh a tisk nástěnných kalendářů

Michal Konečný

Katedra softwarového inženýrství
Vedoucí práce: Ing. Josef Pavlíček, Ph.D.

6. května 2020

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, 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 zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2020 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2020 Michal Konečný. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Konečný, Michal. *Tvorba webové aplikace pro návrh a tisk nástěnných kalendářů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a samotnou implementací webové aplikace pro návrh a tisk nástěnných kalendářů.

Práce se nejprve věnuje analýze současných řešení, dále pak dostupným technologiím, vhodným architektonickým vzorům, požadavkům na samotnou aplikaci a rozborem případů užití.

Práce dále navazuje návrhem na použití konkrétních technologií, zpracováním doménového modelu, návrhem designu z hlediska rozložení prvků a pohledem na design aplikace skrze Nielsenovu heuristiku.

Realizační část je zaměřena na konkrétní implementační detaily. Práce se zde zabývá souborovou strukturou aplikace, podrobným popisem řešení uživatelského požadavku, jenž se týká možnosti stažení výsledného návrhu kalendáře do počítače a též řešením požadavku týkajícího se autentizace a autorizace. Také je zde nastíněna možná budoucnost projektu.

Klíčová slova kalendář, webová aplikace, nástěnný kalendář, návrh kalendáře, aplikace, Java, Spring Framework, vývoj webové aplikace

Abstract

This bachelor thesis deals with the analysis, design and implementation of a web application for designing and printing wall calendars.

Firstly, the thesis is taking look at the analysis of current solutions, then the available technologies, suitable architectural patterns, requirements for the application itself and the analysis of use cases.

As a next subject, the thesis also follows up with a proposal for the use of specific technologies, the processing of a domain model, a design proposal in terms of the layout of elements and a view of the application design through Nielsen heuristics.

The implementation part is focused on specific implementation details. The thesis deals with the file structure of the application, a detailed description of the solution of the user request, which concerns the possibility of downloading the final calendar design to a computer and also the solution of the request related to authentication and authorization. The possible future of the project is also outlined here.

Keywords calendar, web application, wall calendar, calendar design, application, Java, Spring Framework, web application development

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Technologie dnešní doby nám umožňují vytvořit fotografii během několika sekund. Stačí doslova pár dotyků na obrazovku mobilního telefonu. Pořízené fotografie nám však plní úložiště nejen v chytrých telefonech, ale i na počítačích. Jenže co s nimi? Dříve si je lidé nechávali vyvolat a ukládali do alb. Dnes se s nimi spíše pochlubíme přátelům na sociálních sítích a následně upadnou v zapomnění. V poslední době se však objevil jistý trend. Možnost využít fotografie k vlastní tvorbě dárkového předmětu. A tak můžeme vidět, jak vznikají hrníčky s vlastním návrhem potisku, foto rámečky, oblečení, hodiny a v neposlední řadě také kalendáře.

V dnešní době nalezneme na internetu celou řadu aplikací zabývajících se problematikou tvorby kalendáře. Často však jde o rozsáhlé aplikace s možností vlastního návrhu nejen kalendáře, ale i dalších výrobků. Aplikace jsou propojeny s konkrétním e-shopem, jenž vám navržený předmět vyrobí. Co když si však chce uživatel kalendář vytisknout sám? Nebo zaslat do jiného e-shopu?

Aplikace, která je výstupem praktické části této bakalářské práce, umožňuje uživateli si výsledný návrh stáhnout a naložit s ním podle jeho libosti. Díky specializaci na jeden konkrétní produkt, je běh aplikace rychlý, plynulý a prostředí je uživatelsky přívětivé a intuitivní.

Hlavním cíle této práce je vytvoření webové aplikace pro návrh a tisk nástěnných kalendářů. Za tímto účelem bude v teoretické části provedena analýza současného stavu aplikací, umožňující podobnou činnost. Na základě této analýzy bude navržen vhodný architektonický model pro běh aplikace na aplikačním serveru. Též budou vybrány vhodné technologie pro vývoj tohoto typu aplikace.

Na základě rešerše budou zpracovány uživatelské požadavky a podrobně popsány případy užití, které představují základní funkční stránku aplikace.

Tato práce má následující strukturu. V kapitole 1 (Analýza) se práce zabývá současným stavem aplikací, umožňující podobné úlohy, jejich hlavními

výhodami a nevýhodami a uživatelskou přívětivostí. Dále je provedena analýza uživatelských požadavků, které vzešly primárně od zadavatele samotného, též však na základě analýzy současných řešení. Zároveň jsou zde popsány jednotlivé případy užití, které charakterizují základní funkčnost aplikace. Závěrem je provedena analýza technologií a analýza architektonických vzorů.

V kapitole 2 (Návrh) jsou podrobně popsány technologie použité pro vývoj výsledné aplikace, podrobněji rozebrán použitý architektonický vzor a také popsán průběh návrhu designu z hlediska rozložení jednotlivých prvků aplikace. V návaznosti na případy užití, jenž byly rozebrány v předešlé kapitole, byl do této kapitoly též zahrnut rozbor doménového modelu. Na konci kapitoly je pak popsána a dána do souvislosti s výslednou aplikací Nielsenova heuristika.

V kapitole 3, zabývající se realizací, jsou rozvedeny konkrétní kroky, které vedly ke zhotovení aplikace. Nejprve je podrobně popsána souborová struktura aplikace a některé implementační či vývojové detaily. Dále pak autor práce popisuje realizaci dvou konkrétních uživatelských požadavků, seznamuje nás s bezpečností aplikace a přibližuje průběh testování aplikace a závěrem se zamýšlí nad možným budoucím rozvojem aplikace.

V závěru autor práce především hodnotí, zda bylo dosaženo jednotlivých cílů, které byly před začátkem projektu stanoveny.

Kapitola 1

Analýza

Před zahájením samotné tvorby webové aplikace je vždy potřeba provést dvě základní věci. První z nich je konzultace se zákazníkem. Z této konzultace by mělo vyplynout ujasnění si základních požadavků na aplikaci, dále zákazníkova představa o tom, jak by se aplikace měla chovat, stanovit priority funkcí a v neposlední řadě se dotknout též vzhledu aplikace a možnosti využití na různých zařízení. Důležité je pochopit prostředí, pro které bude aplikace vznikat a chování koncových uživatelů.

Druhou základní věcí je analýza trhu. Zjistit, zda podobná, či zcela totožná aplikace již nevznikla. V případě, že takové aplikace existují, je zcela na místě prozkoumat současná řešení. Zjistit jejich výhody a nevýhody a vyvarovat se chyb a nedostatků, které u konkurence vznikly. Často se stává, že zákazník sám nás upozorní na řešení, které by se mu zamlouvalo, pokud by mělo navíc nějakou funkci, jiný vzhled nebo lepší zabezpečení.

Na základě výše uvedeného, se autor práce bude v této kapitole nejprve zabývat současnými řešeními a následně rozebere požadavky zákazníka a požadavky vyplývající z účelu, ke kterému by měla výsledná aplikace sloužit.

1.1 Současná řešení

V současné době nalezneme na internetu mnoho aplikací nabízející návrh vlastního nástěnného kalendáře. Autor práce se rozhodl analyzovat čtyři konkrétní služby a rozebrat jejich výhody a nevýhody z pohledu zákazníka.

1.1.1 CEWE FOTOLAB

První aplikací, na kterou se autor práce blíže zaměřil je CEWE FOTOLAB (`https://www.fotolab.cz`). Důvodem pro analýzu této aplikace byla autorova vlastní zkušenost s ní. Jedná se o rozsáhlou aplikaci propojenou s e-shopem nabízející spoustu služeb. Kromě prodeje fototechniky, nabízí též navržení produktů s vlastním potiskem či designem. Ať už jde o fotoknihy, kryty na

mobil, pouhý tisk fotografií, různé druhy fotodárků, přání a v neposlední řadě též kalendáře.

Na výběr je více druhů kalendářů - nástěnné, plánovací, roční a stolní. Můžeme volit různou velikost a různý poměr stran. Po vstupu do samotného online návrháře si vybereme design. Pak už máme možnost nahrát fotografie a začít je umisťovat do míst, kde nám to nejvíce vyhovuje. Návrhář má spoustu dalších možností - volba barevného schématu, designových prvků nebo rozvržení jednotlivých stránek.

Uživateli může chvíli trvat než se v návrháři zorientuje a najde potřebné funkce. Avšak pokud se uživatel spokojí s před připravenou šablonou, může mít kalendář hotov během několika minut. Návrhář zároveň uživateli signalizuje, zda je kvalita fotografií dostatečná pro tisk či nikoliv.

Pro objednání a tisk kalendáře je nutná registrace. Stejně tak je nutné mít založený účet, pokud si kalendář přejete dočasně uložit a pokračovat na jeho tvorbě později. Díky účtu budete mít přehled o všech svých doposud vytvořených produktech.

Mezi nesporné výhody CEWE FOTOLAB patří široká nabídka možností při návrhu kalendáře. Návrhář může na první pohled vypadat neintuitivně a uživateli může chvíli trvat, než najde funkci či nastavení, které zrovna potřebuje. O něco lépe než online návrhář je na tom desktopová aplikace, kde je více prostoru a funkce jsou zřetelné na první pohled. Další výhodou je určitě možnost uložení rozpracovaného kalendáře, díky založení si uživatelského účtu. Díky účtu získáte také možnost uchovávat si historii všech svých vytvořených či rozpracovaných kalendářů a jiných produktů. Produkt je však následně možné objednat pouze u společnosti CEWE FOTOLAB.

1.1.2 Albumo

Další aplikací umožňující vlastní návrh kalendáře je
Albumo (`https://www.albumo.cz`). Albumo dříve disponovalo podobně jako CEWE FOTOLAB desktopovou aplikací, avšak nyní již pracuje pouze s online designérem. Na své domovské stránce mají uvedeno, že jejich aplikace podporuje všechny druhy zařízení včetně mobilních. V nabídce mají kromě kalendářů, také fotoknihy, fotoobrazy a tisk fotografií samotných.

Návrhář kalendáře nás přivítá výběrem velikosti stran a tématem. Je zde možnost přeskočit volbu tématu a pracovat tak se svou vlastní fantazií. Návrhář má oproti CEWE FOTOLAB intuitivnější rozhraní. Všechna nastavení týkající se kalendáře nalezneme na levé straně, v pravé části pracujeme se samotným kalendářem a v horní části nalezneme možnosti uložení, navrácení provedených úprav či objednání. Vzhledem k možnostem jaké návrhář nabízí si s kalendářem můžeme opravdu vyhrát. Ale i zde nalezneme několik před připravených šablon, do kterých následně zbývá pouze umístit naše fotografie a práce může být brzy hotova. I v Albumo nalezneme upozornění na nízkou kvalitu fotografií.

Pokud si chceme rozpracovaný návrh uložit je nutné mít na Albumo založený účet. Zatímco návrhář je uživatelsky přívětivý, najít ve svém účtu doposud vytvořené návrhy nám nějaký čas zabere. Bohužel ani návrat k rozpracovanému návrhu není vyřešen dokonale, jelikož je nutné najít poměrně malé tlačítko, přes které se k další editaci dostaneme.

Albumo nabízí velmi povedený online designér se spoustou možností vlastních voleb, ale i spoustou před připravených témat. Pokud bychom však chtěli návrh vytvářet na menším zařízení, nebude to snadné ani uživatelsky přívětivé. Problém nastane určitě u mobilních zařízení, protože spoustu ikonek pro nastavení kalendáře se nám skryje mimo obrazovku. Na standardní PC obrazovce nebo u tabletu však není sebemenší problém. I zde platí, že produkt je následně možné objednat pouze u této společnosti.

1.1.3 Fotokalendare.cz

Pokud chce mít člověk návrh nástěnného kalendáře hotov během pár minut, tak právě pro něj je určena aplikace
Fotokalendare.cz (`https://www.fotokalendare.cz`). Fotokalendare.cz má ve své nabídce pouze kalendáře a fotoknihy. Po volbě velikosti a orientace strany se dostáváme k volbě šablony. Na výběr máme ze spousty možností, k výběru šablony se není možné později vrátit, ani ji dále upravovat. Po volbě šablony již zbývá pouze nahrát fotografie a umístit je do kalendáře. Žádné další nastavení a možnosti nehledejte. Jednoduché a rychlé.

Zajímavým způsobem je řešena možnost uložení produktu. Není zde nutná jakákoliv registrace. Stačí zadat e-mail, na který vám přijde odkaz, přes který se ke svému návrhu můžete kdykoliv vrátit. Registrace není nutná ani při objednávce. Objednat zboží si opět můžete pouze u Fotokalendare.cz.

Nespornou výhodou této aplikace je jednoduchost. Vše je jasné, srozumitelné a uživatelsky přívětivé. Stačí pár kliknutí a návrh máte hotov. Naopak velkou nevýhodou pro někoho může být nemožnost jakékoliv vlastní volby a úprav. Uživatel je odkázán pouze na před připravené šablony. Díky jednoduchosti aplikace je zde jednoduše řešena i možnost uložení rozpracovaného návrhu ve formě odkazu, který je vám odeslán na zadaný e-mail.

1.1.4 Printomat

Podobně jednoduchým návrhářem jako má aplikace Fotokalendare.cz disponuje též aplikace Printomat (`https://printomat.cz`). Printomat nabízí fotoprodukty všech druhů. V nabídce můžeme najít hrnky, obrazy, plátna, fotky, fotoknihy, pexeso, prostírání a ani zde nechybí kalendáře.

Kalendáře zde nalezneme roční, nástěnné a stolní. Po výběru jedné z možností se dostaneme k nabídce velikosti a orientace. Následuje volba motivu, který podobně jako u předchozí aplikace nelze již dále měnit či upravovat. Co

lze však měnit je možnost rozložení fotografií a jejich počet. Na výběr máme ze dvou možností - jedna fotografie na stránku nebo dvě fotografie na stránku.

Pokud si chceme rozpracovaný návrh uložit, je nutné mít založený účet. V účtu pak kromě historie našich objednávek a faktur, nalezneme právě i naše rozpracované a uložené návrhy všech produktů.

Pokud si nejprve chceme návrhář vyzkoušet bez přihlášení k účtu, může nás překvapit, že poté, co si návrh vytvoříme a dáme uložit, jsme přesměrování na možnost přihlášení a registrace. Poté se však nevrátíme k našemu rozpracovanému návrhu, ale na úvodní stranu aplikace. To v nás může vzbudit pocit, že náš návrh je ztracen, ale není tomu tak. Pokud se opět vrátíme do návrháře, objeví se možnost pokračovat v rozpracovaném návrhu, díky které se dostaneme zpět k našemu návrhu. Podobně neintuitivní je i možnost opustit rozpracovaný návrh. Na výběr máme pouze z možností uložit a vytvořit objednávku. V případě že návrh máme uložen a chceme designér opustit, musíme buď zavřít okno prohlížeče nebo kliknout na logo aplikace. To v čem však Printomat vyniká je bezesporu široká nabídka produktů.

1.2 Analýza požadavků

Analýza požadavků v softwarovém inženýrství je jednou z prvních fází vývoje softwaru. Cílem analýzy je vymezit funkční požadavky softwaru ze strany zadavatele (zákazníka). Zároveň je však důležité brát v úvahu požadavky na systém ze strany jiných zúčastněných stran, nejčastěji uživatelů. Díky analýze požadavků můžeme přesněji odhadnout pracnost, vymezíme hranice systému a též zachytíme omezení, která jsou na software kladena.

Dle [1] má mít správný požadavek následující vlastnosti:

- dosažitelnost
- ověřitelnost
- jednoznačnost.

Za dosažitelný je považován takový požadavek, který odpovídá potřebě či cíli, jehož řešení je technicky možné v rámci uvažovaných nákladů.

Požadavek, který je definován slovy jako je nadměrný, dostatečný nebo rezistentní, nelze považovat za ověřitelný. Je třeba definovat požadavek takovým způsobem, abychom očekávaný výkon a funkcionalitu mohli kvantitativně ověřit.

Jednoznačností rozumíme požadavek, který má jediný možný význam.

Dále dle [1] musí být požadavek kompletní, tedy zachytit funkční a údržbový koncept, prostředí, ve kterém bude software používán a jeho omezení. To nám umožní lépe pochopit potřeby žadatele (zákazníka).

Musí popisovat potřebu nikoli řešení. Zabývat se otázkou Proč? a Co?, ne však otázkou Jak?

Musí být konzistentní se všemi ostatními požadavky na systém. Požadavek by měl být vhodně definován z hlediska systémové hierarchie. Pokud například evidujeme podrobné požadavky týkající se komponent systému, neměli bychom je sdružovat s požadavky na systém samotný.

1.2.1 Kategorizace požadavků

Požadavky je vhodné dělit do různých kategorií. Díky kategorizaci mají vývojáři usnadněnou práci při implementaci a také při sběru požadavků od uživatelů. Existuje celá řada kategorizací. Níže si popíšeme některé z nich. Autor práce se rozhodl v této práci použít kategorizaci FURPS+.

1.2.1.1 Základní kategorizace

Požadavky se mohou jednoduše dělit na

- funkční
- nefunkční (obecné).

Nefunkční nebo též obecné požadavky určují například výše zmíněná omezení na software, dodržování standardů atd. Obecně řečeno se jedná o požadavky, které sami o sobě neplní nějakou funkci v rámci softwaru.

Funkční požadavky zachycují chování a vlastní funkcionalitu systému. Jde o požadavky, které řeší konkrétní problém v dané aplikaci.

1.2.1.2 Kategorizace FURPS

FURPS je zkratkou pro Functionality (funkčnost), Usability (použitelnost), Reliability (spolehlivost), Performance (výkon) a Supportability (podporovatelnost). V [2] se nachází vůbec první zmínka o modelu FURPS. *„Jedním z důvodů, proč je FURPS efektivní je ten, že jde o jednoduchou mnemotechnickou pomůcku, kterou si snadno zapamatujete. Použití modelu FURPS zahrnuje dva kroky: stanovení priorit a stanovení měřitelných atributů kvality.“* [2, s. 159] (překlad dle autora) Krátce popíšeme jednotlivé kategorie:

- funkčnost: zaměření na hlavní funkcionalitu aplikace a její chování
- použitelnost: požadavky týkající se koncového uživatele, například uživatelské rozhraní, dokumentace atd.
- spolehlivost: kategorie zabývající se četností a závažností chyb, patří zde požadavky na zátěžové testy a jiné
- výkon: požadavky specifikující výkon systému jako celku či dílčích komponent
- podporovatelost: kategorie, do které spadají požadavky na další rozšiřitelnost aplikace či dlouhodobou podporu a údržbu

1.2.1.3 Kategorizace FURPS+

Kategorizace FURPS+ je jednou z nejrozšířenějších kategorizací analýzy požadavků. Jedná se o rozšíření kategorizace FURPS. Ono „+“ ve zkratce FURPS+, přidává další nefunkční kategorie, které se týkají především návrhu, implementace, rozhraní a fyzického rozhraní. Podívejme se blíže na jednotlivá omezení:

- návrhová omezení (design constraints): omezení řešící celkový návrh aplikace, například návrh databáze, kdy je předem pevně stanoveno, jaký typ databáze je třeba použít
- implementační omezení (implementation constraints): omezení standardů, jazyka nebo operačního systému a jeho nastavení
- omezení rozhraní (interface constraints): omezení na spolupráci a komunikaci s externími jednotkami
- fyzická omezení (physical constraints): omezení řešící materiál, tvar, rozměry či váhu hardwaru

1.3 Uživatelské požadavky

První požadavky na systém vždy vzejdou od zadavatele. Další požadavky mohou následovat od přímých uživatelů či ze stávajících řešení. V případě této práce vzešla většina požadavků od zadavatele, další pak z analýzy současných řešení viz. sekce 1.1.

1.3.1 Nefunkční požadavky a omezení

Jak již bylo výše zmíněno, autor práce se rozhodl pro kategorizaci požadavků metodou FURPS+. Nejprve se zaměříme na nefunkční požadavky a omezení.

1.3.1.1 Požadavky na použitelnost

N1 – Aplikace bude dostupná skrze webové rozhraní: Přístup přes webové rozhraní je u aplikací tohoto typu samozřejmostí. U webového rozhraní je důležité uživatelsky přívětivé a intuitivní prostředí.

1.3.1.2 Požadavky na podporovatelnost

N2 – Návrhář bude navržen tak, aby jej bylo možné dále uživatelsky rozšiřovat: Tato práce má za úkol implementovat základní funkcionalitu aplikace. Předpokládá se, že v budoucím čase se bude aplikace rozšiřovat o další funkcionalitu, ať už autorem této práce nebo kýmkoliv jiným. Je proto více než vhodné držet se přinejmenším základních programovacích best practices (osvědčených postupů).

1.3.1.3 Návrhová omezení

N3 – Zapojení databáze Apache Derby: S ohledem na povahu serveru, kam bude výsledná aplikace po vytvoření umístěna, zadavatel silně doporučuje použít pro vývoj a výsledné nasazení databázi Apache Derby.

Je potřeba napsat vedoucímu, aby napsal, proč chce zrovna Derby.

1.3.1.4 Implementační omezení

N4 – Nasazení na zadavatelem zvolený server: Aplikace bude umístěna na server, který určí zadavatel práce. Je potřeba brát to v potaz.

Je potřeba napsat vedoucímu, aby doplnil informace o serveru.

1.3.2 Funkční požadavky

V této části autor práce podrobně popíše jednotlivé funkční požadavky, které jsou stěžejní pro chod aplikace. U každého požadavku bude též stanovena priorita a složitost.
Poznámka: Některé z funkční požadavků nebyly dodány zadavatelem, ale vzešly z analýzy současných řešení.

F1 – Autorizace a autentizace: Aplikace bude umožňovat registraci uživatele. Díky uživatelským účtům bude možno evidovat historii vytvořených či rozpracovaných kalendářů (viz. požadavek F4). Z hlediska autentizace je potřeba zajistit, aby se uživatelé dostali pouze ke svým vytvořeným návrhům. Administrátor bude mít právo editovat účty jednotlivých uživatelů a sledovat statistiky či samotné návrhy kalendářů všech uživatelů.

Priorita: Vysoká
Složitost: Vysoká

F2 – Tvorba kalendáře: Hlavní funkcí aplikace bude tvorba návrhu nástěnného kalendáře. Návrh kalendáře bude ve formátu A4, uživatel si bude moci nahrát do návrháře vlastní fotografie, vybrat si vlastní či před připravený design. V návrháři bude dále možnost volby rozložení prvků kalendáře.

Priorita: Vysoká
Složitost: Vysoká

F3 – Stažení kalendáře: Aplikace bude umožňovat stažení kalendáře ve formátu PDF.

Priorita: Vysoká
Složitost: Vysoká

F4 – Evidence historie vytvořených/rozpracovaných kalendářů: Autorizovaný uživatel si bude moci uložit svůj rozpracovaný či dokončený návrh

a později se k němu vrátit.

Priorita: Střední
Složitost: Nízká

F5 – Vytvoření grafických šablon: V návrháři bude uživateli umožněno vybrat si z před připravených grafický šablon. Budou vytvořeny minimálně dvě grafické šablony.

Priorita: Vysoká
Složitost: Nízká

F6 – Možnost vyzkoušet návrhář bez autorizace: Aplikace umožní i neautorizovaným uživatelům vyzkoušet si prostředí návrháře.

Priorita: Nízká
Složitost: Nízká

F7 – Nápověda k použití aplikace: V případě, že bude mít návrhář více funkcí, může se stát, že některé z funkcí budou skryty za různými tlačítky či ikonami. Je proto vhodné nabídnout uživateli jednoduchou nápovědu, která ho s prostředím rychle obeznámí.

Priorita: Nízká
Složitost: Nízká

1.4 Analýza případů užití

„Případy užití vznikají během sběru a analýzy požadavků a zachycují klíčovou funkcionalitu aplikace. Zaměřují se na chování systému z hlediska vnějšího pohledu.“ [3, s. 25] Díky analýze případů užití budou mít vývojáři skutečný přehled všeho, co bude potřeba naprogramovat. Dle případů užití lze sestavovat akceptační testy, mohou být základem pro uživatelskou příručku nebo lze díky nim lépe odhadnout pracnost na celém projektu.

1.4.1 Případy užití

Dále budou popsány jednotlivé případy užití. U každého případu užití bude uveden název, popis a hlavní aktér. U zajímavějších případů užití bude podrobně rozepsán scénář, který bude popisovat interakci mezi uživatelem a systémem.

1.4.1.1 UC1 – Vytvoření kalendáře

Popis: Uživatel si chce vytvořit nový návrh nástěnného kalendáře. To je možné provést přes hlavní menu v horní části aplikace.
Scénář: Vyplnění všech potřebných nastavení a údajů, nahrání fotografií a uložení výsledného návrhu.

1. Uživatel se přihlásí do svého účtu.
2. Systém uživateli zobrazí úvodní obrazovku.
3. Pokud uživatel doposud nevytvořil žádný návrh:
   a) Systém uživateli zobrazí možnost přejít k tvorbě kalendáře pomocí jednoduchého odkazu.
4. Jinak uživatel přejde k tvorbě nového návrhu pomocí odkazu v hlavním menu v horní části aplikace.
5. Systém uživateli zobrazí obrazovku s jednoduchým formulářem.
6. Uživatel vyplní název kalendáře, rok kalendáře, vybere rozložení kalendáře, design kalendáře nebo vlastní volbu barev a nechá si takto nastavený kalendář zobrazit.
7. Pokud uživatel některou z hodnot nevyplní:
   a) Systém uživatele na tuto skutečnost upozorní a neumožní mu dále pokračovat.
8. Jinak systém uživateli zobrazí pohled na jednotlivé strany kalendáře, počínaje úvodní stranou. Na stejné obrazovce systém uživateli nabídne další možnosti nastavení, či úpravu současných nastavení.
9. Uživatel nahraje do návrháře kalendáře fotografie, vloží je na místa v kalendáři a kalendář uloží.
10. Systém kalendář uloží a uživateli zobrazí pohled na výsledný návrh.

Hlavní aktér: Autorizovaný uživatel

1.4.1.2 UC2 – Úprava kalendáře

Popis: Uživatel si chce upravit již vytvořený či rozpracovaný návrh.
Hlavní scénář: Uživatel si vybere jeden ze svých návrhů a přejde rovnou na editaci.

1. Uživatel se přihlásí do svého účtu.
2. Systém uživateli zobrazí úvodní obrazovku.

3. Pokud uživatel doposud žádný návrh nevytvořil:

   a) Systém uživateli nabídne možnost vytvořit zcela nový návrh a případ končí.

4. Jinak systém uživateli na úvodní obrazovce zobrazí všechny jeho doposud vytvořené návrhy.

5. Uživatel nalezne návrh, se kterým chce dále pracovat a vybere si jednu z možností - „Editovat“ nebo „Zobrazit“.

6. Pokud uživatel vybere možnost „Editovat“:

   a) Systém zobrazí kalendář v editačním módu.

   b) Uživatel provede potřebné úpravy.

   c) Systém změny uloží a zobrazí uživateli aktualizovaný kalendář.

7. Pokud si uživatel vybere možnost „Zobrazit“:

   a) Systém kalendář zobrazí na nové obrazovce.

   b) Pokud si uživatel v levé části obrazovky vybere možnost „Editovat“:

      i. Systém zobrazí kalendář v editačním módu.

      ii. Uživatel provede potřebné úpravy.

      iii. Systém změny uloží a zobrazí uživateli aktualizovaný kalendář.

Hlavní aktér: Autorizovaný uživatel

1.4.1.3 UC3 – Stažení kalendáře

Popis: Uživatel se přihlásí ke svému účtu. Z uložených návrhů si vybere ten, který chce stáhnout. Z nabízených možností vybere možnost „Zobrazit“. Zobrazí se mu výsledný kalendář v plné velikosti. V levé části obrazovky vybere možnost „Stáhnout“. Kalendář se automaticky stáhne do PC ve formátu PDF.
Hlavní aktér: Autorizovaný uživatel

1.4.1.4 UC4 – Smazání kalendáře

Popis: Uživatel si mezi svými návrhy vybere ten, který chce smazat. Smazat kalendář je možné dvěma způsoby. První způsob je okamžité smazání ihned po nalezení návrhu. Druhým způsobem je možnost, si kalendář nejdříve zobrazit, a následně pomocí tlačítka na levém okraji obrazovky kalendář smazat.
Hlavní aktér: Autorizovaný uživatel

1.4.1.5 UC5 – Zobrazení kalendáře

Popis: Uživatel si ze svých uložených návrhů vybere ten, který si chce zobrazit. Z nabízených možností pak vybere možnost „Zobrazit“.
Hlavní aktér: Autorizovaný uživatel

1.4.1.6 UC6 – Úprava vlastního uživatelského účtu

Popis: Uživatel se po přihlášení přesune do pravého horního rohu aplikace, kde nalezne jednoduché menu, přes které se dostane k editaci svého profilu. Systém mu zobrazí jednoduchý formulář, kde si může změnit heslo, uživatelské jméno či email. Pokud dojde k jakékoliv editaci, po uložení je z aplikace odhlášen, aby došlo k aktualizaci změn.
Hlavní aktér: Autorizovaný uživatel

1.4.1.7 UC7 – Úprava cizího uživatelského účtu

Popis: Administrátor edituje některý z uživatelských profilů. Též může jakémukoliv uživateli přidat či odebrat práva administrátora.
Scénář: Administrátor nejprve uživatele vyhledá, následně provádí jeho editaci.

1. Administrátor se přihlásí ke svému účtu.
2. Systém zobrazí úvodní obrazovku s přehledem všech uživatelů a možností vyhledávání mezi nimi.
3. Administrátor vyhledá v seznamu uživatelů uživatele, kterého chce editovat.
4. Systém zobrazí jednoduchý formulář pro editaci vybraného uživatele.
5. Uživatel provede potřebné změny.
6. Systém změny uloží a zobrazí úvodní obrazovku s aktualizovaným přehledem všech uživatelů.

Hlavní aktér: Administrátor

1.4.1.8 UC8 – Vytvoření uživatelského účtu

Popis: Kromě registrace uživatelů, lze uživatele založit přímo z aplikace z prostředí s administrátorskými právy. Administrátor na své úvodní obrazovce nalezne kromě vyhledávání mezi již registrovanými uživateli též možnost vytvořit nového uživatele. Systém mu zobrazí jednoduchý formulář, přes který je možné uživatele vytvořit. Po vytvoření uživatele, systém administrátorovi zobrazí aktualizovaný seznam registrovaných uživatelů.
Hlavní aktér: Administrátor

1.4.1.9 UC9 – Smazání uživatelského účtu

Popis: Administrátor nalezne v přehledu uživatelů uživatele, jehož účet chce smazat. Následně z nabízených možností zvolí možnost „Smazat“. Systém poté administrátorovi zobrazí aktualizovaný seznam všech uživatelů.
Hlavní aktér: Administrátor

1.4.1.10 UC10 – Zobrazení cizího kalendáře

Popis: Administrátor má možnost nahlížet na návrhy kalendářů všech uživatelů. Návrhy uživatelů nalezne na své domovské obrazovce, kde je zobrazeno 10 nejnovějších návrhů, dále se k návrhům dostane přes hlavní menu aplikace. Po kliknutí na položku „Kalendáře“ systém zobrazí stránku s přehledem všech kalendářů. U každého kalendáře je pak možnost si jej zobrazit.
Hlavní aktér: Administrátor

1.4.1.11 UC11 – Zobrazení návrháře bez autorizace

Popis: Při otevření aplikace je komukoliv nabídnuta možnost vyzkoušet si aplikaci bez přihlášení. Po přístupu do této části aplikace, se uživateli zobrazí návrhář s možností nahrávat fotografie a zkoušet si různá nastavení kalendáře. Uživateli není umožněno si výsledný návrh uložit, je mu však nabídnuta možnost přihlášení či registrace.
Hlavní aktér: Uživatel

1.4.1.12 UC12 – Zobrazení nápovědy u návrháře

Popis: Poté, co se uživateli zobrazí samotný návrhář, má uživatel možnost si v levém horním rohu aplikace zobrazit jednoduchou nápovědu. Tato možnost je jak pro autorizované uživatele, tak i pro uživatele, kteří si zkouší návrhář bez přihlášení.
Hlavní aktér: Uživatel

1.4.2 Realizace případů užití

V tabulce 1.1 nalezneme vztah mezi funkčními požadavky na aplikaci a případy užití. Pokud by některý z funkčních požadavků nebyl ve vztahu s žádným případem užití, pak by se jednalo o zbytečný požadavek nebo o požadavek, na který se během tvorby případů užití zapomnělo.

1.4.3 Model případů užití

Na obrázku 1.1 vidíme model případů užití. Jsou zde zaznamenány jednotlivé případy užití, včetně aktérů, a vztahy mezi aktéry a případy užití.

Tabulka 1.1: Vztah funkčních požadavků a případů užití

Případy užití	Požadavky						
	F1	F2	F3	F4	F5	F6	F7
UC1	✓	✓			✓		✓
UC2	✓			✓	✓		✓
UC3	✓		✓	✓			
UC4	✓			✓			
UC5	✓			✓			
UC6	✓						
UC7	✓						
UC8	✓						
UC9	✓						
UC10	✓			✓			
UC11					✓	✓	
UC12							✓

1.5 Technologie

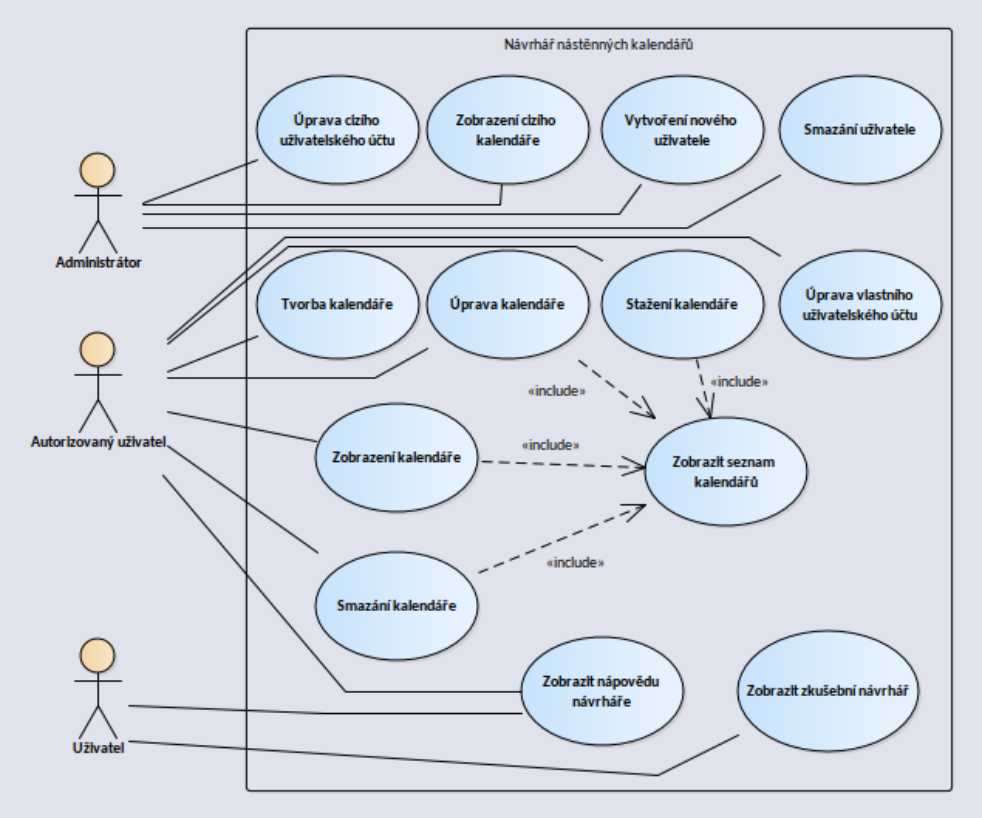
U každého projektu je třeba důkladně promyslet volbu technologií. Právě analýzou technologií se bude zabývat následující část práce. Při volbě technologií je třeba zvážit povahu projektu, jeho účel a využití. S ohledem na požadavek zadavatele na vytvoření webové aplikace, autor práce analyzuje pouze webové technologie. Webové technologie se neskutečným tempem vyvíjí, a tak lze bez nadsázky říci, že to, co bylo ještě včera považováno za novinku, je dnes již zastaralé. Technologie můžeme rozdělit do dvou skupin:

- Front-end
- Back-end

1.5.1 Front-end

Front-end technologie, jsou technologie, skrze které uživateli zpřístupňujeme obsah stránek. Definují rozmístění a kontext obsahu a v neposlední řadě též design výsledné aplikace. Mezi základní front-end technologie patří:

- HTML
- CSS
- JavaScript



Obrázek 1.1: Model případů užití

1.5.1.1 HTML

HTML neboli Hypertext Markup Language je značkovací jazyk, který vznikl v roce 1990 a jehož autory jsou Tim Berners-Lee a Robert Cailliau. Jazyk HTML je tvořen tagy (značkami – odtud značkovací) a jejich atributy. Tento styl zápisu vychází z jazyka SGML.

Poslední verzí je HTML5 a jeho subverze. Dle [4] *„je termín ‚HTML5‘ dnes široce používán pro označení moderních webových technologií, z nichž mnohé (v žádném případě však všechny) jsou vyvinuty na základě WHATWG.“* (překlad dle autora) Také výstup praktické části této bakalářské práce se drží standardů HTML5.

1.5.1.2 CSS

CSS neboli Cascading Style Sheets (kaskádové styly) mají za úkol popis zobrazení elementů na stránkách napsaných v jazycích HTML, XML a jiných. Používají se k oddělení struktury a obsahu dokumentu od vzhledu.

„CSS pravidlo obsahuje selektor, který je následován deklarací. Deklarace

začíná levou složenou závorkou ‚{' a končí pravou složenou závorkou ‚}'. Mezi nimi je pak buď prázdný seznam nebo seznam deklarací oddělených středníkem ‚;'." [5] (přeloženo dle autora)

1.5.1.3 CSS Preprocesory

„CSS preprocesor je program, který umožňuje generovat CSS, díky specifické preprocesorové syntaxi. Existuje množství CSS preprocesorů, z nichž můžeme vybírat, většina z nich nám přidává funkce, které v čistém CSS nenajdeme, jako jsou mixiny, selektor vnoření, selektor dědičnosti, atd. Díky těmto vlastnostem je pak struktura CSS čitelnější a snáze se udržuje." [6] (přeloženo dle autora)

Dále si představíme dva nejznámější CSS prepocesory, kterými jsou Sass a Less. Abychom mohli pracovat s CSS preprocesory, je zapotřebí mít na svém webovém serveru nainstalovaný CSS kompilátor.

Sass je CSS preprocesor, který se více podobá programovacímu jazyku. Dalo by se tedy říci, že je imperativního charakteru. Sass přináší již výše zmíněné výhody oproti čistému CSS:

- proměnné
- mixiny, parameterické mixiny (pro opakující se části kódu)
- zanořování selektorů, dědičnost
- funkce
- cykly, podmínky
- import souborů

Less je naproti tomu spíše deklarativního charakteru, tzn. že se mnohem více podobá čistému CSS a je tak snazší pro učení. Přináší ty stejné výhody jako preprocesor Sass. Dalo by se debatovat, který z jazyků má přehlednější zápis, avšak co do funkčnosti, jsou oba preprocesory na stejné úrovni. Autor práce se při vývoji aplikace rozhodl pro použití právě preprocesoru Less.

1.5.1.4 JavaScript

JavaScript (JS) je skriptovací jazyk, který běží na straně klienta. Díky tomu můžeme měnit obsah webové stránky u uživatele. Jedná se o jazyk, který je interpretovaný (překládaný za běhu), objektově-orientovaný a multiplatformní. Při použití JS je potřeba myslet na to, že uživatel si může kdykoliv JS ve svém prohlížeči vypnout nebo jej přepsat. Proto je třeba zajistit, aby se i v takovém případě zobrazila uživateli potřebná data nebo v případě, kdy JS používáme pro validaci formuláře, byla validace vždy provedena ještě také na straně serveru. Na JS je postavena celá řada frameworků a knihoven. Autor práce se rozhodl použít mimo čistého JS také knihovnu jQuery.

1.5.1.5 jQuery

jQuery je JS knihovna, která byla vydána v roce 2006 Johnem Resigem v rámci newyorského BarCampu. Jedním z důvodů, proč se jQuery stala populární knihovnou, je ten, že dokáže být nápomocná v celé řadě úkolů.

„jQuery knihovna poskytuje obecně cílenou abstraktní vrstvu pro běžné skriptování na webu, a právě proto je užitečná ve většině skriptovacích situací.“ [7, s. 6] (překlad dle autora) Dále dle [7, s. 6] mezi základní funkce jQuery patří:

- Přístup k jednotlivým částem stránky
- Změna vzhledu stránky
- Pozměnit obsah stránky
- Reagovat na uživatelovu interakci se stránkou
- Přidat animaci
- Získat informace ze serveru, bez potřeby obnovení stránky
- Zjednodušit běžné JS úlohy

1.5.2 Back-end

Back-end technologie oproti front-end technologiím běží na serveru. Server přijaté požadavky od klienta obslouží (zpracuje) a výsledky vrací klientovi, který je pomocí front-end technologií zobrazuje uživateli. Back-end je tedy uživateli skrytý a klientská aplikace si z něj pouze odebírá data. V back-endu aplikace nalezneme základní logiku webové stránky, softwaru nebo informačního softwaru.

Existuje celá řada back-end technologií. V dnešní době se pracuje především s webovými frameworky, které vychází z různých jazyků. Níže se podíváme na některé z nich.

1.5.2.1 Java

Java je objektově orientovaný programovací jazyk, který byl představen v roce 1995 společností Sun Microsystems. Dle [8] je Java nejpopulárnějším programovacím jazykem. Žebříček je sestaven na základě počtu vyhledávaných dotazů v internetových vyhledávačích jako jsou Google, Bing, Yahoo, Wikipedia, Youtube, ... Žebříček je jednou měsíčně aktualizován a snaží se zachytit pouze popularitu jazyka nikoliv jeho kvalitu nebo četnost použití.

Nespornou výhodou programovacího jazyka Java je jeho platformní neutrálnost. V této souvislosti se můžeme setkat s pojmem *Java virtual machine*

(JVM). „JVM je specifikace abstraktního stroje, díky kterému může kompilátor Javy generovat zdrojový kód. Konkrétní implementace JVM pro konkrétní hardwarové a softwarové platformy následně poskytují konkrétní realizaci virtuálního stroje. JVM je založen na specifikaci POSIX – průmyslově standardní definici přenosného systémového rozhraní.“ [9] (překlad dle autora)

Stejně jako u front-endových technologií, také u back-endu se můžeme setkat s celou řadou frameworků pro jednotlivé technologie. Nejinak je tomu i u Javy. Blíže se podíváme na dva z nich, které jsou vyučovány v předmětu BI-TJV.

1.5.2.2 Spring

Spring je komplexní Java framework. Spring je natolik komplexní, že bychom jej mohli definovat jako celek, který nám usnadní řešení různých technických problémů. Mezi největší výhody Spring frameworku patří:

- modularita – možnost využití jen části Springu, která zrovna řeší náš problém
- testovatelnost – umožňuje snadné testování
- nezávislost kódu na aplikačním použití
- pokrývá všechny vrstvy aplikace

Právě framework Spring a jeho části je použitý při vývoji výsledné aplikace v rámci této práce.

1.5.2.3 Vaadin

Vaadin je open source webový framework, který pomáhá Java vývojářům, s minimem úsilí dosáhnout velké uživatelské spokojenosti. Vaadin je jediným frameworkem, jenž umožňuje vytvářet uživatelské prostředí zcela v Javě, bez potřeby řešit JS, HTML nebo CSS. Avšak též není problém, tyto technologie ve svém projektu použít. Aplikace Vaadin běží na serveru a zpracovává veškerou komunikaci automaticky a bezpečně. Toto vychází z [10].

1.5.2.4 PHP

PHP je skriptovací programovací jazyk. Je využíván primárně pro psaní internetových stránek a webových aplikací. Je hojně rozšířen také díky využití ve spoustě známých CMS systémů, jako je Drupal nebo Wordpress. PHP je dle [11] nejrozšířenějším skriptovacím jazykem pro web. Mezi dvě nesporné výhody PHP patří rozsáhlost funkcí a nativní podpora mnoha databázových systémů.

Jeho rozšířenost je dána také množstvím frameworků vycházejících právě z jazyka PHP. Frameworky se snaží udržet výhody samotného PHP a přidávají některé další. Dle [12] mezi benefity frameworků patří:

- organizace kódu pro jeho snadnou udržovatelnost
- snadné ladění kódů
- urychlení procesu vývoje
- zlepšení responzivity aplikací
- zvýšení bezpečnosti

Mezi 8 nejpopulárnějších PHP frameworků patří též dle [12]:

- Laravel
- CodeIgniter
- Symfony
- CakePHP
- Yii
- Zend
- FuelPHP
- Phalcon

1.5.2.5 Python

Python je podobně jako PHP skriptovací programovací jazyk. Vznikl v roce 1991 a jeho autorem je Guido van Rossum. Jazyk je díky své jednoduché a čisté syntaxi vhodný pro začínající programátory. Je dynamicky interpretovaný, tedy kód se překládá až během běhu programu. Mezi hlavní výhody Pythonu tedy patří, že je:

- objektově orientovaný
- má přehledný kód
- přenositelný mezi platformami

Jako nejpopulárnější webové frameworky napsané v Pythonu, označuje [13] tyto:

- Django
- CherryPy
- Pyramid

1.6 Architektonický vzor

Následující část je zaměřena na analýzu architektonických vzorů. Architektonický vzor je většinou obecné, opakovatelně použitelné řešení problému, jenž se vyskytuje při vývoji softwaru. Architektonické vzory mají za úkol popsat základní vlastnosti a chování aplikace. Mezi další cíle patří například:

- stanovení komponent a rozhraní,
- určení granularity komponent,
- přiřazení zodpovědnosti jednotlivým třídám.

Během vývoje softwaru je možné použít pro různé části aplikace více architektonických vzorů. Existuje celá řada architektonických vzorů včetně jejich různých dělení. Autor práce se rozhodl pro popis pěti vybraných na základě [14]. Jedná se o tyto architektonické vzory:

- Layered Architecture (Vícevrstvá architektura)
- Event-Driven Architecture (Událostmi řízená architektura)
- Microkernel Architecture (Architektura mikrojádra)
- Microservices Architecture (Architektura mikroslužeb)
- Space-Based Architecture (Prostorová architektura)

1.6.1 Vícevrstvá architektura

Vícevrstvá architektura známá též jako n-vrstvá architektura je jednou z nejznámějších architektur mezi architektonickými vzory. Hojně je využívána při vývoji enterprise aplikací. Také celá řada frameworků je postavena právě na tomto návrhovém vzoru, např. Java EE, Drupal nebo Spring MVC.

Kód je rozdělen do několika vrstev. Data na počátku vstupují do nejvyšší vrstvy a následně postupují směrem dolů, dokud nedosáhnou nejnižší vrstvy, ve které se často nachází databáze. Každá vrstva má na starosti jinou úlohu. Je běžnou skutečností, že při týmovém vývoji aplikace, pracují jednotlivý programátoři nezávisle na různých vrstvách. Vícevrstvou architekturu můžeme dělit na:

- Monolitickou
- Dvouvrstvou
- Třívrstvou

1.6.1.1 Monolitická aplikace

Jak už název napovídá, jedná se o architektura, která pracuje pouze s jednou vrstvou. V této vrstvě se pak odehrávají všechny procesy aplikace. Tento návrhový vzor je vhodný pro velmi jednoduché aplikace. Její výhodou je rychlý počáteční vývoj, naopak nevýhodou je, že taková aplikace je z hlediska dlouhodobého vývoje špatně udržovatelná.

1.6.1.2 Dvouvrstvá architektura

Dvouvrstvá architektura pracuje se dvěma vrstvami. Jedná se o vrstvu prezentační a datovou. Vhodná pro CRUD aplikace. Tedy aplikace pracující se čtyřmi základními operacemi – vytvoření, čtení, úprava a smazání. Často se tento návrhový vzor používá u API serverů. Aplikace se pak vyznačují tím, že komunikují s další aplikací, které vrací daty v neformátované podobě.

1.6.1.3 Třívrstvá architektura

Nejrozšířenější vícevrstvou architekturou je třívrstvá architektura. Jedná se o architekturu skládající se z vrstvy prezentační, business a datové. Tento typ architektury můžeme dále dělit na striktní a relaxovanou. Rozdíl mezi nimi je ten, že zatímco u striktní třívrstvé architektury je každá vrstva vázána pouze na nejbližší vrstvu, u relaxované třívrstvé architektury mohou být vrstvy vázány i na větší vzdálenost, např. vrstva prezentační s vrstvou datovou.

Prezentační vrstva obsahuje HTML stránky, navigaci mezi stránkami nebo třídy formátující výstupy pro uživatele. Tedy je zodpovědná za zobrazení dat uživateli.

Business vrstva zajišťuje veškeré procesy aplikace, validaci a zpracování požadavků. Obsahuje veškerou business logiku aplikace.

Datová vrstva odděluje aplikaci od databáze. Obsahuje třídy a funkce, které zajišťují komunikaci s databází.

Mezi nejpoužívanější model třívrstvé architektury patří model Model-View-Controller (MVC). Komunikaci s databází nám zajišťuje vrstva *model*, který obsahuje informace o typu dat v databázi. Vrstva *view* je nejvyšší vrstvou a zajišťuje správné vyobrazení dat uživateli. Komunikaci mezi vrstvou *model* a *view* zajišťuje vrstva *controller*, která pracuje s různými pravidly a metodami pro transformaci dat, právě mezi dvěma zmíněnými vrstvami. MVC model je v současné době rozšířen především mezi webovými aplikacemi.

Největší výhodou vícevrstvé architektury je tedy jednoznačně rozdělení jednotlivých úloh do různých vrstev aplikace. Díky tomu je aplikace:

- udržovatelná,
- testovatelná

- a můžeme snadno provádět změny v kterékoliv vrstvě bez ovlivnění dalších vrstev.

Úskalí:

- Díky rozdělení do vrstev se aplikace může stát nepřehlednou a bude zapotřebí se podrobně seznámit s každým modulem zvlášť.
- U větších aplikací může být problém s jasným rozdělením rolí jednotlivých komponent a jejich zařazením do správné vrstvy.

Vhodné pro:

- rychlou tvorbu nové aplikace
- podnikové či obchodní aplikace
- aplikace vyžadující testování

1.6.2 Událostmi řízená architektura

Událostmi řízená architektura funguje na základě centrální jednotky (fronty), která přijímá všechna data a poté je předává jednotlivým modulům, které tato data zpracovávají. O tomto předávání se mluví jako o vytvoření „události“ (Event) a tato událost je tedy dále předána ke zpracování.

Například při psaní webové aplikace v jazyce JavaScript, vznikají drobné komponenty, které reagují na různé události, jako je kliknutí myší nebo stisknutí klávesy. Veškerý vstup zajišťuje prohlížeč a zároveň se stará o to, aby aktuální událost, se kterou pracuje, byla zpracovávána odpovídající částí kódu. V prohlížeči se vyskytuje mnoho různých událostí, ale jednotlivé komponenty reagují pouze na události, které se jich týkají. To je hlavní rozdíl oproti vícevrstvé architektuře, kde všechna data obvykle procházejí všemi vrstvami.

Výhody:

- Snadno přizpůsobitelná složitému prostředí.
- Snadno udržovatelná.
- Snadno rozšiřitelná při vzniku nové události.

Úskalí:

- Pokud se komponenty vzájemně ovlivňují, může být problém s jejich testováním.
- Vývoj systémově datové struktury může být složitý, pokud mají jednotlivé události velmi odlišné úlohy (potřeby).

Vhodné pro:

- aplikace, kde data spolupracují pouze s několika málo komponentami
- uživatelská rozhraní

1.6.3 Architektura mikrojádra

Mnoho aplikací má sadu funkcí, které se používají neustále dokola, s různými daty a při různých úkolech. Kupříkladu jakékoliv vývojové prostředí má základní funkce jako je otevření souboru, komentování souboru, editace souboru nebo spuštění procesu na pozadí. Následně můžeme v aplikaci provádět například kompilaci a spuštění samotného kódu.

V tomto případě jsou úlohy typu zobrazení souboru a jeho editace součástí mikrojádra. Ve vývojovém prostředí však můžeme najít funkce nebo si doinstalovat funkce, které již nejsou součástí „základních funkcí“, tedy leží ve vrstvě nad mikrojádrem. Tyto funkce nazýváme pluginy (zásuvné moduly).

Pokud chceme ve svém projektu využít architekturu mikrojádra, musíme nejprve stanovit, které funkce budou součástí mikrojádra a které budou ve formě pluginu. Funkce mikrojádra jsou funkce, jenž jsou využívány každou chvíli, nejsou ovlivněny daty ani typem úloh. Pluginy jsou naproti tomu funkce, které rozšiřují aplikaci o nové fuknce, které však nejsou využívány příliš často nebo mají specifické požadavky a pravidla.

Úskalí:

- Někdy je těžké rozlišit, kterou z funkcí zařadit do mikrojádra.
- Úprava mikrojádra může být velmi náročná nebo dokonce nemožná, pokud na něm závisí množství zásuvných modulů. Jedinou možností je následně upravit i samotné moduly.

Vhodné pro:

- nástroje, jenž jsou využívány širokou skupinou lidí

1.6.4 Architektura mikroslužeb

Architektura mikroslužeb má pomoci vývojářům k tomu, aby z jejich aplikace nevznikl nepraktický, monolitický a dlouhodobě neudržitelný výtvor. Namísto vytváření jednoho velkého programu je cílem vytvořit spoustu malých a jednoduchých programů (služeb), které následně fungují jako celek. V případě, že je potřeba aplikaci rozšířit o novou funkcionalitu, stačí naprogramovat nový program a přidat jej ke stávajícím.

Jedná se přístup, který je podobný architektuře řízené událostmi a architektuře mikrojádra. Přístup mikroslužeb se však používá hlavně tehdy, když

jsou jednotlivé úkoly snadno oddělitelné.

Úskalí:

- Služby (programy) musí být nezávislé, mohlo by dojít k nekonzistenci serveru.
- Ne všechny aplikace mají úlohy určené tak, že je lze od sebe jednoznačně oddělit.
- Při velkém počtu služeb může docházet ke snížení výkonu, kvůli komunikaci mezi jednotlivými službami.
- Může dojít ke zmatení uživatele, pokud se různé služby vykreslují s různým časovým odstupem.

Vhodné pro:

- webové stránky s malým počtem komponent
- rychlý vývoj nových webových aplikací

1.6.5 Prostorová architektura

Mnoho webových stránek je postaveno na databázi a fungují skvěle, do té doby, dokud databáze zvládá zátěž, která je na ni kladena. Když však dojde k přetížení, celý web spadne. Prostorová architektura je určena k tomu, aby se předešlo takovému přetížení. Odstraňuje centrální databázi a nahrazuji ji sdílenou pamětí.

Data jsou rozdělena do dílčích objektů (uzlů), stejně jako odpovědnost za jejich volání. Objekty jsou uloženy v datové mřížce, která je součástí jednotky, která vykonává určitou činnost. Jednotka může svoji datovou mřížku poskytnout jiné datové jednotce, tím je zajištěn přenos informací napříč aplikací. Může být vhodným řešením v případě, kde i přes zvyšování hardwarových prostředků systému neustále dochází k přetížení serveru.

Úskalí:

- Může být problém testovat systém na celkovou zátěž, avšak je možné testovat jednotlivé jednotky.
- Při práci se sdílenou pamětí může být problém s transakcemi.

Vhodné pro:

- aplikace pracující s vysokým objemem dat

Kapitola 2

Návrh

Na základě provedené analýzy, kterou se zabývala kapitola 1, byl autorem práce proveden návrh na výběr technologií, vybrán vhodný architektonický vzor a navržen design výsledné aplikace.

2.1 Technologie

Jak již bylo uvedeno v sekci 1.5, při výběru technologií, je třeba dbát na účel, využití a celkovou povahu projektu. Povahu projektu můžeme jednoduše určit na základě uživatelských požadavků. Analýza uživatelských požadavků již byla též provedena autorem práce a jejich kompletní přehled je k dispozici v sekci 1.3.

V případě rozsáhlejších projektů, je běžnou skutečností, že na projektu pracuje více vývojářů a vývoj je nejčastěji rozdělen na front-endovou část a back-endovou část. Z toho pak plyne potřeba zajistit dostatečnou komunikaci mezi vývojáři a předem jasně stanovit technologie, které budou při vývoji použity.

V tomto případě se však na vývoji aplikace podílel pouze autor práce sám a tudíž výběr technologií byl čistě na něm. S ohledem na povahu projektu a uživatelské požadavky, se autor práce rozhodl pro vývoj aplikace s následujícími technologiemi.

2.1.1 Programovací jazyk

Dle [8] patří mezi nejpopulárnější programovací jazyky jazyk Java. Autor práce se rozhodl na základě analýzy technologií, viz sekce 1.5, dále dle vlastní zkušenosti a na základě uživatelských požadavků pro vývoj aplikace právě v jazyce Java. Díky tomu, že je autor práce s tímto jazykem již obeznámen, může být veškerý čas věnován samotnému vývoji a není zapotřebí vyhrazovat si čas pro seznámení se s touto technologií.

2.1.2 Framework

Definice frameworku dle [15] zní: *„Framework je základní struktura systému nebo konceptu.“* (přeloženo dle autora) Dále je pak tato definice ještě rozvedena následujícím způsobem. *„Framework slouží jako určitý základ naší webové stránky. Poskytuje nám základní funkcionalitu, kterou každá webová stránka potřebuje.“* (překlad dle autora)

Díky frameworku se můžeme rychleji dostat k vývoji samotné podstaty naší aplikace a nemusíme se dlouze zabývat základní funkcionalitou. Dále nám framework umožní držet určitý formát kódu, strukturu aplikace a díky tomu bude i pro pozdější vývoj snazší našemu kódu porozumět a jednoduše na něj navázat.

Framework bývá vybrán na základě programovacího jazyka, se kterým chceme pracovat. Na základě analýzy v sekci 1.5.2 a též na základě vlastní zkušenosti, bylo autorem práce rozhodnuto, použít při vývoji aplikace framework Spring, konkrétně pak jeho komponenty Spring Web MVC a Spring Boot.

*„**Spring Boot** umožňuje vytváření nezávislých, odolných aplikací, postavených na Springu, které ve výsledku stačí ‚jen spustit‘. Spring Boot bere ohled jak na samotnou platformu Spring, tak i na knihovny třetích stran, díky tomu vývoj probíhá s minimem znepokojení. Většina Spring Boot aplikací potřebuje minimum nastavení týkající se Springu samotného.“* [16]

Spring Boot je založen na principu tzv. mikro služeb. Což vývojářům přináší určité výhody, jako je například jednodušší vývoj, kompatibilita s ostatními službami, minimum nastavení nebo urychlení vývoje. Cílem samotného modulu je pak:

- vyhnout se XML nastavení v rámci Springu
- urychlit vývoj aplikace
- vytvářet nezávislou aplikaci
- nabídnout snazší cestu pro vývoj nových aplikací

A právě z těchto cílů, plynou hlavní výhody modulu Spring Boot:

- nabízí flexibilní možnost konfigurace Java Beans, XML konfiguraci nebo jednoduchou práci s databází
- spousta z výše zmíněných nastavení je již před připraveno, tudíž není potřeba žádná další speciální konfigurace
- nabízí možnost tzv. anotací
- usnadňuje práci s provázáním jednotlivých částí aplikace a jejich závislostmi

Spring Web MVC, běžně označovaný jen jako Spring MVC, *„je podobně jako ostatní webové frameworky založený na vzoru* front controller*, kde máme hlavní servlet, tzv. DispatcherServlet, který poskytuje sdílený algoritmus pro zpracování požadavků, zatímco tu skutečnou práci deleguje do jednotlivých komponent. Tento model je velmi flexibilní a podporuje různé přístupy.“* [17] Zkratka MVC pak označuje tři vrstvy, se kterými Spring MVC pracuje. Jedná se o:

- Model – ten obsahuje a spravuje data, se kterými aplikace pracuje
- View – zpřístupňuje uživateli konkrétní informace, konkrétní „pohled“
- Controller – obsahuje logiku celé aplikace

DispatcherServlet pak funguje jako tzv. *front controller*, který přijímá veškeré požadavky směřující na aplikaci, tyto požadavky následně předává konkrétnímu controlleru, který je zpracuje a zpět vrací pohled (View) a data (Model) s ním související. Za největší výhody Spring MVC lze považovat:

- možnost oddělení jednotlivých rolí/vrstev
- rozsáhlé možnosti nastavení
- urychlení vývoje aplikace
- znovupoužitelný kód
- jednoduchý způsob testování aplikace
- anotace umožňující přesměrování na konkrétní stránku

2.1.3 Front-end

Jak bylo uvedeno v sekci 1.5.1, mezi základní front-end technologie patří HTML, CSS a JS. Také v rámci tohoto projektu jsou tyto technologie zahrnuty. Konkrétně se jedná o HTML5, preprocesor Less a také čistý JS v kombinaci s knihovnou jQuery. Na základě výběru frameworku Spring, bylo autorem práce rozhodnuto, že dalšími technologiemi spravující front-endovou část aplikace budou JSP a JSTL.

JSP neboli JavaServer Pages jsou textové soubory s koncovkou .jsp, umožňující dynamické vkládání obsahu do statické části stránek. JSP jsou při prvním zobrazení převedeny na servlet. Servlet je nástroj pro obsluhu protokolu typu *request - response* (požadavek - odpověď). Servlet je dále převeden na Java třídu, tedy dojde k propojení servletu a původního JSP souboru na základě URL.

V rámci JSP byly dlouhou dobu používány tzv. skriplety. Skriplety jsou části kódu, Java kódu, vložené kdekoliv mezi HTML kód za pomoci speciální

značky. Díky tomu je možné vytvořit webovou stránku pouze pomocí JavaServer Pages. Tento přístup se však vůbec nedoporučuje, protože zcela popírá koncept vícevrstvé architektury a oddělení logiky od zbytku aplikace.

JSTL neboli JSP Standard Tag Library je knihovna standardizovaných značek, která je jednou z alternativ na místo skripletů v rámci JSP. Rozdělení knihovny je následující:

- Core – zajišťuje práci s proměnnými, práci s URL, kontrolu a správu stránek, cykly, větvení
- XML – práce s XML
- Functions – především funkce pro práci s řetězci
- Internationalization (Formatting) – formátování, lokalizace, práce s časovými údaji
- SQL – možnost práce s SQL (toto opět porušuje koncept vícevrstvé architektury)

2.2 Architektonický vzor

Na základě analýzy v sekci 1.6 a dále pak na základě volby technologií v předešlé sekci 2.1, bylo autorem práce rozhodnuto, že při vývoji bude použita vícevrstvá architektura. Konkrétně pak půjde o třívrstvou architekturu.

2.2.1 Třívrstvá architektura

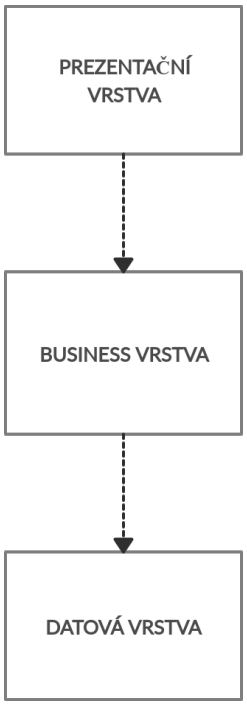
Třívrstvá architektura je vhodná především pro složitější aplikace. Jak už samotný název napovídá skládá se ze tří vrstev:

- Prezentační
- Business
- Datová

Prezentační vrstva se zabývá front-edovou částí aplikace. Tedy především se stará o zobrazení dat uživateli. Obsahuje HTML/JSP stránky a formátuje výstup pro uživatele.

Business vrstva obsahuje veškerou logiku aplikace. Business vrstva by měla být zcela nezávislá na prezentační vrstvě a neměla by být ani závislá na způsobu uložení dat v rámci datové vrstvy. Díky tomu se stane zcela nezávislou na technologických závislostech a můžeme ji kupříkladu přenášet mezi více projekty.

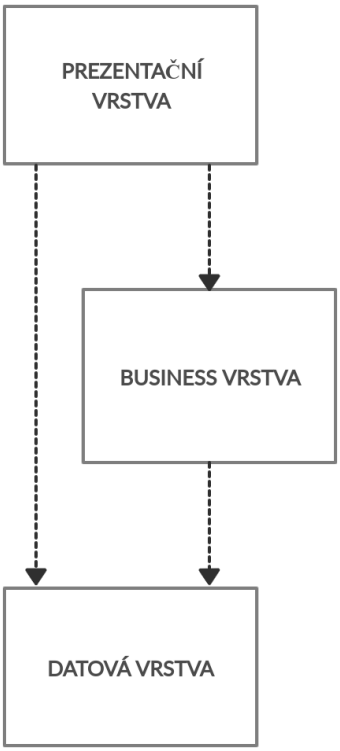
Datová vrstva zajišťuje komunikaci s daty/databází.



Obrázek 2.1: Striktní třívrstvá architektura

Závislost mezi vrstvami je shora dolů. Může se jednat o závislost vždy pouze o úroveň jednu nižší, jak ukazuje obrázek 2.1, tedy prezentační vrstva je závislá na business vrstvě a ta je závislá na datové vrstvě. Mluvíme o tzv. striktní třívrstvé architektuře. Častěji se však používá tzv. relaxovaná třívrstvá architektura. Rozdíl oproti striktní třívrstvé architektuře je v tom, že prezentační vrstva má v tomto případě přímou závislost i na datové vrstvě, jak můžeme vidět na obrázku 2.2. Mezi hlavní výhody třívrstvé architektury patří:

- oddělení jednotlivých částí aplikace
- možnost jednoduše vyměnit jednotlivé vrstvy bez nutnosti zasahovat do dalších vrstev
- usnadňuje testování
- umožňuje mít více prezentačních vrstev
- znovupoužitelnost vrstev



Obrázek 2.2: Relaxovaná třívrstvá architektura

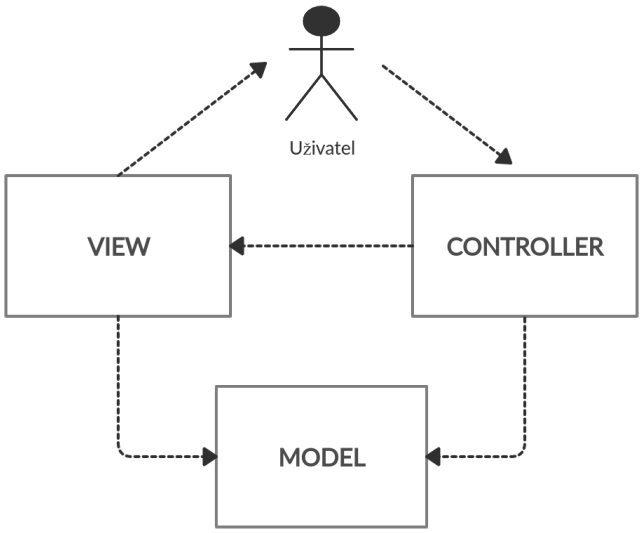
2.2.2 Model MVC

Model/vzor Model-View-Controller řeší rozdělení zodpovědností v rámci prezentační vrstvy. Taktéž řeší některé problémy prezentační vrstvy v rámci vícevrstvé architektury. Smyslem MVC je oddělení business logiky od grafického rozhraní. Model MVC dělí aplikaci do tří částí:

- Model - má na starosti business logiku aplikace
- View - prezentuje data uživateli
- Controller - zodpovědností této části je zpracování uživatelského vstupu

2.2.2.1 Komunikace jednotlivých částí

Controller přijímá požadavky od uživatele, *Controller* tyto požadavky zpracuje a posílá je dále do *Modelu*, který provede potřebnou logiku. *Controller* následně požádá *View* o vykreslení aktualizovaných informací uživateli. *View* si tedy načte aktuální informace z *Modelu* a tyto informace poskytne uživateli.



Obrázek 2.3: Pasivní MVC model

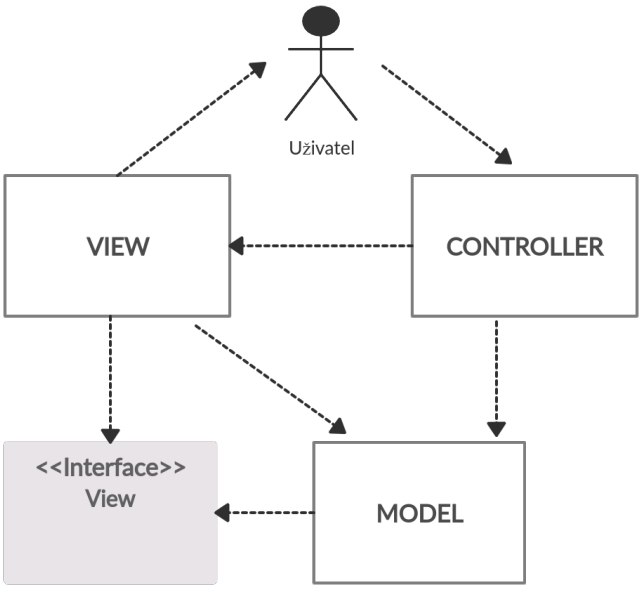
V tomto případě se jedná o tzv. pasivní přístup *Modelu*. *Model* pouze reaguje na požadavky. Toto též popisuje obrázek 2.3.

Druhou variantou MVC modelu je aktivní přístup *Modelu*. Tento koncept je vhodný ve chvíli, kdy potřebujeme uživateli zobrazit aktualizovaná data, aniž by o to sám uživatel požádal. V té chvíli by měl *Model* upozornit *View*, že došlo k nějaké změně a *View* následně aktualizovaná data vykreslit. Abychom tohoto docílili museli bychom obrátit závislost, tedy od *Modelu* směrem k *View*. Avšak tím bychom porušili jedno z pravidel MVC modelu, že *Model* nemá být závislý na *View* ani na *Controlleru*.

Řešením je přidat nějaké rozhraní (interface), kterému v případě jakékoliv změny *Model* oznámí, že ke změně došlo. Následně stačí, aby *View* implementovalo toto rozhraní a díky tomu můžeme uživateli aktualizovat i v případě, kdy si o to sám uživatel nežádal. Tento aktivní přístup *Modelu* popisuje obrázek 2.4.

2.2.3 Vztah třívrstvé architektury a MVC modelu

Vztah MVC modelu a třívrstvé architektury představuje obrázek 2.5. Část *View* zobrazuje data uživateli, *Controller* přijímá požadavky od uživatele, tedy obě tyto komponenty se zabývají front-endovou částí aplikace a tudíž budou patřit do prezentační vrstvy. Z hlediska třívrstvé architektury, prezentační vrstva komunikuje z business vrstvou. V modelu MVC část *View* a *Controller* komunikují s komponentou *Model*. Tedy *Model* řadíme do business vrstvy. S poslední, datovou vrstvou, pak komunikuje pouze *Model*. Z pohledu modelu



Obrázek 2.4: Aktivní MVC model

MVC do ní neřadíme žádnou z jeho komponent.

2.2.4 Vztah třívrstvé architektury a MVC modelu k aplikaci

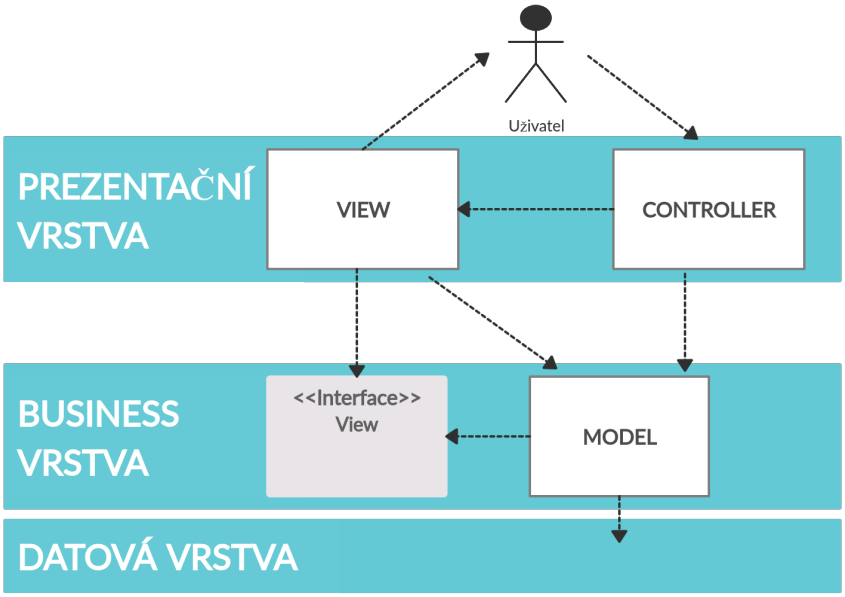
Aplikace je rozdělena do tří vrstev podle navrhované architektury. S ohledem na výběr webového frameworku též dodržuje zásady modelu MVC.

Tedy *View* a *Controller* jsou součástí prezentační vrstvy. *Model* by měl obsahovat veškerou business logiku aplikace. Ve Spring MVC jsou modely objekty, které definují způsob, jakým jsou data mapována na datové úložiště. *Model* proto určitým způsobem zasahuje též do datové vrstvy a není pouze součástí business vrstvy.

Prezentační vrstva má možnost komunikovat jak s business vrstvou, tak i s datovou vrstvou. Business vrstva je závislá pouze na datové vrstvě. V rámci aplikace je tedy využito relaxované třívrstvé architektury. Vzhledem k tomu, že není zapotřebí informovat uživatele o provedených změnách bez toho, aniž by si o to sám zažádal, model MVC se drží pasivního pojetí.

2.3 Doménový model

„Doménový model vytváří síť vzájemně propojených objektů, kde každý objekt představuje nějakou významnou jednotku, ať už jde o nějakou skupinu nebo jednu položku formuláře. Při tvorbě doménového modelu v rámci aplikace, získáme celou vrstvu objektů, které modelují obchodní logiku, ve které



Obrázek 2.5: Vztah MVC modelu a třívrstvé architektury

se pohybujeme.“ [18] (překlad dle autora) Díky objektům (entitám) lépe pochopíme povahu dat, vzájemnou provázanost a spolupráci. Za entity volíme objekty z reálného světa, předměty nebo jednoduše vycházíme z modelu Případu užití. Cílem doménového modelu je tedy:

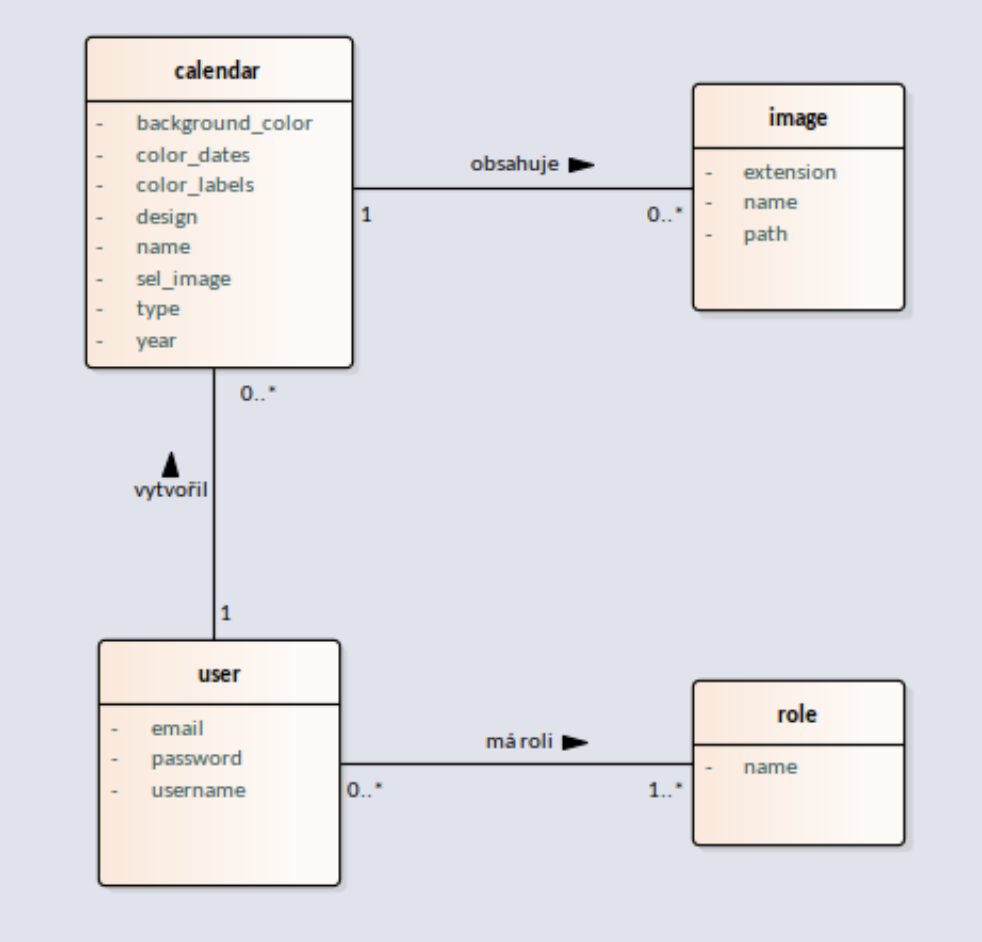
- popis dat,
- popis vztahů mezi entitami,
- popis atributů.

2.3.1 Doménový model pro tuto aplikaci

Na obrázku 2.6 vidíme výsledný doménový model aplikace. Doménový model obsahuje čtyři entity, které byly vybrány na základě modelu Případů užití (obrázek 1.1). Nyní si jednotlivé entity popíšeme, včetně jejich atributů a vztahů mezi nimi.

2.3.1.1 Entita calendar

Entita *calendar* představuje jeden konkrétní návrh kalendáře. Mezi atributy této entity patří:



Obrázek 2.6: Doménový model

- *background_color*: barva pozadí výsledného návrhu,
- *color_dates*: barva data,
- *color_labels*: barva popisků kalendáře – dny v týdnu, měsíc, nadpis na úvodní straně,
- *design*: volba designu,
- *name*: název kalendáře,
- *sel_image*: seznam odkazů na vybrané obrázky,
- *type*: typ kalendáře z hlediska rozložení a orientace,
- *year*: rok kalendáře.

Entita je provázána s objektem *image*, kdy jeden návrh kalendáře obsahuje od nuly až do maxima nahraných obrázků. Dále je také ve vztahu k entitě *user*, kde návrh kalendáře je vytvořen a spravován právě jedním uživatelem.

2.3.1.2 Entita image

Objekt *image* představuje nahrané obrázky do návrháře. Je tedy vázána na entitu *calendar*, kde jeden konkrétní obrázek je mapován na právě jeden návrh kalendáře. Opačně návrh kalendáře může obsahovat neomezené množství obrázků (fotografií). Entita *image* obsahuje tyto atributy:

- *extension*: přípona obrázku,
- *name*: název obrázku,
- *path*: realtivní cestu k obrázku.

2.3.1.3 Entita user

Entita *user* znázorňuje uživatele. Uživatel je ve vztahu s entitou *role*. Uživatel může mít více rolí, minimálně však jednu, tou je *ROLE_USER*. K entitě *user* je dále vázána entita *calendar*. Uživatel může vytvářet návrhy kalendářů, avšak jeden konkrétní návrh je pak vázán na právě jednoho uživatele. Entita *user* obsahuje tyto atributy:

- *email*: e-mail uživatele
- *password*: heslo uživatele
- *username*: uživatelské jméno uživatele

2.3.1.4 Entita role

Entita *role* představuje uživatelské role, díky kterým jsou následně v aplikaci nastavena práva jednotlivých uživatelů. Entita je ve vztahu s entitou *user*, není však podmínkou, že by každá role musela být přiřazena k alespoň jednomu uživateli. Jediným atributem entity *role* je atribut *name*, který představuje jméno role.

2.4 Návrh designu

Předtím, než zahájen samotný vývoj aplikace, je vždy vhodné rozmyslet, jak bude vypadat výsledné grafické uživatelské rozhraní aplikace či webové stránky. Smyslem je určit vhodné rozložení ovládacích prvků, grafickou stránku prvků a zjistit reakci uživatelů na volbu tohoto rozhraní.

Běžným standardem je dodat zadavateli projektu tzv. wireframe. *„Wireframe je grafická kostra webové stránky, která představuje obsah a koncept*

stránky a pomáhá designérům a klientům, diskutovat o detailech tvorby webové stránky.“ [19] (překlad dle autora) Wireframe je tedy jakási skica webu, která obsahuje jednoduché čáry a boxy symbolizující prvky webu. Nejedná se o grafický návrh ve významu volby barev a fontů, ale jde pouze o rozmístění prvků v rámci aplikace nebo její části.

Takovýto návrh můžeme jednoduše vytvořit na obyčejný papír, pomocí grafické aplikace či využít některý z online dostupných nástrojů. Je pak více než vhodné, tento návrh provést v případě, že budeme mít uživatelské rozhraní, které se bude přizpůsobovat dle obrazovky zařízení. Toto též popisuje [20].

Je zcela běžné, že takovýchto návrhů, vznikne před zahájením vývoje nebo i v jeho průběhu celá řada. Plyne to jednoduše z toho, že se na základě návrhu snažíme najít to nejlepší možné řešení pro koncové uživatele. Průběh tvorby wireframu je o opakované tvorbě nových a nových skic, na základě zpětné vazby od klienta, od uživatelů nebo od samotných vývojářů. O tomto se též píše v [21].

2.4.1 Prvotní wireframy aplikace

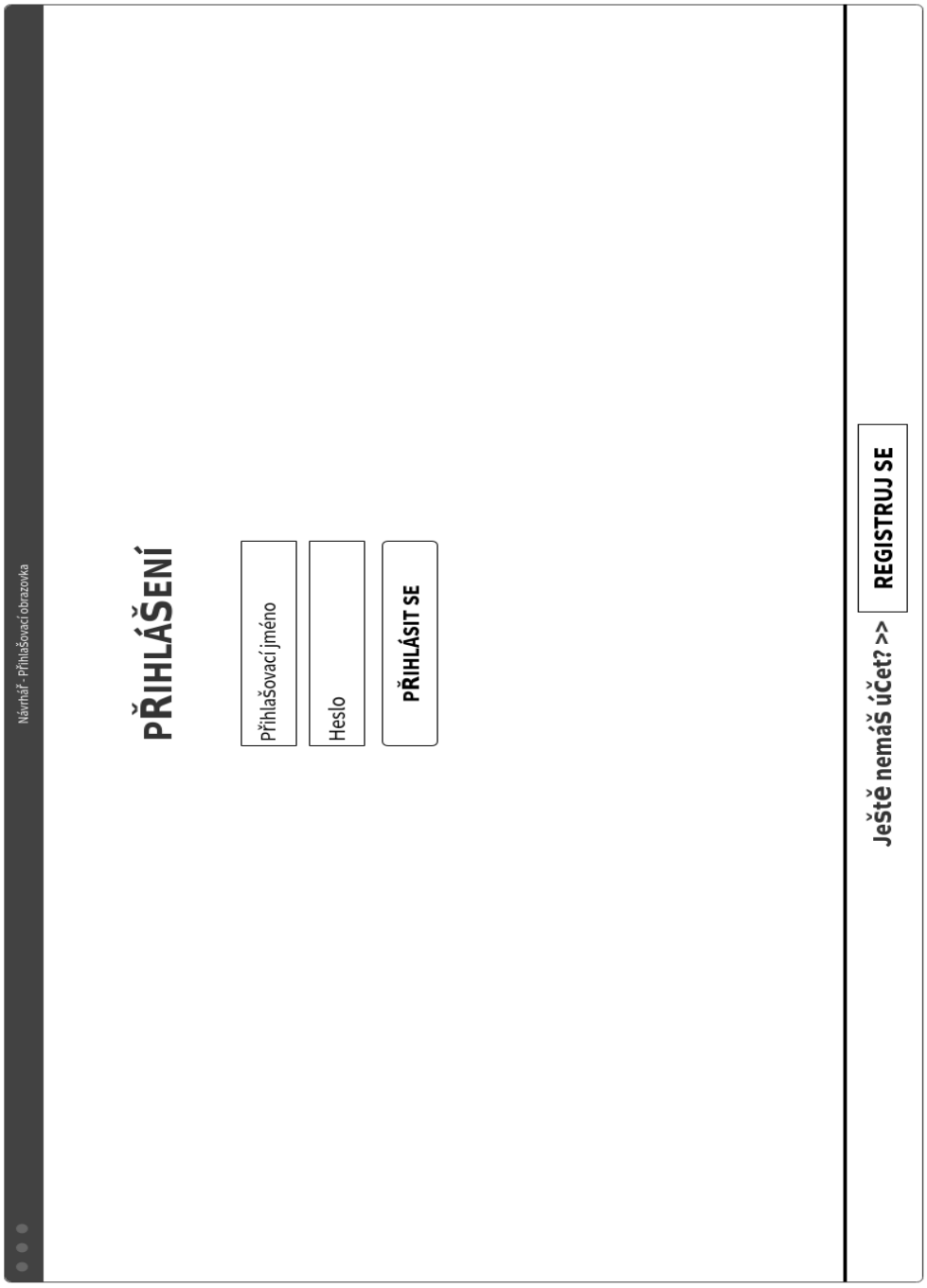
Také autor této práce vytvořil před zahájením vývoje některé wireframy zachycující různé části aplikace. Některé z těchto wireframů se v průběhu vývoje měnily, jiné zůstaly po celou dobu zachovány. Všechny pak byly navrženy pro rozlišení obrazovky 1366×768. Následuje popis a ukázka vybraných wireframů.
Poznámka: S ohledem na množství změn a tedy množství wireframů, které v průběhu práce vznikly, se autor práce rozhodl ukázat pouze prvotní wireframy a k nim dodat slovní popis změn, ke kterým v průběhu vývoje došlo.

2.4.1.1 Úvodní obrazovka

Úvodní obrazovka (obrázek 2.7) zachycuje jednoduchý přihlašovací formulář. Kromě formuláře obsahuje též patičku, která uživateli zobrazuje odkaz na registrační stránku.
Konečná podoba této části: Výsledná úvodní obrazovka byla obohacena o část seznamující uživatele s aplikací a s možností vyzkoušet si aplikaci bez přihlášení. Přihlašovací formulář byl zachován. Došlo tedy k vertikálnímu rozdělení okna na dvě části. Odkaz na registrační stránku byl z patičky přesunut pod přihlašovací formulář a patička byla odstraněna.

2.4.1.2 Obrazovka po přihlášení uživatele s právy administrátora

Tato obrazovka (obrázek 2.8) obsahuje hlavičku, v rámci ní se nachází hlavní navigace aplikace. Dále je pak obrazovka rozdělena vertikálně do dvou částí. Jedna část obsahuje box s přehledem registrovaných uživatelů a možností mezi nimi vyhledávat, druhá část pak zobrazuje box s nejnovějším kalendářem, který přihlášený uživatel vytvořil. Pod tímto rozdělením je pak třetí box, který představuje deset posledních vytvořených kalendářů. Následuje patička, kde



Obrázek 2.7: Wireframe - Úvodní obrazovka

se zobrazuje informace o přihlášeném uživateli, možnost editovat uživatele a odhlásit se z účtu.
Konečná podoba této části: Podoba zůstala z velké části zachována i v rámci výsledné aplikace. Pouze patička byla odstraněna a informace o uživateli byly přesunuty do hlavičky, stejně jako možnost editace a odhlášení.

2.4.1.3 Obrazovka zahájení tvorby kalendáře

Tvorba kalendáře je zahájena jednoduchým formulářem. (obrázek 2.9) Nejprve však i zde nalezneme hlavičku a hlavní navigaci. Následuje již zmíněný formulář. Poté je zde patička, ve které se opět nachází tytéž informace, jako u předešlého wireframu.
Konečná podoba této části: Zde došlo ve výsledné aplikaci ke stejné změně, jako u předchozí obrazovky. Tedy ke zrušení patičky a přesunu informací do hlavičky aplikace. Dále pak došlo k vertikálnímu rozdělení obrazovky. V jedné části zůstal zachován formulář, ve druhé pak přibyl náhled vytvářeného kalendáře.

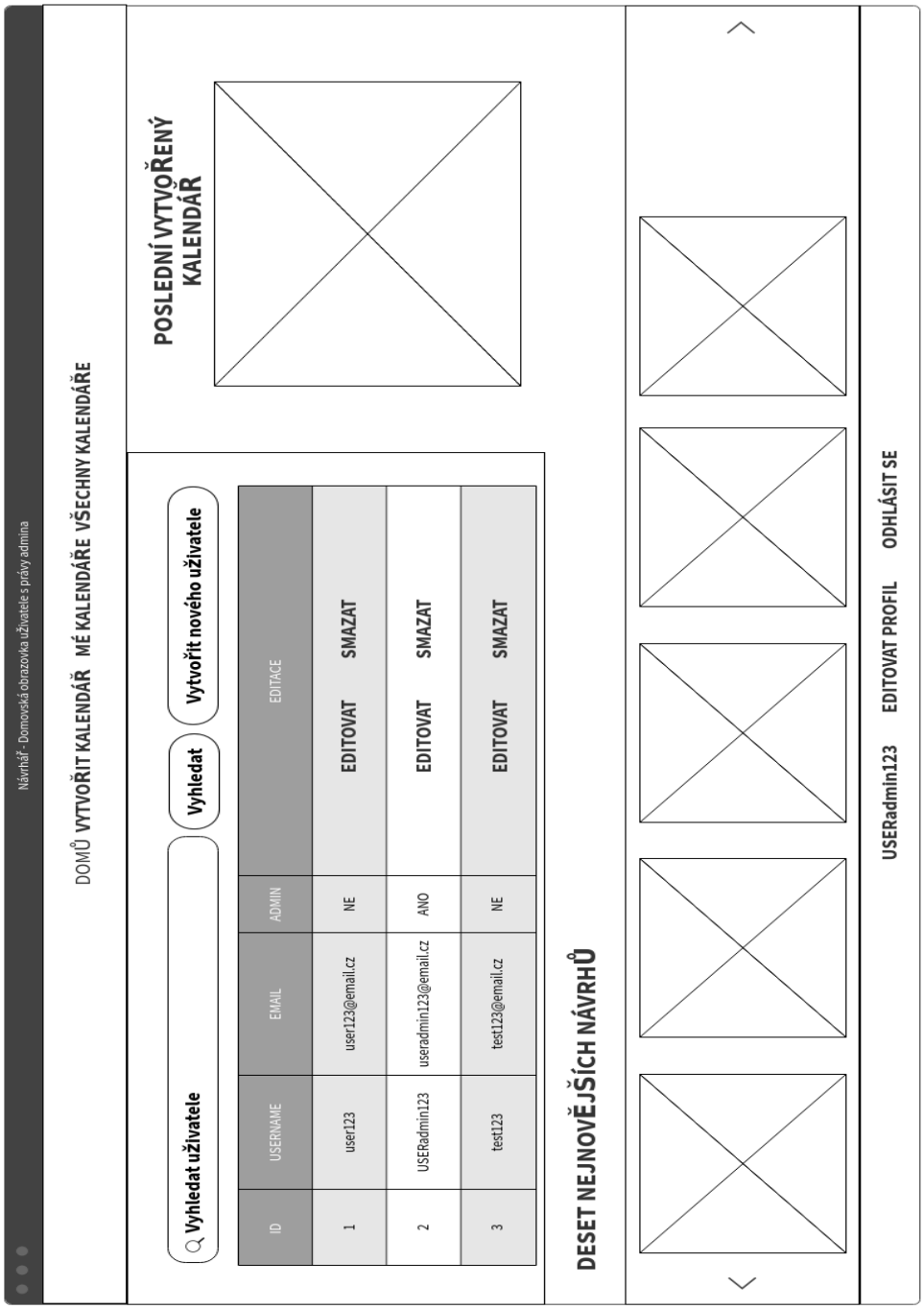
2.4.1.4 Obrazovka pro práci s kalendářem

Srdce celé aplikace pak představuje právě tento wireframe. (Obrázek 2.10) V horní části obrazovky nalezneme hlavičku, kde je na levé straně možnost zobrazení nápovědy a vedle pak hlavní menu. Následuje vertikální rozdělení na dvě části. V levé části se nachází box, který představuje galerii fotek/obrázků, které uživatel následně použije při tvorbě návrhu.

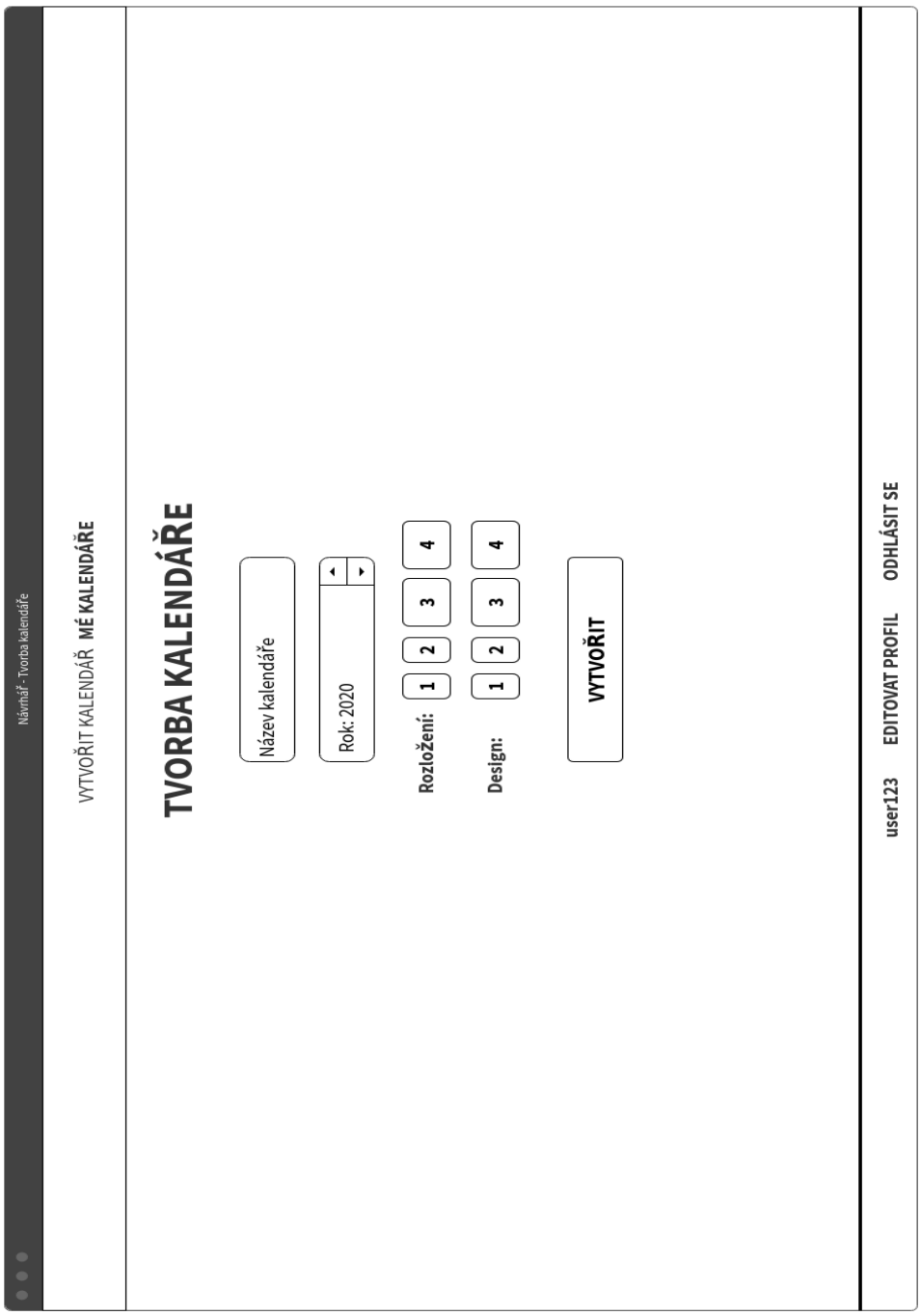
Pravá část je rozdělena horizontálně taktéž na dvě části. V horní části nalezneme formulářová políčka, která byla k vidění i u předcházejícího wireframu. Ve spodní části pak vidíme jednu stranu kalendáře s možností pohybu mezi jednotlivými stranami. Na pravém okraji obrazovky najdeme jednoduchá tlačítka, která usnadňují práci s návrhářem. Spodní část obrazovky zabírá opět patička.
Konečná podoba této části: Právě v této části aplikace bylo v průběhu vývoje rozhodnuto, že dojde ke zrušení patičky v rámci celé aplikace a informace z ní budou přesunuty do boxu, který bude v pravé části hlavičky. Ostatní prvky a rozložení zůstalo zachováno dle prvotního wireframu. Avšak přibyla možnost zobrazit si přehled všech stran kalendáře, což znamenalo vytvoření nového wireframu.

2.4.1.5 Obrazovka s přehledem kalendářů

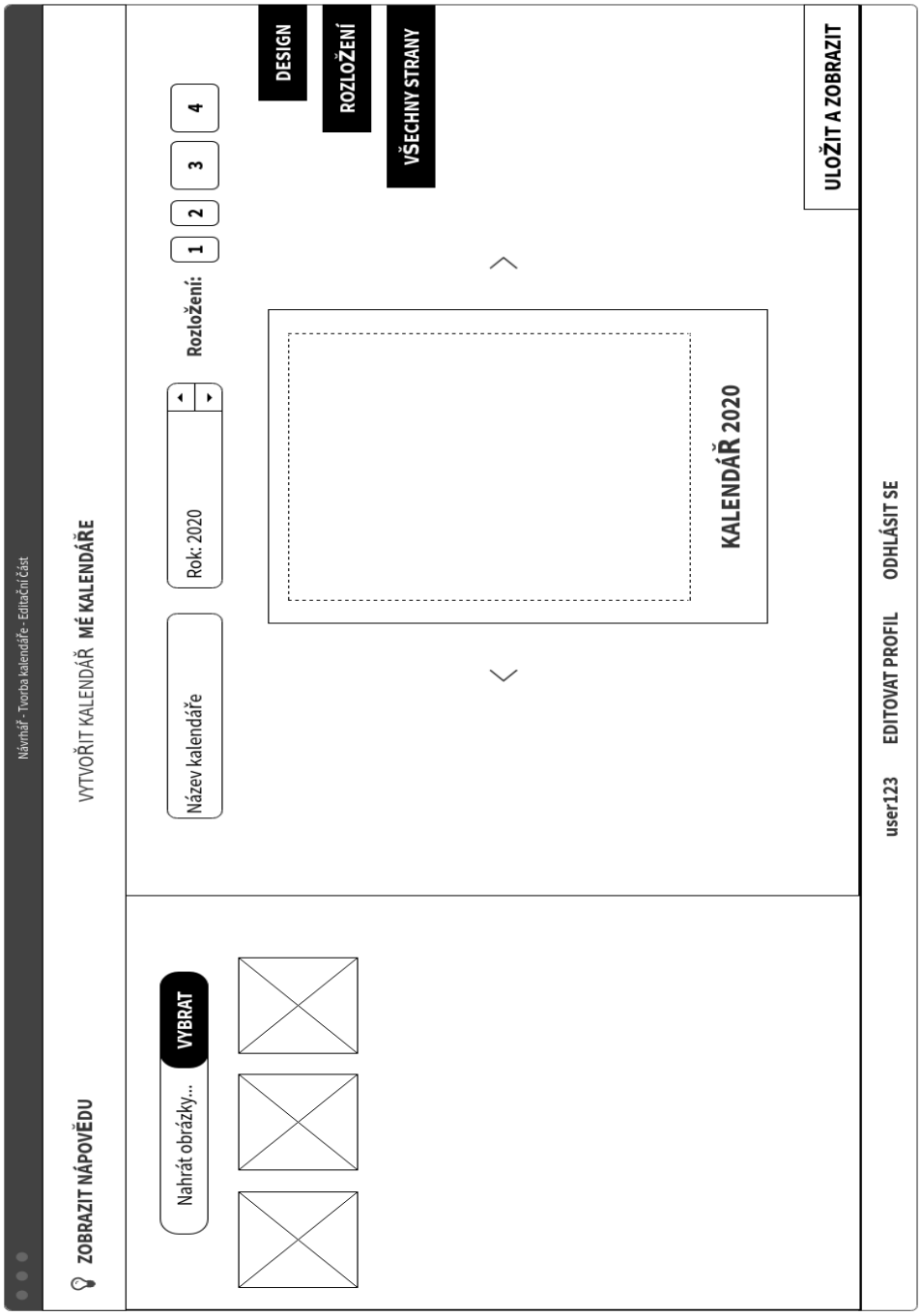
Tato obrazovka (obrázek 2.11) zachycuje jednoduchý box, kde se uživateli zobrazí přehled všech jeho doposud vytvořených a rozpracovaných návrhů. V horní části je i zde menu aplikace a ve spodní části pak patička.
Konečná podoba této části: Výsledná podoba se změnila stejně jako u před-



Obrázek 2.8: Wireframe - Domovská obrazovka uživatele s právy admina



Obrázek 2.9: Wireframe - Tvorba kalendáře - úvodní obrazovka



Obrázek 2.10: Wireframe - Tvorba kalendáře - Editační část

chozích obrazovek přesunutím informací z patičky do hlavičky a zrušením patičky. Zbylé rozložení bylo zachováno.

2.5 Nielsenova heuristika

Obecně heuristická analýza je vhodná pro testování aplikace v případě, kdy aplikaci nemůžeme testovat na koncových uživatelích. Heuristika nám má odhalit problémy s použitelností aplikace. Heuristiku by měl provádět člověk, který v ideálním případě nenavrhoval design. Simuluje tak koncového uživatele. Je doporučeno, aby tuto heuristiku provedlo nejlépe 3-5 lidí. Toto též popisuje [22]. V dnešní době je těchto heuristik hojně využíváno především z důvodu omezených rozpočtů při vývoji aplikace nebo z časových důvodů. Aplikace pak již není dále testována a musí si vystačit s tímto typem testování.

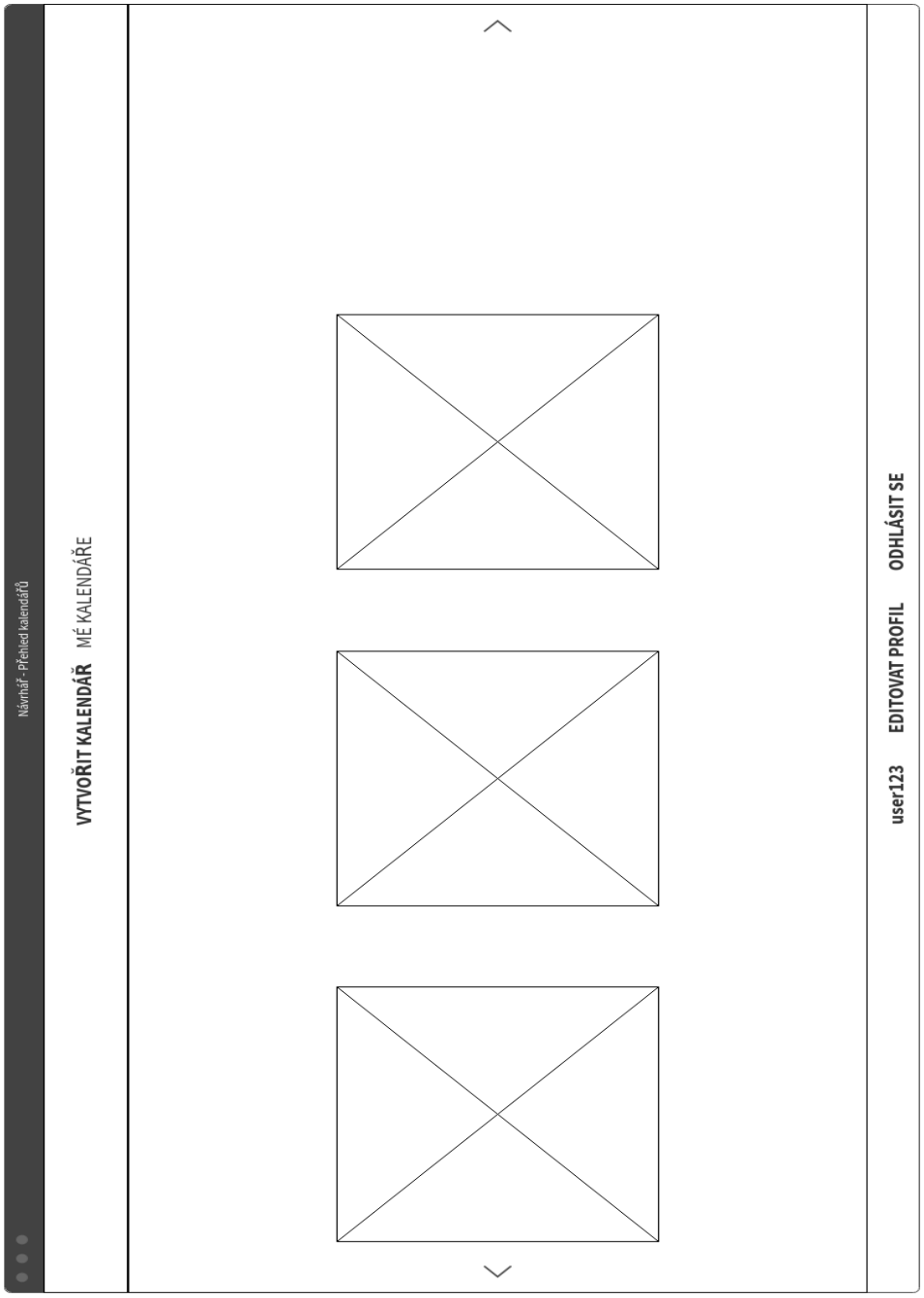
Nielsenova heuristická analýza se skládá z deseti základních pravidel, které nám při jejich dodržení mají zaručit odpovídající použitelnost a intuitivní použití aplikace. Tyto pravidla jsou popsána též v [23].

1. Viditelnost stavu systému – Systém musí vždy uživatele informovat v jakém stavu se nachází. Tedy jestli například čeká na nějaký vstup nebo provádí nějakou operaci.

    - Uživatel je o stavu informován na základě unikátních titulků stránky a dále pak též pomocí odlišení stylu aktivních odkazů v rámci menu.
    - Drobečková navigace, která je s ohledem na toto pravidlo doporučována, byla vzhledem k jednoduchosti aplikace zanedbána.
    - Pokud uživatel čeká na provedení některé operace, konkrétně například na stažení kalendáře nebo načtení návrháře, je o tom informován pomocí krátké zprávy.

2. Propojení systému a reálného světa – Systém má mluvit řečí uživatele a vyhýbat se systémovým pojmům a informacím. Pokud to povaha aplikace umožňuje, měla by se svým obsahem a vzhledem ztotožňovat s reálným světem.

    - Toto aplikace též splňuje. Pokud uživatel například nevyplní správně formulářová pole nebo se snaží dostat do části aplikace, do které nemá s ohledem na svá práva přístup, je o tom informován v běžném jazyku.
    - Aplikace se též snaží zachovat souznění s reálným světem s ohledem na svou povahu, konkrétně tedy pohled na nástěnný kalendář. Kupříkladu v samotném návrháři je zachován poměr stran zmenšeniny, která představuje papír velikosti A4 z reálného světa.



Obrázek 2.11: Wireframe - Přehled kalendářů jednoho uživatele

3. Uživatelská kontrola a svoboda – Uživatelé často spustí některou funkci omylem. V takové situaci je zapotřebí uživateli nabídnout jasně a zřetelně možnost tento stav opustit.
    - V aplikaci jsou funkce, které neumožňují vrátit se po jejich vykonání do původního stavu. V takovém případě je uživatel na tuto skutečnost upozorněn a je mu nabídnuta možnost tuto funkci nevykonat.
    - V žádné chvíli se pak v rámci aplikace nevykonává nějaká funkce samovolně. Kontrola aplikace je čistě na uživateli.
4. Standardizace a konzistence – V rámci aplikace by všechny funkce a činnosti, které vykonávají stejnou věc, měly být označeny stejným způsobem.
    - Tohoto se aplikace snaží plně držet. Příkladem je stejné značení navigace, ať už v rámci menu nebo kdekoliv jinde v aplikaci.
    - V aplikaci se nevyskytují zkratky ani ikony, jejichž význam by uživatel na první pohled neodhalil.
5. Prevence chyb – Lepší než dobrá chybová hlášení, je pečlivý návrh, který chybám zabraňuje.
    - Aplikace uživatele upozorní, pokud špatně vyplní formulářová pole, nebo pokud se snaží dostat do části aplikace, kam nemá přístup.
6. Rozpoznání namísto vzpomínání – Uživatel by měl mít neustále na očích, vše co potřebuje k ovládání aplikace.
    - Stav aplikace zachycují především stylově odlišené odkazy v rámci menu. Menu se v každém stavu aplikace nachází v hlavičce aplikace.
    - Většina částí aplikace je navržena tak, aby všechny prvky aplikace byly viditelné v rámci jedné obrazovky.
    - Části aplikace, do kterých uživatel nemá přístup, jsou skryta.
7. Flexibilní a efektivní použití – Zviditelnit nejpoužívanější prvky aplikace.
    - Jak již bylo uvedeno výše, aplikace je navržena tak, aby ve většině případů zobrazovala všechny prvky v rámci jedné obrazovky. V případě, kdy by došlo k výraznému rozšíření aplikace, by toto mohl být problém z důvodu přehlednosti.
    - V budoucnu je možnost rozšířit aplikaci o klávesové zkratky.
8. Estetický a minimalistický – Dialogy by neměly zobrazovat informace, které jsou irelevantní nebo nepotřebné.

- Toto není v případě výsledné aplikace problém, neboť se zde příliš dialogů nevyskytuje.

9. Pomoc uživatelů pochopit, poznat a vzpamatovat se z chyb – Chybová hlášení by měla být přirozeném jazyce a nikoliv v kódu.

   - Toto je v aplikaci plně dodrženo. Chybová hlášení jsou napsána v přirozeném jazyce, stejně tak důvody proč k chybě došlo, případně i možnost, jak chybu napravit.
   - Především se však aplikace snaží jakýmkoliv chybám předcházet.

10. Nápověda a návody – Přestože je lepší navrhnout systém způsobem, aby nebylo nápovědy zapotřebí, v mnohých případech je lepší nápovědu dodat.

    - Výsledná aplikace má nápovědu, která rychle a jednoduše uživatele seznámí s prostředím návrháře. Tato nápověda je zde především z důvodu uživatelů, kteří aplikaci podobného typu nikdy nepoužívali a práce s ní tak pro ně může být obtížná nebo neintuitivní.
    - Nápověda je lehce dostupná v hlavičce aplikace. V případě, že je uživatel s aplikací obeznámen, lze nápovědu jednoduše skrýt, opět pomocí jednoho kliku v hlavičce.

Kapitola 3

Realizace

V následující kapitola autor popíše několik implementačních detailů, které byly doposud opomíjeny nebo byly zmíněny jen stručně. Dále nás pak blíže seznámí se souborovou strukturou aplikace a řešením některých uživatelských požadavků. Závěrem kapitoly se autor zaměří na bezpečnost, testování aplikace a budoucnost celého projektu.

3.1 Souborová struktura aplikace a implementace

3.1.1 Implementace

Jak již bylo zmíněno v sekci 2.1, aplikace je napsána v jazyce Java, konkrétně jde o verzi 8, při vývoji je použit framework Spring, konkrétně pak jeho moduly Spring Web MVC a Spring Boot. Pro fungování celé aplikace a zajištění komunikace napříč všemi použitými komponentami je využit nástroj Maven. Tomu je věnována samostatná podsekce v této části práce.

3.1.1.1 Best practices

Jak již bylo zmíněno v uživatelském požadavku na podporovatelnost (viz. sekce 1.3.1.2, vývoj aplikace by se měl držet alespoň základních programovacích best practices (osvědčených postupů). Během vývoje se autor práce držel těchto základních pouček, popsaných též zde [24]:

- Udržovat konzistentní odsazení.
- Dodržovat DRY princip – Neopakovat svůj vlastní kód.
- Vyhýbat se přílišnému zanoření kódu.
- Vhodně zalamovat řádky kódu, aby nebyly příliš dlouhé.
- Udržovat vhodnou strukturu projektu a rozdělovat kód do více souborů.

- Dodržovat jmenné konvence.
- Psát, co nejjednodušší kód.

3.1.1.2 Maven

Pro sestavení aplikace byl použit nástroj Maven. *„Apache Maven je softwarový nástroj pro spravování projektu s rozsáhlými možnostmi. Je založen na konceptu ‚Project Object Model' (POM), díky tomu může spravovat sestavování aplikace, report a dokumentaci a to vše na základě jediného souboru s potřebnými informacemi.“* [25]

Project Object Model (POM) je základní složkou Mavenu. Jedná se o XML soubor, který obsahuje nastavení a informace o projektu, díky kterým je pak Maven schopen aplikaci sestavit. Soubor bývá umístěn v kořenové složce aplikace. Do souboru se též umisťují pluginy a knihovny, se kterými chceme následně v aplikaci pracovat. Maven sám pak zajistí dodržení všech potřebných závislostí a instalaci potřebných balíčků.

V případě, kdy chceme umístit naši aplikaci na aplikační server, je zapotřebí vytvořit instalační soubor s příponou war nebo jar. Toto též Maven jediným příkazem dokáže.

3.1.2 Souborová struktura aplikace

Souborová struktura aplikace vychází z tohoto článku [26]. Kompletní adresářovou strukturu vidíme na obrázku 3.1. Dále si blíže popíšeme některé části aplikace.

3.1.2.1 Adresář controller

Tato složka obsahuje controllery. Jak již bylo zmíněno v sekci 2.2.2, controller zpracovává uživatelský vstup. V aplikaci jsou konkrétně tři controllery:

- CalendarController – Zpracovává vše ohledně kalendáře, například vykreslení konkrétního kalendáře, všech kalendářů nebo aktualizaci informací o kalendáři.
- ImageController – Řeší pouze smazání nahraného obrázku do návrháře.
- UserController – Zabývá se uživateli, například zobrazení přihlašovacího formuláře, registračního formuláře nebo zobrazení všech registrovaných uživatelů.

3.1.2.2 Adresář model

„Ve Spring MVC model pracuje jako kontejner, který obsahuje data aplikace. Data mohou být v jakékoliv podobě, kupříkladu objekty, řetězce, informace

z databáze atd.“ [27] V případě výsledné aplikace se jedná o tzv. entity, kde každá entita představuje jednu tabulku v databázi. Její instance jsou pak řádky tabulky. V aplikace nalezneme čtyři takovéto entity:

- Calendar – Představuje v databázi tabulku kalendáře, včetně jeho vlastností, kterými jsou id, název, rok, typ, design, barva popisků, barva data, barva pozadí a vybrané obrázky.
- Image – Představuje v databázi tabulku obrázku. Má čtyři vlastnosti: id, název, cestu a příponu.
- Role – Představuje v databázi tabulku role. Role má pouze id a název.
- User – Představuje v databázi tabulku uživatele. Uživatel má stejně jako kalendářčtyři vlasnosti: id, uživatelské jméno, e-mail a heslo. Dále však v tomto modelu nalezneme položku nové heslo, staré heslo a potvrzovací heslo, tyto položky však nejsou zapisovány do databáze, slouží pouze pro práci s ověřením hesla.

3.1.2.3 Adresář repository

Adresář repository obsahuje rozhraní, která zajišťují přímou práci s databází. V případě výsledné aplikace se jedná pouze o vyhledávání v rámci jednotlivých tabulek. Adresář obsahuje:

- CalendarRepository
- ImageRepository
- RoleRepository
- UserRepository

Přestože se v aplikaci pracuje pouze s rozhraním UserRepository, byla vytvořena i zbylá rozhraní pro případ rozšíření aplikace.

3.1.3 Adresář service

Tento adresář obsahuje soubory, které řeší business logiku aplikace. Nalezneme zde rozhraní a implementaci jednotlivých rozhraní pro různé části aplikace. Je zde logika pro práci s kalendářem, s obrázky, s uživateli nebo třeba logika pro automatické přihlašování na základě tokenu. Jedná se o tyto konkrétní soubory:

- CalendarService
- CalendarServiceImpl
- ImageService

- ImageServiceImpl
- SecurityService
- SecurityServiceImpl
- UserDetailService
- UserDetailServiceImpl
- UserService
- UserServiceImpl

3.1.3.1 Adresář validator

Adresář validator obsahuje soubory se zdrojovými kódy, které se starají o validaci různých dat. Nalezneme zde validaci pro vytváření kalendáře, pro tvorbu nového uživatele, aktualizaci hesla nebo nahrávání obrázku do návrháře. Jmenovitě jde o tyto validátory:

- CalendarCreateValidator
- NewPasswordValidator
- OldPasswordValidator
- UpdateUserValidator
- UploadImageValidator
- UserValidator

3.1.3.2 Adresář resources

Tento adresář obsahuje soubor application.properties, díky kterému lze jednoduše nastavit spojení s databází. Zde je plně využito výhody frameworku Spring Boot, který na základě tohoto souboru a nastavení v něm, sám zajistí spojení s databází. Ve chvíli, kdy je potřeba vyměnit databázi, stačí upravit údaje pouze v tomto souboru a Spring Boot zařídí vše ostatní.

Dalším souborem, který zde nalezneme, je soubor validation.properties. Tento soubor obsahuje zprávy pro uživatele, v případě chybné validace dat.

Soubor data.sql pak slouží k importu dat do databáze, při prvotním spuštění aplikace. Jediná data, která jsou do nově vytvořené databáze vložena, jsou data pro role. Role pro běžného uživatele a role pro uživatele s právy administrátora.

3.1.3.3 Adresář webapp

Složka webapp obsahuje soubory pro práci s front-endovou částí aplikace. Nalezneme zde několik podadresářů:

- css – Tento podadresář a soubory v něm jsou generovány na základě překladu z jazyka Less.
- img – Obrázky použité v rámci aplikace.
- js – JavaScriptové soubory.
- less – Zdrojové kódy popisující vzhled aplikace.
- views – JSP templaty pro vykreslení uživatelského rozhraní.

3.2 Stažení kalendáře

Jedním z uživatelských požadavků na aplikaci, bylo umožnit uživateli stažení vytvořeného návrhu ve formátu PDF do svého zařízení. Toto je též zmíněno v sekci 1.3. K vyřešení tohoto požadavku bylo možno přistoupit dvěma způsoby.

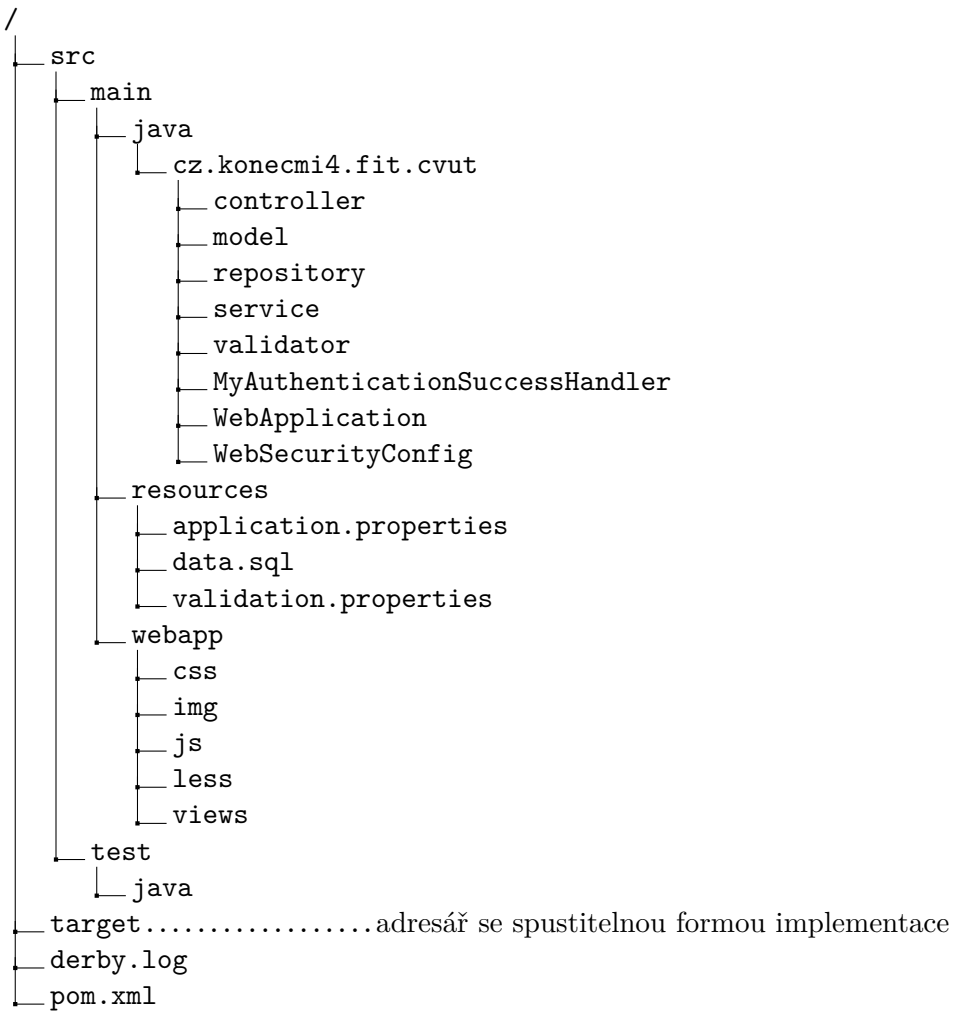
Jedním způsobem bylo vymyslet vlastní řešení, které toto umožní, druhým pak najít volně dostupné funkční řešení, jenž by se následně v rámci výsledné aplikace použilo. Autor práce se rozhodl pro druhou variantu.

Poté, co byla provedena analýza dostupných řešení, bylo třeba ve finále rozhodnout mezi dvěma funkčními řešeními. Prvním z nich bylo řešení postavené na jazyku Java od společnosti iText – pdfHTML (`https://itextpdf.com/en/products/itext-7/pdfhtml`. Druhým pak jednoduchá aplikace s řadou nastavení napsaná v jazyce JavaScript – html2pdf (`https://github.com/eKoopmans/html2pdf.js`, jejímž autorem je Erik Koopmans.

3.2.1 iText - pdfHTML

pdfHTML nabízí spoustu nastavení a využití, jak konvertovat celou stránku či jen její část do PDF souboru. Můžeme vybrat odstavec, nadpis, obrázek nebo třeba patičku stránky a tu následně nechat zkonvertovat. Můžeme nastavit speciální styly, které nám upraví výsledný styl PDF. Styly můžeme vkládat inline způsobem nebo do zvláštního souboru.

Stránku můžeme různě rozdělit, vynutit zalomení stránky nebo její rozložení. Celou dobu pracujeme v jazyce Java, zároveň je možnost tento doplněk (rozšíření) nainstalovat pomocí nástroje Maven. Aplikace je nabízena pod licencí AGPL. Tedy v případě použití této aplikace, je nutno zveřejnit části kódu, kde je aplikace použita.



Obrázek 3.1: Souborová struktura aplikace

3.2.2 Erik Koopmans - html2pdf

html2pdf podobně jako pdfHTML umí konvertovat celé HTML stránky nebo jejich části do výsledného PDF souboru. Aplikace využívá služeb dalších dvou menších aplikací. Konkrétně jde o aplikaci html2canvas, jejímž autorem je Niklas von Hertzen (`https://html2canvas.hertzen.com`) a druhou aplikací je jsPDF, kterou vytvořil James Hall (`https://github.com/MrRio/jsPDF`). html2pdf je nabízeno pod licencí MIT. Jedinou podmínkou této licence je zachovat v programu text licence a jméno autora.

html2pdf je napsána v jazyce JavaScript a její zprovoznění je velmi jednoduché. Je možné využít některého z balíčkovacích nástrojů nebo si stáhnout jeden soubor a ten vložit do naší aplikace, díky čemuž získáme kompletní funkcionalitu aplikace.

Právě tato aplikace byla nakonec vybrána a použita při vývoji výsledné aplikace, a to především z důvodu jednoduché instalace a nastavení.

3.3 Autentizace a autorizace

Mezi další uživatelský požadavek byl zařazen požadavek na autentizaci, který byl dále rozšířen o požadavek na autorizaci i s ohledem na budoucí rozvoj aplikace.

3.3.1 Autentizace

Autentizace je proces, jehož výstupem je určit totožnost uživatele. Uživatel, který byl ověřen a byl mu umožněn přístup do aplikace, nadále podléhá určitým omezením. O tom více v sekci 3.3.2.

Autentizaci na webu bychom mohli přirovnat k mnoha situacím v reálném životě. Jednou z nich je například příchod do budovy velké obchodní společnosti, kde máte naplánovanou schůzku v jedné z jejich kanceláří. Nejdříve je zapotřebí nahlásit svoji přítomnost na recepci. Zde uvedete své jméno – přihlásíte se. Při odchodu se opět zastavíte na recepci a oznámíte svůj odchod, vaše jméno je vyškrtnuto ze seznamu. Po celou dobu, co je vaše jméno vedeno na seznamu, máte právo se pohybovat v budově. Samozřejmě s určitými omezeními. Pokud recepční při vašem příchodu po vás kromě jména požaduje též váš podpis, pak v analogii s aplikací se může jednat o vaše heslo.

Jde o velmi zjednodušený příklad, který má pouze lépe vysvětlit proces autentizace. K procesu autentizace se dá přistupovat mnoha způsoby. Vždy záleží na povaze dat, se kterými aplikace pracuje a samozřejmě na finančních prostředcích, které máme k dispozici.

Jedním z přístupů k autentizaci je jednoduchá autentizace na základě hesla. Uživatel si díky registračnímu formuláři založí v aplikaci účet, kde uvede jedinečné uživatelské jméno nebo e-mail, případně obojí a zvolí si heslo. Na základě těchto údajů pak bude ověřována jeho totožnost při přihlášení do aplikace.

Úskalí tohoto typu autentizace je spolehnutí se pouze na uživatelovo zvolené heslo. Bezpečnostní politika hesel je neustále aktuální téma ve světě autentizace. Mezi deset nejhorších a zároveň často používaných hesel v roce 2019 patří dle společnosti SplashData [28] tato:

1. 123456
2. 123465789
3. qwerty
4. password
5. 1234567

6. 1234678
7. 12345
8. iloveyou
9. 111111
10. 123123

Ze stejného seznamu též vychází článek [29], jehož autorkou je Anita George a kde nalezneme pár rad, jak si zvolit bezpečné heslo. Základním pravidlem je nepoužívat v rámci více účtů, napříč aplikacemi, stejné heslo. Pro každou aplikaci, každý účet, bychom měli používat různá hesla. Dalším předpokladem je používat dostatečně dlouhé heslo, o délce minimálně 8 znaků, kombinovat malá a velká písmena, numerické a speciální znaky. Zároveň by heslo nemělo obsahovat žádné fráze, ani části jména, názvů nebo slovníková slova. Je pochopitelné, že ve chvíli, kdy tato základní pravidla dodržíme, nemusí být snadné si všechna hesla pamatovat. K tomu však v dnešní době slouží tzv. *password manager* (správce hesel), který si tato heslo pamatuje za nás a nám si stačí zapamatovat pouze jedno heslo k tomuto správci.

Dodržet některá z těchto opatření lze uživatele „donutit“, díky správné implementaci aplikace. Výsledná aplikace této práce se drží právě tohoto typu autentizace. Autor práce si uvědomuje výše zmíněná úskalí, a proto je aplikace implementována tak, aby těmto problémům s bezpečností hesel předcházela.

Uživatel je tedy nucen zvolit si heslo o délce minimálně 8 znaků a heslo musí obsahovat kombinaci malých, velkých písmen a numerických znaků. Heslo je následně pomocí Spring Security frameworku, jenž využívá třídu BCryptPasswordEncoder, zahešováno a uloženo do databáze. Bližší specifikace třídy BCryptPasswordEncoder je k dipozici v dokumentaci Spring Security frameworku [30].

Dalším typem autentizace je vícefázová autentizace. Nejčastěji jde o dvoufázovou autentizaci. Jedná se o autentizaci, při které se vychází nejen z ověření totožnosti na základě uživatelského jména a hesla, ale též je zapotřebí součinnost uživatele, který něco konkrétního vlastní. Nejlepším příkladem z reálného světa dvoufázové autentizace je příklad s platební kartou. Uživatel potřebuje k platbě nákupu platební kartu, zároveň však potřebuje znát heslo – PIN.

V případě webových aplikací, konkrétně v bankovnictví, se pak často využívá dvoufázová autentizace způsobem, kdy je po uživateli vyžadováno, zadat kromě standardních přihlašovacích údajů, též kód, který je mu odeslán formou SMS nebo e-mailem.

V poslední době se též rozmáhá trend tzv. biometrické identifikace. Jedná se o identifikaci, kdy je zapotřebí ověřit identitu uživatele na základě jeho biometrických údajů. Nejčastěji se jedná o otisk prstu, rozpoznávání obličeje nebo hlasu, případně skenování oční sítnice.

Dvoufázová orientace má též svá úskalí, avšak jde jednoznačně o bezpečnější variantu oproti jednoduché autentizaci založené pouze na bezpečnosti hesla. Využívá se v případech, kdy je zapotřebí ochránit důležitá data nebo komunikaci mezi uživateli. Příkladem budiž bankovní data nebo přístup k e-mailové komunikaci. Jak již bylo zmíněno dříve a též s ohledem na povahu dat, se kterými výsledná aplikace pracuje, se autor práce rozhodl použít jednoduchou autentizaci.

3.3.2 Autorizace

Proces autorizace jde téměř vždy ruku v ruce s procesem autentizace. Pozměňme nyní dříve zmíněnou zjednodušenou analogii pro lepší vysvětlení procesu autentizace a přibližme díky ní proces autorizace.

Ve velké obchodní společnosti pracuje osoba určitého postavení. Každá osoba, která v této společnosti pracuje, má zaměstnaneckou kartu, díky které má přístup do různých míst a kanceláří v budově, kde společnost sídlí. Každý zaměstnanec má samozřejmě přístup do budovy samotné a též do své vlastní kanceláře. Ředitel společnosti má přístup do každé místnosti ve firmě. Podobně pak třeba vedoucí různých oddělení mají přístup do konferenčních místností, kde se běžní zaměstnanci nedostanou.

Proces autorizace má rozhodnout, zda daný uživatel či stroj, je oprávněn požadovat po aplikaci vykonání určitého požadavku. Toto je nejčastěji rozhodnuto na základě autentizace.

Dále si popíšeme dva základní přístupy autorizace, které jsou podrobně rozebrány zde [31, s. 150-196]. Jde o autorizaci opírající se pouze o účty jednotlivých uživatelů a autorizaci založenou na rolích.

3.3.2.1 Autorizace založená na účtech

Principem této autorizace je umožnit registrovaným uživatelům využívat některé funkce aplikace. Naopak uživatelé, kteří účet založený nemají, jsou o některé funkce ochuzeni.

Jak zmiňoval výše uvedený zjednodušený příklad z reálného světa, každý zaměstnanec má minimálně přístup do budovy, ve které má svou kancelář a do kanceláře samotné. Uživatelé, kteří mají založený účet v aplikaci, mohou přistupovat pouze ke svému účtu. Dále pak mohou svůj účet dle libosti nebo spíše dle možností, která aplikace nabízí, upravovat. Nemají však přístup k účtům jiných uživatelů.

Co když však budeme chtít mít v aplikaci uživatele, který bude mít přístup a možnost spravovat všechny účty ostatních uživatelů. Co když budeme chtít mít uživatele, který bude mít možnost moderovat diskuzi nebo komentářovou sekci v rámci aplikace. K tomu nám již tento způsob autorizace nebude stačit.

3.3.2.2 Autorizace založená na rolích

K tomu abychom mohli mít uživatele s různou úrovní práv, je zapotřebí vytvořit a přiřadit daným uživatelům role. Role určuje operace, které jsou uživatelé vlastnící tuto roli, oprávněni provádět.

Poměrně často nalezneme aplikace, kde jsou použity pouze dva typy rolí. Role označující běžného uživatele. Tato role nemá žádná zvláštní práva a uživatel s touto rolí má přístup pouze ke svému účtu. Druhou rolí je pak administrátorská role. Uživatelé s touto rolí jsou oprávněni provádět veškeré operace a mají neomezený přístup v rámci aplikace.

Systém rolí se též s oblibou používá v případě, kdy máme v rámci aplikace různé typy programu, s různými možnostmi, které jsou uživateli zpřístupněny na základě nějakého poplatku. Poté můžeme mít v rámci aplikaci uživatele s rolí, kteří využívají pouze zdarma dostupný program. Dále kupříkladu uživatele využívající základní program a v posledním případě uživatele, kteří si zaplatí za premium program.

Autor práce při vývoji aplikace pracoval nejprve s autorizací opírající se účty uživatelů. Uživatelé, kteří účet v rámci aplikace neměli vytvořený, měli pouze možnost si návrhář vyzkoušet, avšak výsledný návrh si nemohli stáhnout do PC. S ohledem na možný budoucí vývoj aplikace, se však autor rozhodl pro využití autorizace na základě rolí. Byly však vytvořeny pouze dvě základní role, a to role pro běžného uživatele a administrátorská role. Možnost vyzkoušet si návrhář i pro neregistrované uživatele byla zachována. Ve výsledné aplikaci jsou tedy použity oba výše popsané přístupy autorizace.

3.4 Bezpečnost a testování

3.4.1 Bezpečnost

Ve chvíli, kdy slyšíme o bezpečnosti webových aplikací, si často představíme bankovní aplikace, e-mailové klienty nebo obecně aplikace pracující s citlivými údaji a zajištění bezpečnosti právě těchto citlivých dat. Avšak citlivá data jsou jen špičkou ledovce, co se týká zabezpečení webových aplikací. Mezi deset největších bezpečnostních rizik ve světě webových aplikací patří dle společnosti OWASP tato [32]:

1. Injekce
2. Neošetřená autorizace
3. Zobrazení citlivých dat
4. XML External Entity
5. Chyby v zamezení přístupu

6. Špatná konfigurace zabezpečení
7. Cross-Site Scripting (XSS)
8. Neošetřená deserializace
9. Komponenty se známými zranitelnostmi
10. Nedostatečné logování a monitorování

Společnost OWASP neboli *„Open Web Application Security Project, je otevřená komunita, jejímž cílem je umožnit organizacím vyvíjet, pořizovat a udržovat aplikace a API, které jsou důvěryhodné.“* [33] (překlad dle autora) Seznam deseti největších bezpečnostních hrozeb je aktualizován společností OWASP každé 3-4 roky.

Nyní nás autor práce s jednotlivými riziky blíže seznámí a též nás obeznámí s řešeními ve výsledné aplikaci. Některá z těchto rizik ve výsledné aplikaci řeší samotné frameworky, které byly při vývoji použity.

3.4.1.1 Injekce

Jedním z hlavním cílů útočníka je najít způsob, jak spustit v rámci vaší aplikace svůj vlastní kus kódu. Útočník pak díky tomu může například odcizit vaše data nebo data poškodit a tím vaši aplikaci znehodnotit.

V rámci většiny webových aplikací nalezneme možnost vyhledávání. Tato funkce většinou funguje tak, že přijme vstup od uživatele, vytvoří z něj databázový dotaz a odešle jej do databáze. A právě této situace se snaží zneužít útočník a místo obyčejného dotazu, zadá do vyhledávače vaší aplikace část kódu, který se následně odešle do databáze a tam je spuštěn.

V tomto případě se jedná o SQL injekci. Existují však mnohé další, například XML nebo LDAP injekce, SQL však patří mezi nejrozšířenější.

V rámci výsledné aplikace nejsou použity ručně vytvářené dotazy do databáze. Veškerá práce s databází je řešena skrze Spring Data JPA framework pomocí JPA entit a framework sám veškeré dotazy do databáze ošetřuje.

3.4.1.2 Neošetřená autorizace

Problémy s bezpečností při autorizaci a též při autentizace jsou popsány v sekci 3.3. Ve stejné sekci autor práce popisuje realizaci v rámci výsledné aplikace.

3.4.1.3 Zobrazení citlivých dat

Jak bylo zmíněno v úvodu této kapitoly, při zmínce o bezpečnosti aplikací, se lidem nejčastěji vybaví citlivá data. Citlivá data by měla být uchovávána v nečitelné podobě. Mezi citlivá data patří též uživatelská hesla.

Co se týká výsledné aplikace a citlivých dat, tak tato aplikace pracuje pouze s uživatelskými hesly. Jak již bylo zmíněno v sekci 3.3, hesla jsou před zapsáním do databáze hešována za pomoci bcrypt enkodéru.

3.4.1.4 XML External Entity

Pokud aplikace přijímá uživatelský XML vstup, pak je tento vstup zpracováván skrze XML parser. Pokud je tento parser špatně nastaven, může se díky tomu útočníkovi podařit aplikaci poškodit. Toto též popisuje [34].

S ohledem na to, že výsledná aplikace nepracuje s XML daty, není zapotřebí tuto hrozbu řešit.

3.4.1.5 Chyby v zamezení přístupu

Útočníci rádi zneužívají toho, že aplikace nemají správně naprogramovanou práci s oprávněními. Problém autorizace též popisuje sekce 3.3. Ve stejné sekci autor popisuje, jakým způsobem je autentizace řešena ve výsledné aplikaci.

3.4.1.6 Špatná konfigurace zabezpečení

Můžeme dodržet veškerá dříve zmíněná opatření a přesto se nám může lehce stát, že naše aplikace se stane zranitelnou. A to sice ve chvíli, kdy se nám stane, že při spuštění aplikace do produkčního běhu zapomeneme skrýt některá vnitřní nastavení aplikace. Tento příklad též zmiňuje [35, s. 16]. Též v [35, s. 16] se píše, že *„se nemusí jednat o zranitelnost aplikace v pravém slova smyslu, ale může to útočníkovi velmi ulehčit jeho snahu aplikaci poškodit nebo odhalit opravdovou zranitelnost aplikace.“* (překlad dle autora)

3.4.1.7 Cross-Site Scripting (XSS)

„Cross-site scripting zranitelnosti jsou určitým druhem injekce, kdy se útočník snaží dostat kód se svým skriptem (může jít o javascriptový skript) nebo HTML kód do vaší zranitelné aplikace.“ [35, s. 13] (překlad dle autora) Díky tomu se může útočník dostat k heslům, provádět phishingové útoky a získávat citlivá data.

Tento typ zranitelnosti vzniká pokaždé, když přijímáme nějaký vstup od uživatele a následně mu jej zpátky zobrazujeme. Může jít o data z vyplněného formuláře, o výraz, který uživatel vyhledává nebo o zobrazení komentáře.

3.4.1.8 Neošetřená deserializace

Neošetřená deserializace může opět vést k tomu, že se bude útočník snažit dostat do naší aplikace vlastní kód nebo skript, který by naši aplikaci poškodil.

V rámci výsledné aplikace deserializaci zajišťují použité frameworky. Nebylo zapotřebí toto řešit.

3.4.1.9 Komponenty se známými zranitelnostmi

Je zřejmé, že ve chvíli, kdy v aplikaci použijeme frameworku nebo knihovnu, o které je známo, že se potýká s některým z problémů zranitelnosti, tak tím vystavujeme naši aplikaci potenciálnímu nebezpečí.

O frameworcích a knihovnách, které byly použity při vývoji aplikace, není známo, že by byly zranitelné.

3.4.1.10 Nedostatečné logování a monitorování

Pokud se zanedbá logování a monitorování, může být složité odhalit, že se s naší aplikací děje něco nestandardního. Díky monitorování získáme přehled o fungování naší aplikace a pokud se aplikace začne chovat způsobem, který neodpovídá běžnému chování aplikace, je třeba zkoumat, zda nedošlo k nějakému útoku nebo pokusu o útok.

3.4.2 Testování

Pro testování business logiky jsou doporučovány především jednotkové testy. Jednotkové testy testují izolovaně jednotlivé komponenty. V rámci výsledné aplikace však není žádná vhodná komponenta, pro kterou by se jednotkový test dal použít. Framework Spring Web MVC nabízí vlastní rozšíření pro testování komponent, a to sice Spring MVC Test framework. V případě kdyby byly tyto testy pro testování výsledné aplikace vytvořeny, byly by vloženy do složky src/test/java. Podrobná struktura aplikace byla probrána v sekci 3.1.2.

Namísto testování business logiky se tedy autor aplikace zaměřil na testování uživatelského rozhraní. Kromě již dříve zmíněné Nielsenovy heuristiky v sekci 2.5, se autor rozhodl pro kognitivní průchod.

3.4.3 Kognitivní průchod

„Kognitivní průchod je technika, která se zaměřuje především na snadnost učení aplikace. Může ji provádět jeden hodnotitel nebo skupina hodnotitelů.“ [36, s. 66] Tuto techniku je možné použít kdykoliv během vývoje, případně i na konci vývoje a slouží k tomu, aby vývojář, případně designér, zjistil, zda navržené rozložení prvků je uživatelsky přívětivé a pro uživatele srozumitelné. Výhody této metody jsou: (popsány též v [36, s. 67])

- Metoda nevyžaduje funkční produkt ani reálné uživatele.
- Může být použita kdykoliv během vývoje.
- Poskytuje důležité informace, které mohou být použity pro řešení problémů.
- Má dobře definovaný postup, je založena na otázkách a je zaměřena především na schopnost učení se. (Ve významu práce s aplikací.)

Pro kognitivní průchod byl autorem práce vybrán hodnotitel, který do dané chvíle nepřišel s podobnou aplikací do kontaktu. Autor práce se stal pozorovatelem a vyhodnocoval jednotlivé kroky hodnotitele. Při pozorování autor vyhodnocoval průchod na základě těchto čtyř otázek, vycházející též z [36, s. 73]:

- Podařilo se uživateli dosáhnout správné akce?
- Byl uživatel dostatečně upozorněn na vykonání správné akce?
- Odpovídá akce, kterou se uživatel snaží vykonat, tomu, co od ní uživatel očekává?
- Je-li akce provedena, vidí uživatel odezvu aplikace?

Pokud by na některou z otázek bylo zodpovězeno záporně, je zapotřebí aplikaci vhodně upravit, aby se na otázku dalo odpovědět kladně. Následuje popis a vyhodnocení několika úkolů, které byly při průchodu realizovány, a které měly potvrdit, že kdokoliv bude aplikaci používat, nebude mít problém se v ní orientovat.

3.4.3.1 Registrace uživatele

Uživatel si prvně zobrazí aplikaci a pro její použití je zapotřebí registrace.

Hodnotitel přešel s přihlašovací obrazovky na registrační obrazovku. Neměl problém najít odkaz vedoucí na registrační formulář. Po vyplnění a odeslání formuláře, jej aplikace přesměruje do jeho nově vytvořeného účtu.

Hodnotitel měl poznámku k úvodní přihlašovací obrazovce, kde by ocenil doplňující informaci o aplikaci a více zvýraznil možnost registrace. Do výsledné aplikace bylo toto zohledněno. Původně navržená obrazovka pro přihlašovací formulář byla pozměněna, jak již bylo zmíněno v sekci 2.4.1.1.

3.4.3.2 Zahájení tvorby kalendáře

Uživatel chce začít tvořit nový návrh kalendáře.

Hodnotitel se přihlásil do aplikace. Vzhledem k tomu, že doposud žádný návrh nevytvořil, je mu na úvodní obrazovce po přihlášení ihned zobrazena možnost začít tvořit nový návrh. Stejně tak může začít tvořit návrh pomocí odkazu v hlavním menu. Aplikace následně uživateli zobrazila formulář, skrze který byla zahájena tvorba kalendáře. Poté, co hodnotitel formulář vyplnil a odeslal, mu aplikace zobrazila okno se samotným návrhem a dalšími možnostmi nastavení.

Hodnotitel měl problém se zorientovat a pochopit některé prvky formuláře, který mu byl zobrazen při zahájení tvorby návrhu. Toto bylo autorem práce zohledněno v dalším vývoji aplikace a tato obrazovka byla doplněna o náhled návrhu, na základě toho, jaká nastavení si uživatel ve formuláři vybere. Též popsáno v sekci 2.4.1.3.

3.4.3.3 Nahrání fotografií do návrháře a výběr vlastních barev pro výsledný návrh

Navazuje na předcházejí bod, uživatel si do návrháře nahraje fotografie a zvolí vlastní barvy, které budou použity ve výsledném návrhu.

Hodnotitel neměl problém s nahráním fotografií. Funkce nahrání fotografií je tedy dostatečně viditelná. Problém nastal s vkládáním fotografií do kalendáře. Hodnotitel netušil, jakým způsobem dostat fotografie z „galerie“ do návrhu. Dalším problémem pak bylo nastavení vlastních barev. Hodnotitel nemohl tuto možnost v rámci obrazovky v rozumném čase nalézt.

Autor práce tedy do této části aplikace doplnil jednoduchou nápovědu, která uživatele rychle seznámí s jednotlivými funkcemi. Dle autora je toto vhodné též z pohledu budoucího rozvoje aplikace, kde se předpokládá rozšíření právě této obrazovky o další prvky.

3.4.3.4 Zobrazení již vytvořeného návrhu a jeho stažení

Uživatel si chce prohlédnout vytvořený návrh a stáhnout si jej do PC.

Hodnotitel se přihlásil do aplikace, která mu hned na úvodní obrazovce po přihlášení nabídla seznam všech jeho doposud vytvořených návrhů. Hodnotitel si našel ten, který si chtěl zobrazit a stáhnout. Aplikace mu návrh zobrazila. Hodnotitel si návrh prohlédl a pomocí tlačítka v levé části obrazovky stáhl do PC.

Hodnotitel neměl v tomto případě sebemenší problém. Ocenil dostupnost tlačítka pro stažení návrhu i ve chvíli, kdy měl odskrolovanou obrazovku.

3.5 Budoucnost projektu

Aplikaci je možné dále rozšiřovat a zdokonalovat. Vzhledem k zvolenému architektonickému vzoru není problém pracovat na jakékoliv části aplikace bez nutného zásahu do vrstev ostatních. Na co všechno je možné se do budoucna zaměřit?

V první řadě určitě ještě více propracovat funkce samotného návrháře. Nabídnout uživateli možnost navrhnout si zcela vlastní šablonu. Umožnit uživateli zvolit si vlastní rozložení prvků, volbu vlastního pozadí, fontu písma aj. V této souvislosti by bylo pravděpodobně vhodné, nahradit stávající jQuery knihovnu jinou, modernější JS knihovnou.

Ve chvíli, kdy by uživatelé mohli tvořit vlastní šablony, bylo by určitě zajímavým řešením, aby mohli tyto šablony sdílet s ostatními uživateli. Mít možnost využít šablony jiných uživatelů, mít možnost přehledu o nejpoužívanějších šablonách nebo zavést bodovací systém, na základě kterého, by uživatelé mohli hodnotit vytvořené šablony jiných uživatelů.

S tím, co bylo popsáno výše, pak souvisí rozšíření funkcí uživatelů s administrátorskými právy. Bylo by zapotřebí jim nabídnout kompletní přehled

o všech hodnoceních, statistikách a mít možnost zasáhnout do hodnocení nebo odstranit nevhodný obsah. V případě, kdy by bylo umožněno uživatelům například komentovat vytvořené šablony nebo návrhy samotné, by pak administrátoři měli mít možnost tyto komentáře spravovat. Popřípadě vytvořit novou roli, která by byla určena pro správu komentářové části aplikace.

Dále by bylo vhodné vylepšit registraci uživatelů. Jak bylo zmíněno v předcházející sekci 3.4, momentálně je registrace řešena jednoduchým formulářem, po jehož vyplnění se uživateli nechá vytvořit účet a do databáze je uloženo zašifrované heslo, zvolené při registraci. Avšak vzhledem k nutnosti zadat e-mail, by bylo možné například odesílat registrovanému uživateli e-mail s potvrzovacím odkazem, případně pak s generovaným heslem, aby byla zajištěna ještě větší bezpečnost účtu.

Trendem poslední doby je pak tzv. OpenId autentizace, blíže popsána v [31, s. 134-146]. Jedná se o autentizaci na základě účtu vytvořeného v jiné aplikaci poskytující OpenId. Uživatel si nemusí zakládat účet v dané aplikaci, ale využije toho, že má ověřený účet u některé jiné aplikace, a pomocí tohoto účtu získá přístup do dané aplikace. Mezi poskytovatele OpenId patří například společnost Google.

Bylo by vhodné provést testování na reálných uživatelích. Jak již bylo zmíněno v sekci 3.4, testování bylo prováděno formou Nielsenovi heuristiky a pomocí kognitivního průchodu. Nikdo však neotestuje a neodhalí nedokonalosti aplikace lépe než koncový uživatel. Na základě výsledků tohoto testování pak provést analýzu, jak s těmito informacemi od koncových uživatelů pracovat, jak jich v rámci aplikace zlepšit nebo zcela změnit.

Aplikaci lze též rozvíjet z hlediska designu. Design, který autor práce použil, je velmi jednoduchý a to z toho důvodu, že byl kladen důraz především na funkční stránku aplikace a na intuitivní práci s jednotlivými prvky aplikace.

V neposlední řadě by se dalo pracovat na napojení aplikace na nějakou platební bránu a kupříkladu umožnit uživatelům vybrat si, v jakém e-shopu/společnosti, by si svůj návrh mohli nechat vytisknout. To by nejdříve znamenalo provést analýzu, zda by bylo možné takovou spolupráci s některými e-shopy navázat a následně též zohlednit nové technické problémy a požadavky, které by to do aplikace přineslo.

Závěr

Cílem práce bylo na základě rešerše současného stavu aplikací, navrhnout vhodný architektonický model a naprogramovat webovou aplikaci pro návrh a tisk nástěnných kalendářů. Návrhář měl být vyvíjen takovým způsobem, aby jej bylo možné v budoucno dále rozšiřovat. Zároveň měly být uživatelům připraveny alespoň dvě grafické šablony, dle kterých by si uživatel návrh mohl sestavit. Aplikace měla umožnit uživateli výsledný návrh kalendáře stáhnout ve formátu PDF a velikosti A4. Posledním požadavkem od zadavatele pak bylo umožnit uživatelům registraci v rámci aplikace.

Na základě rešerše autor práce provedl analýzu současných řešení, konkrétně se zaměřil na čtyři různé aplikace, díky čemuž získal přehled o tom, jak by se výsledná aplikace měla chovat a jakým způsobem by bylo vhodné navrhnout přívětivé uživatelské prostředí. V rámci další analýzy se zaměřil na uživatelské požadavky, dostupné technologie a architektonické vzory, které by pro tento typ aplikace mohly být vhodné.

Z této analýzy byl pro vývoj vybrán programovací jazyk Java, framework Spring a vhodné front-endové technologie spolupracující s tímto frameworkem, především JSP a JSTL. Z architektonických vzorů byl vybrán vzor třívrstvé architektury, konkrétně pak relaxační třívrstvá architektura a model MVC. Autor práce se též zaměřil na návrh uživatelského prostředí, zvláště z hlediska rozložení prvků a jejich intuitivnosti.

K vyřešení problému se stažením výsledného návrhu do PC ve formátu PDF, autor využil již funkčního a volně dostupného řešení. Taktéž byl dodržen požadavek týkající se registrace uživatelů. Uživatelé díky registraci získají historii všech svých vytvořených návrhů a též si v rámci svého účtu mohou ukládat rozpracované návrhy a později se k nim vrátit.

Také požadavek na připravení grafických šablon byl splněn. Autor práce připravil uživatelům čtyři různé grafické šablony. Dvě grafické šablony pro kalendář s orientací na výšku a dvě pro kalendář s orientací na šířku. Šablony mají pevně dané pozadí, barvu popisků a barvu data. Uživatelům je též umožněno pracovat s vlastní fantazií a mohou si vybrat barvy, které budou

v jejich návrhu použity.

Díky zvolenému architektonickému vzoru, technologiím a dodržením základních programovacích praktik, je možné tuto aplikaci v budoucnu dále rozšiřovat o novou funkcionalitu nebo jednoduše vylepšovat funkcionalitu stávající.

Bibliografie

1. ARMY, United States Government. *Systems Engineering Fundamentals*. Createspace Independent Pub, 2013. ISBN 1484120833. Dostupné také z: `https://web.archive.org/web/20060211165311/http://www.dau.mil/pubs/pdf/SEFGuide%5C%2001-01.pdf`.
2. GRADY, Robert B.; CASWELL, Deborah L. *Software Metrics: Establishing a Company-wide Program*. USA: Prentice Hall, 1987. ISBN 0138218447.
3. BRUEGGE, B.; DUTOIT, A.H. *Object-oriented software engineering: conquering complex and changing systems*. USA: Prentice Hall, 1999. ISBN 0134897250.
4. Introduction. In: *HTML Living Standard* [online]. 2020 [cit. 2020-04-19]. Dostupné z: `https://html.spec.whatwg.org/#is-this-html5?`.
5. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. In: *The World Wide Web Consortium* [online]. 2011 [cit. 2020-04-19]. Dostupné z: `https://www.w3.org/TR/CSS2/syndata.html#rule-sets`.
6. MDN Web Docs Glossary: Definitions of Web-related terms – CSS preprocessor. In: *MDN web docs* [online]. 2020 [cit. 2020-04-19]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor`.
7. CHAFFER, Jonathan; SWEDBERG, Karl. *Learning JQuery: Better Interaction Design and Web Development with Simple JavaScript Techniques*. Birmingham, Velká Británie: Packt Publishing, 2007. ISBN 1847192505.
8. TIOBE Index for April 2020. In: *TIOBE - The Software Quality Company* [online]. 2020 [cit. 2020-04-21]. Dostupné z: `https://www.tiobe.com/tiobe-index/`.

9. The Java Language Environment. In: *Oracle Technology Network - Java* [online]. 1997 [cit. 2020-04-21]. Dostupné z: https://www.oracle.com/technetwork/java/intro-141325.html.

10. *Vaadin - The Best Java framework for Progressive Web Apps* [online]. Vaadin Ltd., 2020 [cit. 2020-04-21]. Dostupné z: https://vaadin.com.

11. Historical yearly trends in the usage statistics of server-side programming languages for websites. In: *W3Techs - World Wide Web Technology Surveys* [online]. © 2009-2020 [cit. 2020-04-21]. Dostupné z: https://w3techs.com/technologies/history_overview/programming_language/ms/y.

12. 8 Popular PHP Frameworks For Web Development in 2020. In: *Hacker Noon* [online]. 2020 [cit. 2020-04-21]. Dostupné z: https://hackernoon.com/8-popular-php-frameworks-for-web-development-in-2020-od3f38ez.

13. Top 10 Python Frameworks for Web Development In 2020. In: *Net Solutions* [online]. 2020 [cit. 2020-04-21]. Dostupné z: https://www.netsolutions.com/insights/top-10-python-frameworks-for-web-development-in-2019/.

14. RICHARDS, Mark. *Software Architecture Patterns*. USA: O'Reilly Media, 2015. ISBN 1491971436.

15. Why You Should Use a Web Framework. In: *DEV Community* [online]. 2018 [cit. 2020-04-23]. Dostupné z: https://dev.to/silkster/why-you-should-use-a-framework--1ga2.

16. Spring Boot - Overview. In: *Spring* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://spring.io/projects/spring-boot.

17. Web on Servlet Stack. In: *Spring* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html.

18. FOWLER, M. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2003. ISBN 0321127420.

19. JIA, Grace. Prototype Design: What is a Wireframe, and What is a Prototype? In: *Mockplus* [online]. 2017 [cit. 2020-04-25]. Dostupné z: https://www.mockplus.com/blog/post/what-is-a-wireframe-what-is-a-prototype.

20. DWORMAN, Garett. When To Prototype, When To Wireframe – How Much Fidelity Can You Afford? In: *UsabilityGeek* [online]. 2014 [cit. 2020-04-25]. Dostupné z: https://usabilitygeek.com/when-to-prototype-when-to-wireframe-fidelity/.

21. LIM, Winnie. A Beginner's Guide to Wireframing. In: *https://webdesign.tutsplus.com* [online]. 2012 [cit. 2020-04-25]. Dostupné z: https://webdesign.tutsplus.com/articles/a-beginners-guide-to-wireframing--webdesign-7399.

22. NIELSEN, Jakob. How to Conduct a Heuristic Evaluation. In: *Nielsen Norman Group* [online]. 1994 [cit. 2020-04-25]. Dostupné z: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/.

23. NIELSEN, Jakob. 10 Usability Heuristics for User Interface Design. In: *Nielsen Norman Group* [online]. 1994 [cit. 2020-04-25]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/.

24. KRANITSAS, Thomas. Coding Best Practises. In: *https://www.topcoder.com* [online]. 2017 [cit. 2020-04-28]. Dostupné z: https://www.topcoder.com/coding-best-practices/.

25. Welcome to Apache Maven. In: *http://maven.apache.org* [online]. 2020 [cit. 2020-04-28]. Dostupné z: http://maven.apache.org.

26. FADATARE, Ramesh. Spring MVC Project Structure. In: *Java Guides* [online]. 2019 [cit. 2020-04-28]. Dostupné z: https://www.javaguides.net/2019/01/spring-mvc-project-structure.html.

27. Spring MVC Model Interface. In: *www.javatpoint.com* [online]. ©2011-2018 [cit. 2020-04-28]. Dostupné z: https://www.javatpoint.com/spring-mvc-model-interface.

28. The Top 50 Worst Passwords of 2019. In: *SplashData* [online]. 2019 [cit. 2020-05-01]. Dostupné z: https://www.teamsid.com/1-50-worst-passwords-2019/.

29. GEORGE, Anita. Here are the 10 worst passwords of 2019, and a few tips on creating better ones. In: *https://www.digitaltrends.com* [online]. 2019 [cit. 2020-05-01]. Dostupné z: https://www.digitaltrends.com/computing/the-10-worst-passwords-of-2019-and-tips-to-create-more-secure-ones/.

30. Class BCryptPasswordEncoder. In: *Spring Security* [online]. 2020 [cit. 2020-05-01]. Dostupné z: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html.

31. CHAPMAN, Nigel; CHAPMAN, Jenny. *Authentication and Authorization on the Web*. Velká Británie: MacAvon Media, 2012. ISBN 0956737056.

32. OWASP Top 10 Application Security Risks - 2017. In: *OWASP* [online]. 2017 [cit. 2020-05-04]. Dostupné z: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_Top_10.html.

33. About OWASP. In: *OWASP Top Ten 2017* [online]. 2017 [cit. 2020-05-04]. Dostupné z: `https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/`.

34. XML External Entity (XXE) Processing. In: *OWASP* [online]. 2020 [cit. 2020-05-04]. Dostupné z: `https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing`.

35. SULLIVAN, Bryan; LIU, Vincent. *Web Application Security, A Beginner's Guide*. USA: McGraw-Hill Education, 2012. ISBN 0071776168.

36. WILSON, Chauncey. *User Interface Inspection Methods: A User-Centered Design Method*. USA: Elsevier Books, 2014. ISBN 012410391X.

Příloha A

Seznam použitých zkratek

FURPS Functionality, Usability, Reliability, Performance and Supportability

PDF Portable Document Format

CRUD Create, Read, Update, Delete

API Application Programming Interface

MVC Model-View-Controller

HTML Hypertext Markup Language

CSS Cascading Style Sheets

SGML Standard Generalized Markup Language

XML Extensible markup language

JS JavaScript

WHATWG Web Hypertext Application Technology Working Group

Sass Syntactically Awesome Style Sheets

Less Leaner Style Sheets

JVM Java Virtual Machine

POSIX Portable Operating System Interface

open source Otevřený software, otevřenost znamená jak technickou dostupnost kódu, tak legální dostupnost – licenci softwaru

CMS Content Management System

JSP JavaServer Pages

JSTL JSP Standard Tag Library

URL Uniform Resource Locator

SQL Structured Query Language

GUI Graphical User Interface

DRY Don't Repeat Yourself

SMS Short message service

OWASP Open Web Application Security Project

LDAP Lightweight Directory Access Protocol

Todo list

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS