FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webový editor Jinja šablon
Student:	Štěpán Štrba
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat webový editor Jinja šablon, který umožní našeptávání i další pomocné prvky po vzoru tradičních IDE včetně náhledu (vykreslení se zadanými daty). Jedna šablona se může skládat z více souborů a to i statických (např. obrázek). Současně šablony nesmí být omezeny pouze na generování HTML, ale prakticky jakýkoliv textový typ souboru. V práci postupujte v souladu s metodami softwarového inženýrství:

- Seznamte se s jazykem Jinja2 a související nástrojovou podporou v rámci analýzy.
- Proveďte stručnou rešerši obdobných řešení (například pro jiné jazyky).
- Sestavte funkční a nefunkční požadavky na aplikaci dle analýzy a rešerše.
- Vytvořte návrh architektury a UI aplikace naplňující požadavky. Při návrhu se zaměřte na budoucí rozšiřitelnost a uživatelskou přívětivost.
- Implementujte a otestujte aplikaci dle návrhu.
- Zhodnoťte výsledky práce a navrhněte další budoucí rozvoj.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 4. března 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webový editor Jinja šablon

Štěpán Štrba

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

10. května 2021

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Markovi Suchánkovi za odborné vedení, pomoc a cenné rady při zpracování této práce. Také chci poděkovat své rodině a přátelům za jejich podporu při studiu.

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 10. května 2021 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Štrba, Štěpán. *Webový editor Jinja šablon*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021. Dostupný také z WWW: ⟨`http://jinjaeditor.eu`⟩.

Abstrakt

Bakalářská práce se zabývá návrhem a implementací webového editoru Jinja šablon s podporou zvýraznění syntaxe, našeptávání syntaxe a vykreslení napsané Jinja šablony.

Nejdříve je čtenář seznámen s požadavky na aplikaci. Dále je provedeno porovnání a následný výběr používaných technologií. S ohledem na požadavky je navržena architektura a UI aplikace. V realizační části je popsána implementace funkčních požadavků a uživatelského rozhraní aplikace. Následně je představeno testování kódu aplikace a její nasazení.

Webový editor Jinja šablon byl úspěšně implementován a je dostupný na adrese www.jinjaeditor.eu.

Klíčová slova editor šablon, zvýraznění syntaxe, našeptávání syntaxe, vykreslení šablony, Jinja, webová aplikace, React

Abstract

The aim of this bachelor thesis is to design and implement web editor of Jinja templates with support of syntax highlighting, code completion, and rendering of written Jinja template.

First, the reader is introduced to the requirements of the application. Next comparison and selection of used technologies is performed. Then the architecture and UI of the application is designed with regards to the requirements. After that, the implementation of functional requirements and user interface is described. Finally testing the application code and its deployment is introduced.

Web editor of Jinja templates was successfully implemented and is available at www.jinjaeditor.eu.

Keywords template editor, syntax highlighting, code completion, template rendering, Jinja, web application, React

Obsah

Seznam obrázků

Seznam ukázek kódu

Úvod

Občas potřebuji vytvořit mnoho podobných textových dokumentů, například webové stránky, faktury či emaily. Tyto dokumenty ale nechci vytvářet po jednom, když většina jejich obsahu je stejná. Nejlehčí by bylo si sepsat jeden dokument, šablonu, s požadovaným obsahem a v něm si části, které chci nahradit různými daty, označit nějakými značkami. K tomuto mohu využít šablonovací systém.

Šablonovací systém nahrazuje speciální značky v šablonách daty, tím vytvoří požadovaný výstup s možností lehké změny částí výstupu. Požadovaný výstup může být jakýkoliv textový dokument, například webová stránka, faktura nebo email. Při vývoji aplikací mohu takový systém využít k oddělení prezentační vrstvy od aplikační. Prezentační vrstvu mohu měnit bez obavy, že poškodím aplikační kód. Zároveň tím zlepšuji čitelnost kódu a redukuji opakující se části požadovaného výstupu. Jedním z těchto systémů je Jinja.

Jinja je moderní textový šablonovací systém pro Python, který byl v roce 2008 zveřejněn softwarovým programátorem Arminem Ronacherem. Systém Jinja byl vytvořen po vzoru šablonovacího systému Django, toto se projevuje například v podobnosti syntaxe. Oproti Django ale poskytuje výrazy podobné syntaxi jazyka Python, zajišťuje vyhodnocování šablon v sandboxu a jeho šablonovací procesor by měl být až 20x rychlejší než šablonovací procesor Django. Tento šablonovací jazyk také využívá dědičnost šablon, automatické HTML escapování kvůli prevenci XSS a jednoduchý filtrovací systém podobný Unixové pipeline. Jinja se stále rozvíjí, jeho nejnovější verze se běžně označuje Jinja2 a používá se například ve webových frameworcích Flask a Bottle.

Pro tento šablonovací systém zatím neexistuje webový editor, který by našeptával a zvýrazňoval syntaxi. V této práci se budu věnovat vytvoření takového editoru. Nejdříve se seznámím s jazykem Jinja a provedu stručnou rešerši webových editorů pro jiné jazyky. Poté sestavím funkční a nefunkční požadavky a podle těchto požadavků navrhnu architekturu a UI aplikace. Podle těchto návrhů aplikaci implementuji a otestuji.

V první kapitole stanovím všechny cíle, které chci při řešení práce postupně splnit. V druhé kapitole provedu analýzu podobných webových editorů a následně sestavím požadavky aplikace. Tyto požadavky rozdělím na nefunkční a funkční, které dále rozdělím do čtyř kategorií metody MoSCoW. Ve třetí kapitole zvolím použité technologie. V této kapitole také navrhnu architekturu a UI aplikace. V následující kapitole se budu zabývat implementací částí aplikace a její UI. V páté kapitole představím testování a nasazení aplikace. V poslední kapitole představím prototyp aplikace. V závěru stanovené cíle zhodnotím a navrhnu další budoucí rozvoj.

Kapitola 1

Cíl práce

V této bakalářské práci je hlavním cílem vytvoření webového editoru Jinja šablon. Dílčí cíle se řídí dle metod softwarového inženýrství. Prvním dílčím cílem je provedení stručné analýzy obdobných webových editorů (například pro jiné jazyky). Dalším dílčím cílem je sestavení funkčních a nefunkčních požadavků dle analýzy, rešerše a zadání bakalářské práce. Třetím dílčím cílem je vytvoření návrhu architektury a UI aplikace. Při návrhu je potřeba se zaměřit na budoucí rozšiřitelnost a uživatelskou přívětivost. Posledním cílem je aplikaci podle vytvořeného návrhu implementovat a otestovat.

Kapitola 2

Analýza

Pro správné určení funkčních i nefunkčních požadavků provedu analýzu již existujících řešení. Pro Jinja šablony těchto řešení není mnoho, proto do analýzy zahrnu i editory programovacího jazyka JavaScript. Při analýze budu pracovat s již částečně stanovenými požadavky vedoucího práce ze zadání bakalářské práce. Mezi tyto požadavky patří:

- Našeptávání syntaxe a další pomocné prvky po vzoru tradičních Integrated Development Environment (IDE)
- Vykreslení šablony se zadanými daty
- Správa projektových souborů
- Psaní šablony skládající se z více souborů

2.1 Podobné aplikace

Podobné aplikace se dají rozdělit do dvou kategorií - webové IDE a softwarové IDE. Softwarové IDE většinou nabízí mnohem více funkcí. Je ale nutnost instalace, a tudíž se i nabízí možnost práce offline. To s webovou aplikací nelze zaručit, softwarovým editorům tedy neplánuji konkurovat a do analýzy je nezařadím. Webových editorů Jinja šablon není mnoho, zařadím do analýzy tedy i webové IDE programovacího jazyka JavaScript. Tyto IDE jsem našel pomocí internetových vyhledávačů a vybral jsem je na základě popularity.

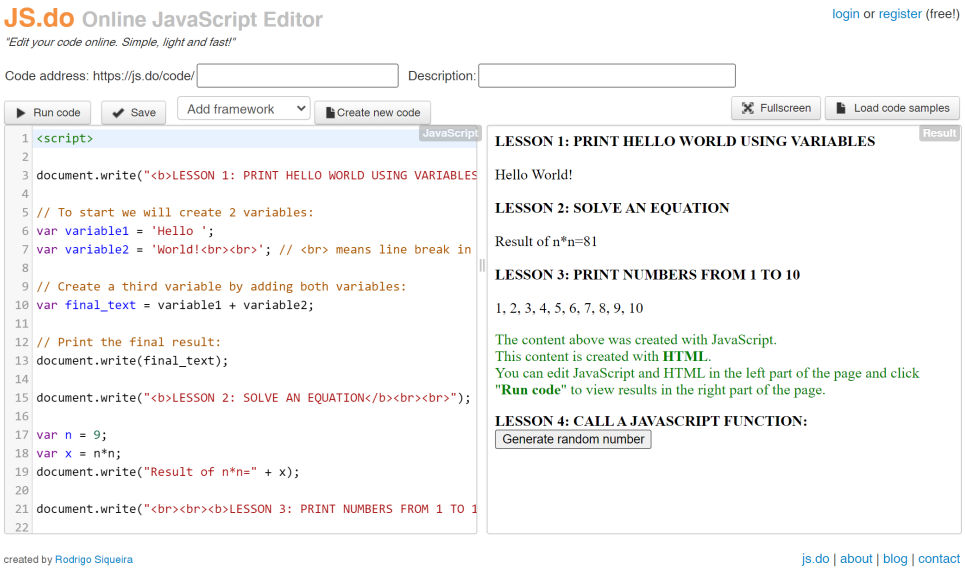
2.1.1 TTL255 J2Live - Live Jinja2 Parser

TTL255 J2Live - Live Jinja2 Parser [1] je webový editor Jinja šablon. V tomto editoru lze vykreslit napsanou šablonu s přidanými daty. Uživatelské rozhraní je intuitivní a aplikace nabízí i různé módy a možnosti vykreslení šablony. V aplikaci ale chybí požadovaná možnost psaní šablony skládající se z více

souborů, našeptávání syntaxe nebo jakékoliv další pomocné prvky po vzoru tradičních IDE.

2.1.2 Online JavaScript Editor js.do

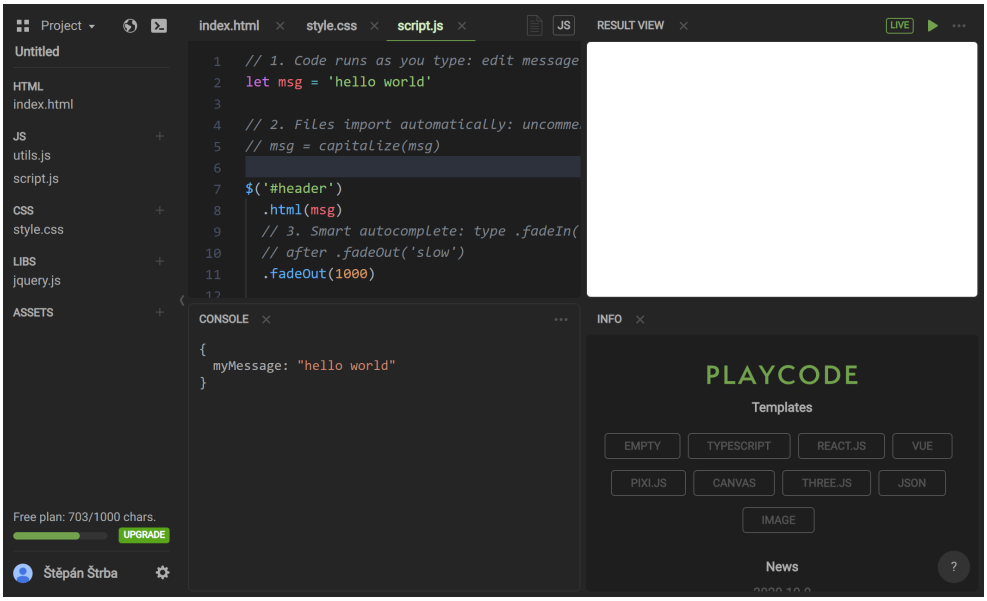
Online JavaScript Editor js.do [2] je webový editor JavaScriptu s vykreslením napsaného kódu. Napsaný kód je možné uložit a sdílet s ostatními. Aplikace využívá k editaci kódu JavaScript komponentu CodeMirror [3]. Tato komponenta podporuje mnoho pomocných prvků po vzoru tradičních IDE, včetně našeptávání, vyhledávání v textu a zvýraznění syntaxe. Aplikace z těchto pomocných prvků ale využívá jen zvýraznění syntaxe, také chybí správa projektových souborů.



Obrázek 2.1: Online JavaScript Editor js.do [2]

2.1.3 p5.js Web Editor

p5.js Web Editor [4] je webový editor JavaScript, HTML i CSS kódu s vykreslením napsaného kódu. Aplikace k editování kódu také používá JavaScript komponentu CodeMirror podobně jako Online JavaScript Editor. Navíc ale využívá například vyhledávání v textu a zobrazování číslování řádků kódu. Dále aplikace obsahuje správu projektových souborů a složek včetně možnosti je stahovat a nahrávat. V aplikaci je možné nastavit barevné téma.



Obrázek 2.2: Aplikace PlayCode [5]

2.1.4 PlayCode

PlayCode [5] je webový editor JavaScript, HTML i CSS kódu s vykreslením napsaného kódu. Tato aplikace má moderní uživatelské rozhraní, imitující tradiční softwarové IDE, včetně možností změny barevného tématu.

V aplikaci je možné spravovat projektové soubory společně s nahráním vlastních souborů. Nahrané soubory lze využívat v kódu, ale chybí možnost je editovat. Projektové složky jsou přednastavené a nelze přidat další. Projekt je možné uložit a sdílet s ostatními. Editor kódu podporuje pomocné prvky podobně jako CodeMirror. Aplikace má i placenou verzi, ve které lze sdílet vykreslený kód, změnit adresu uloženého projektu a vykreslené stránky, zvolit viditelnost projektu a napsat více než 1000 znaků v projektu.

2.1.5 Shrnutí

Výsledkem této analýzy je zjištění, že neexistuje webový editor Jinja šablon, který by splňoval již částečně stanovené požadavky. Existuje ale mnoho editorů jiných jazyků, které naše požadavky z části splňují. Tyto editory většinou podporují alespoň část pomocných prvků po vzoru tradičních IDE (našeptávání a zvýraznění syntaxe, vyhledávání v textu či číslování řádků). Uživatelské rozhraní a ovládání mají podobné a míří tím na uživatele počítačů či tabletů. V případě, že mají správu projektových souborů, podporují i jejich nahrávání.

2.2 Požadavky

Specifikace požadavků je podstatnou součástí plánování rozsahu projektu. Požadavky rozdělím na funkční a nefunkční. Funkční požadavky dále rozdělím do kategorií must have, should have, could have a won't have podle metody MoSCoW.

2.2.1 Funkční požadavky

Funkční požadavky určují, co má aplikace dělat a co musí umět.

F1 Projektový adresář - V projektovém adresáři budou vypsané všechny Jinja šablony a projektové složky.

- **F1.1 Přehled projektových souborů a složek**
- **F1.2 Organizace projektových souborů a složek** - Jinja šablony a projektové složky bude možné vytvořit, mazat, přesouvat a přejmenovat. Jinja šablony bude navíc možné nahrát i stáhnout.
  - **F1.2.1 Vytváření Jinja šablon a projektových složek**
  - **F1.2.2 Odstranění Jinja šablon a projektových složek**
  - **F1.2.3 Přesouvání Jinja šablon a projektových složek**
  - **F1.2.4 Přejmenování Jinja šablon a projektových složek**
  - **F1.2.5 Automatické ukládání Jinja šablon a projektových složek**
  - **F1.2.6 Ukládání Jinja šablon a projektových složek na cloud**
  - **F1.2.7 Nahrání Jinja šablon**
  - **F1.2.8 Stažení Jinja šablon**

F2 Textový editor Jinja šablon - Textový editor Jinja šablon umožní pomocné prvky po vzoru tradičních IDE.

- **F2.1 Zvýraznění Jinja syntaxe zadanými barvami**
- **F2.2 Změna barev zvýraznění Jinja syntaxe**
- **F2.3 Pohyb v textu pomocí myši i klávesnice**
- **F2.4 Vybírání textu pomocí myši i klávesnice**
- **F2.5 Odstranění vybraného textu**
- **F2.6 Kopírování vybraného textu**
- **F2.7 Vkládání zkopírovaného textu**
- **F2.8 Našeptávání Jinja syntaxe při psaní** - Našeptávání syntaxe bude doplňovat nedopsaná klíčová slova
- **F2.9 Find and replace** - Funkce hledání či nahrazení textu

F2.10 Multiple cursor - Více textových kurzorů pro možnost editace textu na více místech zároveň

F2.11 Zobrazení číslování řádků

F3 Vykreslení šablony - V aplikaci bude možné vykreslit otevřenou Jinja šablonu, při vykreslení se vykreslí i ostatní použité šablony.

F3.1 Vykreslení šablony v textovém formátu

F3.2 Vykreslení šablony v HTML formátu

F3.3 Stažení vykreslené šablony

2.2.2 Nefunkční požadavky

Nefunkční požadavky určují omezení a standardy kladené na aplikaci.

N1 Webová aplikace - Aplikace bude implementována formou webu.

N2 Responzivita - Design aplikace bude mířit na uživatele počítačů či tabletů. Aplikace bude responzivní, aby mohla být využívaná uživateli s různými rozlišeními obrazovek.

N3 Ovládání - Aplikace bude ovládána myší a klávesnicí. Ovládání bude intuitivní dle zvyklostí z běžně užívaných editorů a IDE.

N4 Lokalizace - Aplikace bude připravená pro přeložení do jiných jazyků.

N5 Indikace stavu - Uživatel vždy vidí, co se děje a v případě dlouhé/složité operace se zobrazí nějaká indikace načítání.

N6 Podpora prohlížečů - Aplikace bude podporovat následující verze prohlížečů:

- Google Chrome (od verze 89)
- Firefox (od verze 87)

2.2.3 MoSCoW

Podle metody MoSCoW [6] rozděluji požadavky do čtyř kategorií:

1 Must have - nejdůležitější požadavky, bez kterých je projekt nesplněný

F1	Projektový adresář
F1.1	Přehled projektových souborů a složek
F1.2.1	Vytváření Jinja šablon a projektových složek
F1.2.2	Odstranění Jinja šablon a projektových složek
F1.2.8	Stažení Jinja šablon
F2	Textový editor Jinja šablon
F2.3	Pohyb v textu pomocí myši i klávesnice

2 Should have - důležité požadavky, které přidávají významnou hodnotu

- **F2.1** Zvýraznění Jinja syntaxe zadanými barvami
- **F2.4** Vybírání textu pomocí myši i klávesnice
- **F2.6** Kopírování vybraného textu
- **F2.7** Vkládání zkopírovaného textu
- **F2.8** Našeptávání Jinja syntaxe při psaní
- **F3.1** Vykreslení šablony v textovém formátu

3 Could have - požadavky, které přidávají malou hodnotu

- **F1.2.3** Přesouvání Jinja šablon a projektových složek
- **F1.2.4** Přejmenování Jinja šablon a projektových složek
- **F1.2.7** Nahrání Jinja šablon
- **F1.2.5** Automatické ukládání Jinja šablon a projektových složek
- **F2.5** Odstranění vybraného textu
- **F2.9** Find and replace
- **F3.3** Stažení vykreslené šablony

4 Won't have - nedůležité požadavky, které mohou být přidány v budoucí verzi

- **F1.2.6** Ukládání Jinja šablon a projektových složek na cloud
- **F2.2** Změna barev zvýraznění Jinja syntaxe
- **F2.10** Multiple cursor
- **F2.11** Zobrazení číslování řádků
- **F3.2** Vykreslení šablony v HTML formátu

Kapitola 3

Návrh

3.1 Split Stack Development

Split Stack Development je návrhový vzor, který rozděluje vývoj frontendu a backendu na dvě části. Tyto části fungují nezávisle na sobě a komunikují spolu přes API [7] (překlad Tat Dat Duong). Toto rozdělení umožňuje zvolit různé technologie pro frontend a backend bez omezení na jejich vzájemné podpoře. Jelikož jsou samotné části na sobě nezávislé, tak mohou být samostatně a současně vyvíjeny.

3.2 Frontend

3.2.1 Document Object Model

Document Object Model (DOM) spojuje webové stránky se skripty či programovacími jazyky pomocí reprezentování struktury dokumentu v paměti jako logický strom. Každá větev stromu končí v uzlu, který obsahuje objekt.

Metody DOM umožňují přístup k tomuto stromu a podporují změnu struktury, stylu a obsahu dokumentu. Pomocí těchto metod je také možné přidat funkce, které se na základě nějaké události zavolají. Tyto události mohou být například kliknutí myší či stisknutí klávesy. [8]

Díky DOM lze změnit část stránky bez nutnosti načtení celé nové stránky. To značně zrychlí načtení změn na stránce a tím se i aplikace stane více uživatelsky přívětivá. DOM také umožnil aplikace typu Single Page Application.

3.2.2 Single Page Application

Webové aplikace se dají rozdělit na návrhové vzory Single Page Application (SPA) a Multi Page Application (MPA). Každý má své pro a proti a je vhodný na různé použití. Pro správný vývoj aplikace je nutné jeden z nich zvolit.

Multi Page Application při každé změně obsahu překreslí celou stránku. Zároveň překreslí i stejné části, například menu či footer. Při načítání nového obsahu je na prohlížeči, aby zobrazil načítací animaci. Například při odesílání formuláře se může stát, že si načítání nové stránky uživatel nevšimne a zašle formulář několikrát. MPA jsou vhodné pro webové obchody.

Single Page Application překresluje jen části stránky, které se mají změnit. Díky tomu je načítání nového obsahu mnohem rychlejší než u MPA. Při načítání obsahu je na vývojáři, aby zobrazil načítací animaci uživateli. SPA se tímto přibližují funkcionalitě desktopových aplikací.

Výsledná webová aplikace bude obsahovat projektový adresář, textový editor Jinja šablon a zobrazení vykreslené šablony. Zobrazení všeho na jedné stránce bude pro uživatele více přívětivé, než kdyby každá část aplikace byla na jiné stránce. Kvůli tomu zvolím SPA.

3.2.3 Výběr technologií

K psaní frontendu vyberu jeden z momentálně nejvíce používaných frameworků, které jsou na to určeny. Správně vybraný framework mi značně ulehčí práci, jelikož již bude obsahovat velkou část základního kódu a já se tak budu moci soustředit na vytváření funkčností aplikace.

3.2.3.1 Angular

Angular [9] je frontend framework vyvíjený společností Google od roku 2010. V roce 2016 byl kompletně přepsán, aby lépe řešil nové výzvy ve vývoji webových aplikací. Angular využívá jazyk TypeScript a podporuje obousměrné vázání dat, díky kterému probíhá synchronizace dat v reálném čase.

3.2.3.2 React

React [10] byl vytvořen v roce 2011 společností Facebook a následně vydán v roce 2013. Aplikace napsané v Reactu se skládají jen z komponent, k jejich rychlému překreslení je využíván virtuální DOM.

Všechny komponenty v Reactu obsahují stav. Při jeho změně React aktualizuje virtuální DOM, porovná ho se starou verzí virtuálního DOM a v reálném DOM překreslí jen změněné části.

3.2.3.3 Vue.js

Vue.js [11] byl při vývoji inspirován již zmíněnými frameworky, proto jim je velmi podobný a implementuje mnoho jejich schopností. Například obousměrné vázání dat z Angularu a využití virtuálního DOM z Reactu. Oproti těmto frameworkům je mnohem jednodušší a méně omezující.

3.2.3.4 Shrnutí

Aplikace napsané v Angularu jsou sestavené z mnoha částí (moduly, komponenty, šablony, services, pipes...), každá část má určenou funkci. Těmto částem může být těžké porozumět a správně je použít.

React využívá pouze komponenty a nechává na programátorovi, zda je chce rozdělit do více částí. React je také vysoce zpětně kompatibilní. Oproti Vue je React více používaný a díky tomu má i velkou podporu od svých uživatelů. Pro psaní frontend části aplikace jsem proto zvolil React.

3.2.4 Testování

Automatické testování kódu je nutnou součástí vývoje aplikace. Díky napsaným testům získám jistotu ve správné chování aplikace a zároveň budu nucen k psaní lehce testovatelných funkcí. Lehce testovatelné funkce jsou kratší a jednodušší, tedy i lépe pochopitelnější. Testy se dají rozdělit do následujících typů:

Unit test se zaměřuje na testování nejmenších částí aplikace, například funkcí nebo metod.

Snapshot test je používán k testování vzhledu aplikace. Tento test porovná aktuální vzhled aplikace s předchozím a zvýrazní nalezené změny.

Integrační test kontroluje, zda jednotlivé části aplikace správně fungují po jejich spojení s ostatními.

End to end test simuluje vstup od uživatele a tím testuje aplikaci jako celek.

Pro testování frontend části aplikace jsem zvolil jen unit testy, porovnám tedy technologie, které psaní těchto testů podporují.

3.2.4.1 Jest

Jest [12] je testovací framework vytvořený Facebookem se zaměřením na jednoduchost. Tento framework byl primárně navržen pro aplikace napsané v Reactu, ale může být použit i pro jiné JavaScript aplikace. Jest dokáže vygenerovat zprávy o množství kódu pokrytého testy, podporuje mocking a je vhodný na snapshot, integrační i unit testy, které spouští izolovaně a paralelně.

3.2.4.2 AVA

AVA je minimalistická knihovna pro testování Node.js aplikací, která obsahuje velmi stručnou API a tím nutí programátora k psaní jednoduchých testů. K neobsaženým vlastnostem, například mocking, je potřeba využít dalších knihoven.

AVA spouští testy souběžně, pro každý testovací soubor má samostatný proces [13] (překlad Martin Pilný). Tato knihovna podporuje integrační a unit testy.

3.2.4.3 Mocha

Mocha je open source flexibilní framework k testování JavaScript kódu. Podobně jako AVA obsahuje Mocha jen nejnutnější vlastnosti k psaní a spouštění testů.

Mocha pouští testy sériově, umožňující tím flexibilní a přesné hlášení chyb. Výjimky neodchycené při běhu testu ohlašuje v rámci správného testu [14] (překlad Bc. Narek Vardanjan). Tento framework je vhodný na integrační, unit a end to end testy.

3.2.4.4 Shrnutí

Mocha spouští testy sériově, kvůli tomu je pomalejší než ostatní vybrané technologie. Jest oproti AVA obsahuje vše, co potřebuji. Navíc byl navržen pro React aplikace a je doporučen v oficiální dokumentaci Reactu. Z těchto důvodů jsem pro psaní unit testů zvolil testovací framework Jest.

3.2.5 Překlad

3.2.5.1 React-i18next

React-i18next [15] je založený na frameworku i18next. Tento framework nabízí možnost zvolení používaných knihoven, rozdělení překladů do více souborů a jejich načítání na vyžádání. I18next také obsahuje velké množství modulů a pluginů.

3.2.5.2 React Intl

Knihovna React Intl [16] byla vytvořena společností Yahoo a je součástí sady knihoven FormatJS pro překlad aplikací. Tato knihovna nabízí několik komponent pro překlad textu a formátování datumů, časů, měn a čísel.

3.2.5.3 Shrnutí

Popsané nástroje pro překlad jsou si velmi podobné a výběr není jednoduchý. Framework i18next ale podporuje mnohem více frameworků než sada knihoven FormatJs, má pro mě srozumitelnější syntaxi a nabízí možnost lehkého rozdělení překladů do více souborů pro případ, že se aplikace rozroste. Kvůli těmto důvodům jsem zvolil react-i18next.

3.3 Backend

3.3.1 Výběr technologií

Jinja je šablonovací systém pro jazyk Python, její šablony tedy nemohu vykreslit na frontendu, pro který jsem zvolil javascriptový framework React. Jinja šablony budu vykreslovat na backendu, pro který musím vybrat nějaký framework v jazyce Python.

Frameworky napsané v jazyce Python se dělí na full stack frameworky a microframeworky. Full stack frameworky obvykle obsahují velké množství vestavěných funkcí, modulů a závislostí. Microframeworky naopak obsahují naprosté minimum a nechávají na programátorovi, jaké moduly a závislosti sám zvolí.

3.3.1.1 Django

Django [17] je jeden z full-stack frameworků, který využívá MVC architekturu. Tento framework se snaží zahrnout všechny potřebné funkcionality ke tvorbě webu. Mezi ně patří například autentizace, URL routing, šablonovací systém, ORM pro mapování objektů na databázové tabulky a mnoho dalšího.

3.3.1.2 Bottle

Bottle [18] je microframework originálně určený pro vytváření API. Aplikace napsané v tomto frameworku jsou vždy jen v jednom souboru. V základu obsahuje URL routing, šablonovací systém a vestavěný vývojový server. Kromě standardní knihovny pro Python nemá žádné závislosti.

3.3.1.3 Pyramid

Pyramid [19] je další microframework, jehož cílem je udělat co nejvíce s minimální složitostí. Projekt lze začít s projektovou šablonou nebo vhodný šablonovací systém, přístup k databázi a mnoho dalšího zvolit až během vývoje. Díky tomu je Pyramid vhodný pro rozvíjející se aplikace.

3.3.1.4 Flask

Flask [20] je microframework postavený na knihovně webových aplikací WSGI Werkzeug a Jinja2. Obsahuje integrovaný vývojový server, šablonovací systém Jinja2 a podporu pro unit testy. Flask pomáhá postavit solidní základ webové aplikace bez nutnosti jakýchkoliv závislostí.

3.3.1.5 Shrnutí

Na backendu bude API server pro vykreslování Jinja šablon a v budoucí verzi aplikace nejspíše i ukládání projektových souborů a složek na cloud. Kvůli

tomu neočekávám využití většiny funkcionalit zahrnutých ve full-stack frameworcích, jakým je Django. Aplikace napsané v Bottle musí být v jednom souboru, proto je jejich správa a rozšiřování s postupem času těžší.

Nenašel jsem žádný důvod, proč nepoužít microframework Pyramid. Bohužel jsem ale také nenašel žádný důvod proč bych ho měl zvolit oproti Flask. Flask již v základu obsahuje požadovaný šablonovací systém Jinja2 a je momentálně nejvíce využívaný microframework, má tedy i velkou podporu od svých uživatelů. Díky těmto důvodům byl Flask jasná volba.

3.3.2 Testování

Pro testování backend části aplikace jsem, stejně jako pro testování frontend části aplikace, zvolil unit testy. Porovnám tedy jen technologie, které psaní těchto testů podporují.

3.3.2.1 Unittest

Unittest [21], také nazývaný PyUnit, je framework pro psaní unittestů v jazyce Python. Tento framework byl inspirován testovacím Java frameworkem JUnit a je součástí standardní Python knihovny od její verze 2.1. Unittest podporuje znovupoužití částí testů a jejich organizaci do tříd, další funkčnosti je možné přidat použitím nějakého z mnoha podporovaných pluginů.

3.3.2.2 Pytest

Pytest [22] je testovací framework pro jazyk Python s podporou pro více jak 800 pluginů. Pytest podporuje parametrizaci testů a spouštění testů napsaných pro unittest.

3.3.2.3 Shrnutí

Rozhodl jsem se pro zvolení frameworku Unittest, jelikož je součástí standardní Python knihovny a má syntaxi podobnou testovacímu frameworku Jest, který jsem zvolil pro psaní unit testů frontend části aplikace.

3.4 Architektura

Architektury k porovnání jsem zvolil na základě jejich využití a popularitě ve webovém vývoji.

3.4.1 MVC

Architekturu MVC poprvé představil Trygve M. H. Reenskaug v roce 1979 [23] a momentálně je jedna z nejznámějších architektur. Jedná se o třívrstvou

architekturu s obousměrným tokem dat, podle které by kód měl být strukturován do tří vrstev:

Model se stará o logiku a data v aplikaci.

View zobrazuje data uživateli.

Controller získává data z vrstvy Model a předává je vrstvě View. Zároveň se stará o vstup od uživatele.

3.4.2 Flux

Flux [24] je vícevrstvá architektura pro tvorbu SPA, která je navržena Facebookem po jejich problémech s architekturou MVC. Flux využívá jednosměrný tok dat a doporučuje rozdělení kódu do následujících vrstev:

Action je objekt se všemi potřebnými informacemi k provedení určité akce. Tento objekt může zaslat jak uživatel, tak samotná aplikace. Například po obdržení dat ze serveru.

Dispatcher pouze přijímá objekty Action a volá vhodné funkce z vrstvy Store.

Store obsahuje veškerou logiku a stav aplikace.

View zobrazuje data uživateli a mění se pouze když je potřeba.

3.4.3 Redux

Redux [25] je primárně JavaScript knihovna pro správu stavu aplikace, navíc ale přináší vlastní architekturu vycházející z Flux architektury. Podle této architektury by kód měl být rozdělen do následujících vrstev:

Action je stejný objekt jako v architektuře Flux.

Store může být jen jeden v celé aplikaci. Store přijímá akce a společně s aktuálním stavem aplikace je zasílá vrstvě Reducer, od té získá nový stav aplikace, na který změní aktuální stav aplikace.

Middleware poskytuje možnost přidání funkcionality mezi odesláním akce a okamžikem, kdy akce dosáhne vrstvy Reducer.

Reducer je funkce, která vrací nový stav aplikace na základě přijatých parametrů. Na základě vstupu musí mít pokaždé stejný výstup a zároveň nesmí měnit aktuální stav aplikace.

View vrstva je stejná jako v architektuře Flux.

3.4.4 CBA

Component-based architektura se zaměřuje na rozklad designu na jednotlivé funkcionální nebo logické komponenty. Komponenta je modulární, přenositelný, zaměnitelný a znovupoužitelný soubor dobře definovaných funkcionalit, které zapouzdřují svoji implementaci a odhalují ji jako rozhraní vyšší úrovně [26] (překlad Le Thanh Hung). V této architektuře lze využít návrhový vzor Container Component, podle kterého by kód měl být rozdělen do následujících vrstev:

Container se stará o logiku a data v aplikaci.

Component zobrazuje data uživateli.

3.4.5 Shrnutí

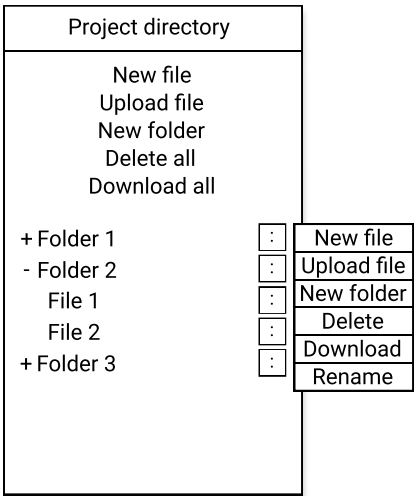
Zvolil jsem architekturu CBA s návrhovým vzorem Container Component. React využívá jednosměrný tok dat, kvůli tomu není vhodný MVC se svým obousměrným tokem dat. Architektura Flux je vhodná spíše pro větší aplikace, jako je například Facebook. Redux vychází z architektury Flux a očekávám, že mi při psaní aplikace bude stačit správa stavu aplikace již obsažená v Reactu.

3.5 Uživatelské rozhraní

Uživatelské rozhraní je část aplikace, kde dochází k interakci mezi uživatelem a aplikací. Aplikace přijímá vstup od uživatele a na jeho základě vykoná nějakou akci, například zobrazení dat nebo jejich změnu. Takové rozhraní by mělo být intuitivní, efektivní a uživatelsky přívětivé.

Při vytváření uživatelského rozhraní se inspiruji jak u softwarových IDE (například PhpStorm [27]), tak u webových IDE již zmíněných v kapitole 2.1. Uživatelské rozhraní se snažím zachovat jednoduché, aby jím uživatel nebyl příliš zatížen a neměl problém nalézt požadovanou část aplikace. V případě dlouhé akce je zobrazeno načítání, úspěšné či neúspěšné dokončení akce je uživateli vždy dáno najevo. Aplikace je responzivní, aby byla použitelná na obvyklých obrazovkách počítačů či tabletů.

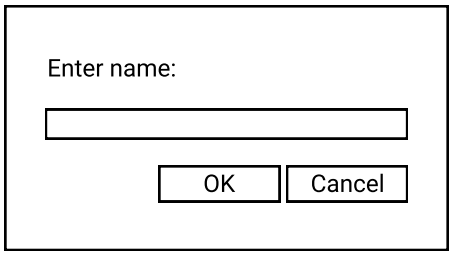
Aplikaci na základě požadavků obsažených v kapitole 2.2.1 rozdělím na tři části. Při návrhu každé části nejdříve shrnu, co musí z hlediska návrhu uživatelského rozhraní obsahovat. Pro správné splnění bakalářské práce je potřeba splnit všechny funkční požadavky zmíněné v kategoriích must have a should have v kapitole 2.2.3. Plánuji splnit i některé funkční požadavky z kategorie could have, navrhnu tedy i požadavky obsažené v této kategorii.



Obrázek 3.1: Návrh projektového adresáře

3.5.1 Projektový adresář

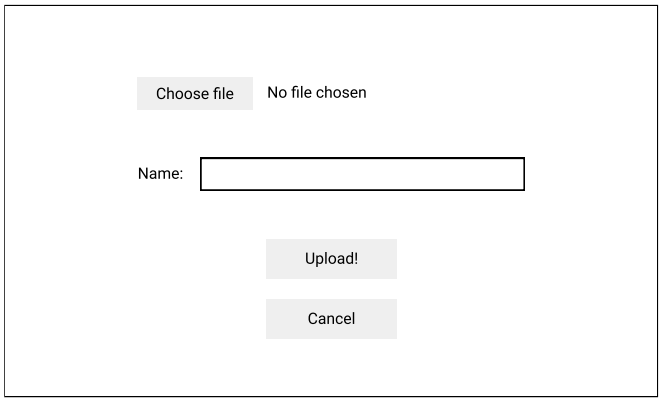
Podle kategorií v kapitole 2.2.3 by projektový adresář měl obsahovat výpis všech Jinja šablon a projektových složek. Tyto šablony a složky by mělo být možné vytvořit, mazat, přesouvat a přejmenovávat. Jinja šablony by navíc mělo být možné nahrávat a stahovat.



Obrázek 3.2: Návrh zvolení názvu

Jinja šablony a projektové složky zobrazím v seznamu. Odlišení, která šablona či složka je v jaké složce, provedu pomocí zvětšeného odsazení. Ke správě Jinja šablon a složek použiji menu s tlačítky. Při tvorbě a přejmenování šablony či složky zobrazím textové políčko pro název šablony či složky s tlačítky pro jeho odeslání nebo zrušení.

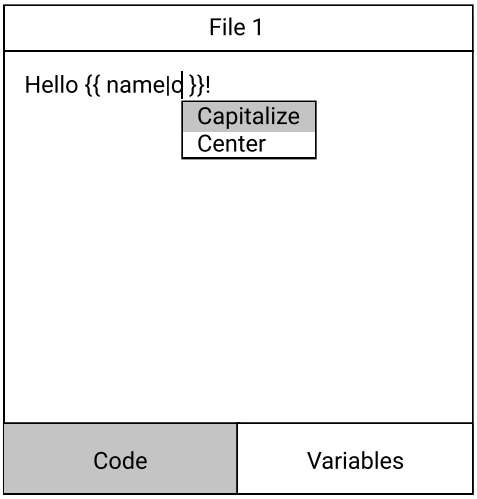
Pro nahrávání Jinja šablony se zobrazí formulář s tlačítkem pro zvolení nahrávané šablony, textovým políčkem pro zvolení jména šablony a tlačítko, po jehož stisknutí se šablona nahraje do aplikace. Pod tímto tlačítkem bude ve formuláři také tlačítko ke zrušení formuláře. K přesouvání Jinja šablon a složek využiji klasický drag and drop.



Obrázek 3.3: Návrh nahrání šablony

3.5.2 Textový editor

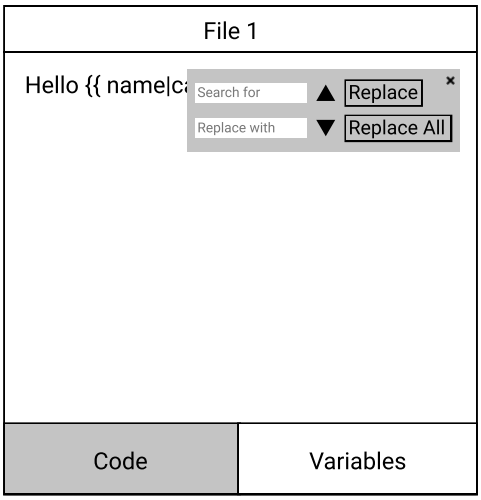
V aplikaci budou dva textové editory, jeden pro editaci Jinja šablon a druhý pro editaci proměnných pro správné vykreslení Jinja šablony.



Obrázek 3.4: Návrh textového editoru

Podle kategorií must have a should have v kapitole 2.2.3 by textový editor Jinja šablon měl podporovat pohyb v textu a jeho vybírání. Vybraný text by mělo jít kopírovat, vkládat a mazat. Dále by měl zvýrazňovat a našeptávat Jinja syntaxi a podporovat operaci find and replace. Z hlediska návrhu uživatelského rozhraní je potřeba jen rozhodnout o vzhledu našeptávání Jinja syntaxe a find and replace.

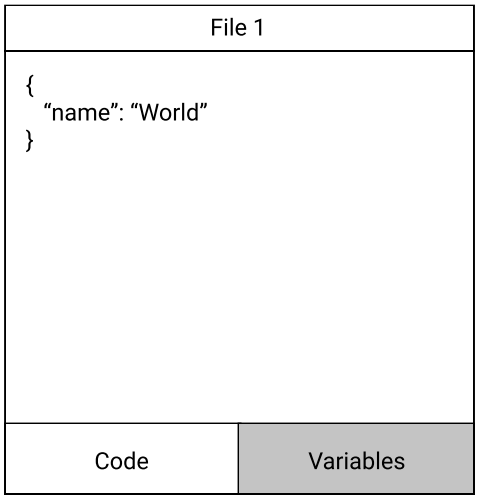
Našeptávání zobrazím jako seznam možností, který se zobrazí na místě kurzoru v editoru. Find and replace zobrazím jako dvě textová políčka pod



Obrázek 3.5: Návrh find and replace v textovém editoru

sebou, horní pro hledaný text a dolní pro jeho náhradu. Po straně bude celkem 5 tlačítek. Dvě z těchto tlačítek pro pohyb mezi nálezy, jedno pro nahrazení zvoleného nálezu a jedno pro nahrazení všech nálezů. Dále tlačítko pro uzavření find and replace.

Z textových editorů bude zobrazen vždy jen jeden, mohou tedy být na stejném místě. K jejich přepínání zvolím záložky v podobě dvou tlačítek.

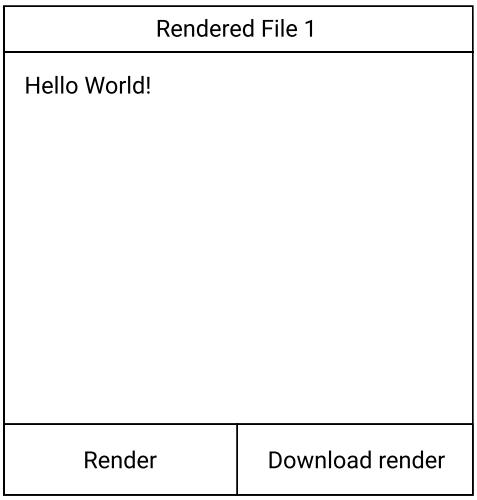


Obrázek 3.6: Návrh editoru proměnných

3.5.3 Vykreslení šablony

V této části aplikace má být vykreslená šablona v textovém formátu a možnost pro stažení vykreslené šablony.

Vykreslenou šablonu v textovém formátu zobrazím jako textový editor, který nebude možné upravovat. Pro stažení vykreslené šablony zvolím jednoduché tlačítko, vzhledem stejné jako tlačítka pro záložky z návrhu textového editoru. Také nesmí chybět tlačítko, po jehož stisknutí se šablona vykreslí.



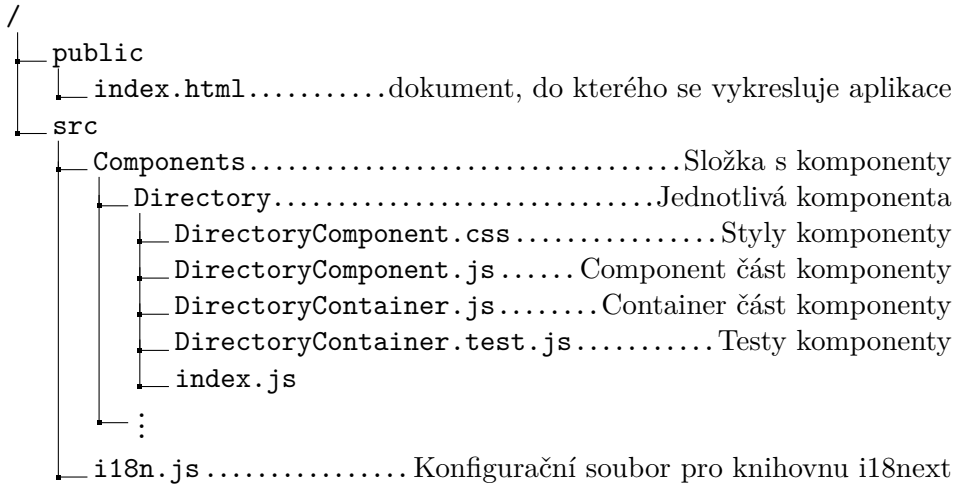
Obrázek 3.7: Návrh vykreslení šablony

Kapitola 4

Realizace

4.1 Frontend

V této kapitole popíši strukturu zdrojového kódu a implementaci funkčností aplikace dle požadavků v kapitole 2.2.3.



Obrázek 4.1: Struktura frontend části aplikace

4.1.1 Struktura

Frontend je rozdělen do komponent dle zvolené architektury. Tyto komponenty jsou rozděleny na soubory Container a Component. Zdrojové kódy komponent jsou strukturovány do složek společně s jejich testy a styly. U každé komponenty je také soubor `index.js`, který pouze exportuje Container dané komponenty. Díky tomu jsou importy těchto komponent čitelnější.

Takto strukturovaný projekt následuje vzor Directory per Component [28],

ten nabízí mnoho pozitiv. Programátor, který na projektu nikdy nepracoval, se v projektu rychle zorientuje a neměl by mít problém najít část kódu, kterou hledá. Zároveň je možné komponenty lehce využít v jiném projektu či je celé vyměnit za jiné.

```
localStorageSafeSetItem(key, value){
    try{
        localStorage.setItem(key, value);
        return true;
    } catch(error){
        if(this.isQuotaExceeded(error)){
            this.addNotification(i18n.t('notificationLabels:app
            ↪  error'),
            ↪  i18n.t('app:notifications.errors.storage
            ↪  full'), 'danger', 5000);
        }
        else {
            this.addNotification(i18n.t('notificationLabels:app
            ↪  error'), i18n.t(
            ↪  'app:notifications.errors.localstorage'),
            ↪  'danger', 5000);
        }
        return false;
    }
}
```

Ukázka kódu 1: Funkce k bezpečnému ukládání do localStorage

4.1.2 App

Komponentu App používám jako obal pro vykreslení hlavních částí aplikace – projektového adresáře, textového editoru a vykreslení šablony.

Některé části aplikace potřebují stejná data. Například proměnné zvolené šablony jsou potřeba jak v textovém editoru (pro jejich editaci), tak ve vykreslení šablony (pro zaslání na server společně s textem zvolené šablony k jejímu vykreslení). Kvůli nevyužití Reduxu musím tato data spravovat v komponentě App a předávat je částem aplikace. Společně s těmito daty předávám také funkce k jejich úpravám. Využití výchozí správy stavu aplikace již obsažené v Reactu je oproti Reduxu jednodušší a plně dostačující pro tuto aplikaci.

Mezi společná data dále patří struktura projektového adresáře, id, název a text vybrané šablony. Tato data při jejich změně či zavírání stránky ukládám na localStorage. V případě chyby při ukládání upozorním uživatele notifikací.

4.1.3 Projektový adresář

Tuto část aplikace jsem implementoval v komponentě Directory. V komponentě přijímám strukturu projektového adresáře a pomocí rekurzivní komponenty DirectoryNode ji vykresluji. Složky ve vykreslené struktuře lze otevírat a zavírat. Při zavírání aplikace ukládám do localStorage seznam otevřených složek, aby je uživatel nemusel otevírat znovu při dalším použití aplikace.

Ke správě projektového adresáře jsem implementoval následující funkčnosti.

4.1.3.1 Vytváření Jinja šablon a projektových složek

Při vytváření Jinja šablony či nové složky žádám uživatele o zadání jejího názvu. Tento název kontroluji, zda je platný. Nejdříve kontroluji název pomocí balíčku valid-filename [29]. Poté kontroluji, jestli již neexistuje stejně pojmenovaná Jinja šablona či složka na stejném místě. Pokud je název platný, tak Jinja šablonu či složku vložím do projektového adresáře. V případě jakéhokoliv problému (například neplatný název) upozorním uživatele notifikací.

```
1  deleteNode(files, nodeId){
2      if(typeof files[nodeId] === 'undefined'){
3          return;
4      }
5      while(typeof files[nodeId].children !== 'undefined'
6      && files[nodeId].children.length){
7          this.deleteNode(files, files[nodeId].children[0]);
8      }
9      if(nodeId === this.props.selectedNodeId){
10         this.props.resetSelectedNode();
11     }
12     if(nodeId !== 'node0'){
13         localStorage.removeItem(nodeId);
14         files[files[nodeId].parent].children.splice(
           ↪  files[files[nodeId].parent].children.indexOf(
           ↪  nodeId),
           ↪  1);
15         delete files[nodeId];
16     }
17 }
```

Ukázka kódu 2: Funkce k odstranění Jinja šablon a projektových složek

4.1.3.2 Odstranění Jinja šablon a projektových složek

K odstranění Jinja šablon a projektových složek využívám jednoduchou rekurzivní funkci. V případě odstranění složky spustím stejnou rekurzi na všechny její potomky. Jinja šablony odstraním jak z projektového adresáře, tak z localStorage, aby zbytečně nezabíraly místo.

V JavaScriptu jsou proměnné s objekty jen reference na místo v paměti, kde je objekt uložen. Při jejich předávání do funkcí se předává pouze tato reference a při volání se vytváří kopie reference. Díky tomu mohu přímo měnit předanou strukturu adresáře. Tato změna se projeví v komponentě App a po změně stavu komponenty App i v dalších částech aplikace.

4.1.3.3 Nahrání Jinja šablon

Při nahrání Jinja šablony nejdříve kontroluji název stejně, jako v případě vytváření nové Jinja šablony. Poté šablonu uložím do localStorage a zobrazím v projektovém adresáři.

```
1  downloadNode(targetId){
2      if(typeof this.props.files[targetId] === 'undefined'){
3          this.props.addNotification(i18n.t(
           ↪  'notificationLabels:app error'),
           ↪  i18n.t('directory:notifications.error.file
           ↪  unknown'), 'danger');
4      }
5      else if(typeof this.props.files[targetId].children !==
       ↪  'undefined'){
6          let children = this.props.files[targetId].children;
7          for(let i = 0; i < children.length; i++){
8              this.downloadNode(children[i]);
9          }
10     }
11     else if(typeof this.props.files[targetId].name !==
       ↪  'undefined'){
12         this.props.downloadText(this.getNodeText(targetId),
           ↪  this.props.files[targetId].name);
13     }
14 }
```

Ukázka kódu 3: Funkce ke stažení Jinja šablon

4.1.3.4 Stažení Jinja šablon

Ke stažení Jinja šablon jsem napsal rekurzivní funkci, která očekává id Jinja šablony či projektové složky. Pokud předané id odkazuje na Jinja šablonu, tak konvertuji text šablony na data URL a s pomocí HTML anchor elementu spustím jeho stažení. V případě předané id projektové složky spustím stejnou rekurzivní funkci pro všechny potomky této složky.

4.1.4 Textový editor Jinja šablon

Textový editor Jinja šablon je implementován v komponentě Code. V této komponentě vykresluji editory textu a proměnných vybrané Jinja šablony s možností mezi nimi přepínat.

K editoru textu Jinja šablony jsem kvůli požadavku **F2.1** nemohl využít pouze HTML textarea element, jelikož nepodporuje změnu barev jednotlivých slov. Textarea element jsem schoval a využívám ho k přijímání vstupu od uživatele. Upravovaný text, společně s textovým kurzorem, zobrazuji samostatně. K editaci textu vybrané Jinja šablony jsem implementoval následující funkčnosti.

```
1  keyDownLeft(event){
2      event.preventDefault();
3      let newSelection = this.getSelectionStart();
4      let newLineId = this.selectedLineId;
5
6      //if cursor is at the start of line and there is another
       ↪  line above selected
7      //   move to above line and set cursor to end of line
8      //else move cursor to left
9      if(newSelection === 0 && newLineId !== 0){
10         newLineId--;
11         newSelection = this.props.lines[newLineId].text.length;
12     }
13     else if(newSelection !== 0){
14         newSelection--;
15     }
16     this.keyDownSetCursor(newLineId,
       ↪  this.props.lines[newLineId].text, newSelection);
17 }
```

Ukázka kódu 4: Funkce ke zpracování zmáčknutí levé šipky

4.1.4.1 Pohyb v textu pomocí myši i klávesnice

Textový kurzor je možné přesunout kliknutím na požadované místo nebo následujícími klávesy.

Šipky nahoru a dolů pohybují textový kurzor o jeden řádek nahoru či dolů. Když je kurzor na prvním či posledním řádku, tak se přesune na začátek či konec řádku.

Šipky doleva a doprava pohybují textový kurzor o jeden znak doleva či doprava. Když je kurzor na začátku či konci řádku, tak se přesune o jeden řádek nahoru či dolů.

Home přesune textový kurzor na začátek řádku.

End přesune textový kurzor na konec řádku.

Page Up a Page Down přesune textový kurzor o jednu stránku nahoru či dolů.

4.1.4.2 Vybírání textu pomocí myši i klávesnice

Text lze vybrat pomocí táhnutí myši nebo pomocí držení klávesy Shift a použití jakékoliv klávesy k pohybu v textu. Vybraný text zvýrazňuji a vkládám do schovaného HTML textarea elementu. Díky tomu byla implementace funkčností s vybraným textem lehčí.

4.1.4.3 Odstranění vybraného textu

Vybraný text lze přepsat nebo zcela smazat použitím jakékoliv alfanumerické klávesy.

4.1.4.4 Kopírování vybraného textu

Díky využití HTML textarea elementu lze zvolený text jednoduše zkopírovat pomocí klávesové zkratky Ctrl + C. Zvolený text lze také zkopírovat pomocí kontextového menu po kliknutí pravým tlačítkem. Pro zobrazení správného kontextového menu po kliknutí pravým tlačítkem na chvíli zobrazím schovaný textarea element.

4.1.4.5 Vkládání zkopírovaného textu

Zkopírovaný text lze vložit pomocí klávesové zkratky Ctrl + V nebo kliknutím pravým tlačítkem a zvolením požadované možnosti v kontextovém menu podobně jako při kopírování vybraného textu.

4.1.4.6 Zvýraznění Jinja syntaxe zadanými barvami

Výrazy v Jinja syntaxi se dají rozdělit do tří kategorií podle jejich oddělovačů. Při zobrazení textu vybrané Jinja šablony kontroluji každý řádek, zda neobsahuje výraz z těchto kategorií. Když nějaký výraz obsahuje, tak ho zvýrazním barvou dané kategorie.

```
1  addCodeCompletionLines(keywordArray, writtenWord, className){
2      // adds all jinja keywords from keywordArray that start
       ↪  with writtenWord
3      let resultArr = [];
4      if(writtenWord){
5          keywordArray.forEach(keyword => {
6              if(keyword.indexOf(writtenWord) === 0 &&
               ↪  writtenWord.length < keyword.length){
7                  resultArr.push({'className': className, 'word':
                   ↪  keyword});
8              }
9          });
10     }
11     return resultArr;
12 }
```

Ukázka kódu 5: Funkce k nalezení vhodných Jinja výrazů pro našeptávání

4.1.4.7 Našeptávání Jinja syntaxe při psaní

Pro zobrazení našeptávání je potřeba, aby uživatel psal mezi Jinja oddělovači. Při každé změně textu mezi těmito oddělovači hledám, a následně zobrazuji, všechny výrazy, které začínají na zatím napsaný výraz a jsou v kategorii určené podle napsaných oddělovačů. Našeptávaný výraz lze zvolit pomocí šipek a klávesy Enter nebo pomocí kliknutím myši. Po jeho zvolení ho doplním.

4.1.5 Vykreslení šablony

Tato část aplikace umožňuje zvolenou Jinja šablonu vykreslit a následně její vykreslení zobrazit. Vykreslenou Jinja šablonu lze také stáhnout. Zobrazení této části aplikace je implementováno v komponentě Render a samotná implementace je popsaná v následujících podkategoriích.

4.1.5.1 Vykreslení šablony v textovém formátu

Pro vykreslení zvolené Jinja šablony odesílám požadavek na backend. Implementace přijmutí tohoto požadavku a odeslání odpovědi je popsaná v imple-

mentaci backendu. Součástí tohoto požadavku musí být proměnné a cesta ke zvolené Jinja šabloně. Dále požadavek musí obsahovat texty všech Jinja šablon v projektu, z důvodu možného využití dědičnosti.

V případě jakékoliv chyby (například špatného formátu proměnných zvolené šablony) před odesláním požadavku na vykreslení upozorním uživatele notifikací. V případě chyby při vykreslení na backendu zobrazím vrácenou chybu místo vykreslené šablony.

```
1  getTemplates(files = this.props.files['node0'].children,
   ↪  templates = {}, path = ''){
2      for(let key of files){
3          if(this.props.files[key].children){
4              templates = this.getTemplates(
               ↪  this.props.files[key].children, templates, path
               ↪  + this.props.files[key].name + '/');
5          }
6          else {
7              templates[path + this.props.files[key].name] =
               ↪  this.getTemplateData(key);
8          }
9      }
10     return templates;
11 }
```

Ukázka kódu 6: Funkce k získání textů všech Jinja šablon v projektu

4.1.5.2 Stažení vykreslené šablony

Ke stažení vykreslené Jinja šablony využívám data URL a HTML anchor element stejně jako v případě stažení Jinja šablony v projektovém adresáři.

4.2 Backend

Dle požadavků zmíněných v kapitole 2.2.1 by backend měl vykreslovat Jinja šablony a ukládat Jinja šablony a projektové složky na cloud. Ukládání Jinja šablon a projektových složek na cloud jsem v kapitole 2.2.3 zařadil do kategorie Won't have. V této kapitole se tedy zaměřím jen na implementaci vykreslení Jinja šablon.

4.2.1 Struktura

Struktura backend části aplikace je velmi jednoduchá. Celá aplikace je v souboru `app.py` a má jen jeden endpoint. Dále projekt obsahuje soubory pro testování, nasazení a nainstalování aplikace.

```
/
├── app.py ..... Aplikace
├── app_test.py ..... Testy aplikace
├── requirements.txt ..... Požadavky pro instalaci aplikace
├── setup.py ..... Konfigurační soubor pro instalaci aplikace
└── vercel.json ..... Konfigurační soubor pro nasazení aplikace
```

Obrázek 4.2: Struktura backend části aplikace

4.2.2 Vykreslení šablony

Jinja podporuje dědičnost šablon a vykreslení proměnných, kvůli tomu potřebuji ke správnému vykreslení přijímat všechny napsané Jinja šablony, cestu ke zvolené šabloně a její proměnné. Tato data přijímám ve formátu JSON.

Uživatelé budou moci k vykreslení posílat jakékoliv šablony chtějí, pro zamezení zneužití nebezpečných atributů a metod jsem využil Jinja sandbox. Jinja sandbox vykresluje šablony izolovaně a v případě pokusu o přístup k nezabezpečenému kódu vrátí chybu.

Šablony k vykreslení lze načítat mnoha způsoby, Jinja nabízí například načítání ze souborů, Python slovníku nebo funkce. Načítání šablon je také možné napsat vlastní. Vybral jsem načítání již obsažené v Jinja s názvem DictLoader. Toto načítání přijímá seznam jmen šablon vázaných na jejich text, ve kterém poté žádané šablony hledá. Díky tomu není nutné přijaté šablony na serveru ukládat.

Na požadavek o vykreslení Jinja šablony ji vracím vykreslenou společně se stavovým kódem 200. V případě chyby při vykreslení vracím chybovou hlášku se stavovým kódem 400. Taková chybová hláška obsahuje i cestu k šabloně a řádek, na kterém chyba nastala. V případě jakékoliv jiné chyby vracím obecnou chybovou hlášku se stavovým kódem 500.

4.3 Uživatelské rozhraní

Při implementaci funkčností jsem využil návrh uživatelského rozhraní popsaný v kapitole 3.5. Uživatelské rozhraní jsem implementoval v částech Component a pro opakující se elementy jsem udělal nové komponenty (například Button a DirectoryNode). Konečný vzhled aplikace je představen v kapitole 6.

```
post_data = request.get_json()
if not validate_data(post_data):
    message = 'Error: wrong json structure'
    status_code = 400
else:
    env = SandboxedEnvironment(
        loader=jinja2.DictLoader(post_data['templates']),
        autoescape=jinja2.select_autoescape(['html', 'xml'])
    )
    template = env.get_template(post_data['selected'])
    message = template.render(post_data['context'])
```

Ukázka kódu 7: Část funkce k vykreslení přijaté Jinja šablony

Kapitola 5

Testování a nasazení

5.1 Testování kódu

Při vývoji aplikace se může vyskytnout mnoho chyb, proto jsem během vývoje napsal unit testy pro automatické testování napsaného kódu. Tyto testy se zaměřují na testování malých částí aplikace. Pro správné napsání unit testů jsem využil mocking k oddělení testované části aplikace od jejího zbytku.

5.1.1 Frontend

Ke psaní unit testů pro frontent část aplikace jsem použil framework Jest spolu s balíčkem react-test-renderer, který využívám k vykreslení testovaných React komponent. Soubory s testy jsou umístěny ve složce s testovanou komponentou. Po této komponentě jsou také pojmenovány (viz obrázek 4.1).

Testy funkcí jsou strukturovány do describe bloků. Napsané unit testy mimo jiné kontrolují výstup testované funkce, změnu stavu aplikace a volání ostatních funkcí.

5.1.2 Backend

Na backendu jsem napsal unit testy pomocí frameworku Unittest. Testy jsem podle testované funkce strukturoval do tříd. Soubor s testy je pouze jeden, a to přímo v kořenovém adresáři. Při psaní unit testů jsem mimo jiné kontroloval výstup testovaných funkcí a volání ostatních funkcí.

```
    class TestValidateData(unittest.TestCase):

    def test_invalid_structure(self):
        json_arr = [
            {},
            {
                'selected': ''
            },
            {
                'selected': '',
                'context': ''
            },
            {
                'selected': '',
                'context': '',
                'templates': ''
            },
            {
                'selected': 'test',
                'context': '',
                'templates': {
                    'test2': ''
                }
            }
        ]
        for json in json_arr:
            with self.subTest(msg='wrong json structure',
            ↪  json=json):
                self.assertFalse(validate_data(json))
```

Ukázka kódu 8: Část unit testu validace struktury přijatých dat

5.2 Uživatelské testování

K uživatelskému testování jsem využil webovou službu Loop11 [30], ve které lze vytvářet scénáře s úkoly a otázky. Aplikace byla pomocí této služby otestována osmi uživateli na různých operačních systémech i prohlížečích.

Použité operační systémy byly jmenovitě Windows 10, Mac OSX 10.15, Manjaro a Arch Linux. Mezi použitými prohlížeči byly Mozilla Firefox verze 88.0 a Google Chrome verze 90.0.4430.85 a 90.0.4430.93.

5.2.1 Scénáře

Na začátku testování byl respondent obeznámen s funkcí testované aplikace a základní syntaxí Jinja šablon. Poté mu byly předloženy následující scénáře.

5.2.1.1 Vytvoření a vykreslení šablony

Vytvořte složku s názvem ‘templates‘, v ní vytvořte novou šablonu s názvem ‘template.html‘. V této šabloně nastavte:

1. kód šablony

   {{ jinja }}

2. proměnné šablony

   { ”jinja”: ”Hello {{ name|capitalize }}! This is {{ pageName }”}

Proměnné pouze zkopírujte a vložte místo již přednastavených, pokud jste v nich našli nějakou chybu, tak ji zatím nijak **neopravujte**. Tuto šablonu vykreslete a stáhněte vykreslenou. Měl by se Vám vykreslit následující text:

Hello {{ name|capitalize }}! This is {{ pageName }

5.2.1.2 Nahrání šablony

Vytvořte novou složku s názvem “upload”. Pokud jste v předchozím úkole nestáhli vykreslenou šablonu “template.html” tak ji stáhněte nyní. Poté ji nahrajte do složky “upload”. Do proměnných nahrané šablony doplňte následující proměnné:

{”name”: ”world”, ”pageName”: ”Jinja editor”}

5.2.1.3 Opravení chyby při vykreslení šablony

Vykreslete šablonu “rendered template.html” nahranou v předchozím úkole a zkontrolujte její vykreslení. V případě chyby při vykreslení se ji pokuste opravit.

5.2.2 Otázky

Po provedení scénářů byly respondentům položeny následující otázky.

5.2.2.1 Povedlo se Vám vyřešit chybu při vykreslení šablony v posledním úkole? Pokud ano, jak? Byla chybová hláška dostatečně specifická?

Ve vykreslované šabloně v posledním scénáři úmyslně chyběla uzavírací závorka }. Jedni z prvních respondentů chybu opravili již v prvním scénáři. Do prvního scénáře jsem proto doplnil, aby respondenti případně nalezenou chybu neopravovali. Všichni po přečtení chybové hlášky úspěšně chybu opravili.

5.2.2.2 Ohodnoťte prosím intuitivnost webové aplikace.

V této otázce měl respondent ohodnotit intuitivnost webové aplikace. Na výběr měl mezi čtyřmi odpověďmi: velmi intuitivní, spíše intuitivní, spíše neintuitivní a velmi neintuitivní. Většina zvolila odpověď spíše intuitivní.

5.2.2.3 Měli jste v průběhu plnění úkolů nějaký problém? Pokud ano, popište prosím daný problém.

V této části si respondenti převážně stěžovali na nepřehlednost ovládání projektového adresáře. Při vytváření nových složek a souborů si všimli pouze vypsaných ovládacích prvků v horní části projektového adresáře. Tyto ovládací prvky se poté snažili použít i pro vytváření a nahrávání souborů do složek. Jeden respondent v návaznosti na toto také zmiňoval nemožnost přesunu souborů a složek.

5.2.2.4 Co se Vám na aplikaci líbilo nejvíce?

Zde respondenti vychválili použití tmavých barev, notifikací a jednoduchého uživatelského rozhraní bez spousty tlačítek.

5.2.2.5 S čím byl největší problém?

V této otázce respondenti znovu převážně vytknuli nepřehlednost ovládání projektového adresáře.

5.2.3 Shrnutí

Při testování se přišlo na několik menších chyb a nedostatků. Mnoho z nich bylo opraveno, většinou ještě před tím, než mohl aplikaci otestovat další respondent. Mezi nalezené a opravené nedostatky patří například:

- Vytvořená složka je automaticky zavřená
- Po kliknutí pravým tlačítkem na složku či šablonu se nezobrazí menu s nabídkou k této složce či šabloně
- Nepřehlednost vytváření nové složky či šablony

5.3 Nasazení

K nasazení aplikace jsem využil bezplatnou službu Vercel [31], která samotné nasazení řeší za mě. Službě Vercel jsem zpřístupnil moje GitHub repositáře frontend i backend části aplikace. Díky tomu dokáže Vercel automaticky udržovat nasazenou aplikaci na nejnovější verzi.

Pro správné nasazení backend části aplikace jsem napsal konfigurační soubor. V tomto souboru udávám, o jaký typ aplikace se jedná a ve kterém souboru se aplikace nachází. Před nasazením frontend části aplikace se spustí její unit testy, v případě selhání se nasazení neprovede. Ke správnému fungování nasazené frontend části aplikace jsem vložil adresu na nasazenou backend část aplikace do proměnné prostředí.

```
describe('RenderContainer::handleDownloadButton', () => {
    let mockDownloadText = jest.fn();
    const renderContainer = renderer.create(<RenderContainer
        downloadText={mockDownloadText}
    />);

    beforeEach(() => {
        jest.clearAllMocks();
    });

    test('', () => {
        renderContainer.getInstance().setState({
            renderedText: 'renderedText',
            renderedName: 'renderedName'
        })
        return flushPromises().then(() => {
            renderContainer.getInstance(
            ↪ ).handleDownloadButton({});
            expect(mockDownloadText).toBeCalledWith(
            ↪ 'renderedText', 'rendered
            ↪ renderedName');
        });
    })
})
```

Ukázka kódu 9: Unit test stažení vykreslené šablony

Kapitola 6

Prototyp

Po načtení webové aplikace se uživateli zobrazí projektový adresář, editor Jinja šablon a vykreslení zvolené Jinja šablony.

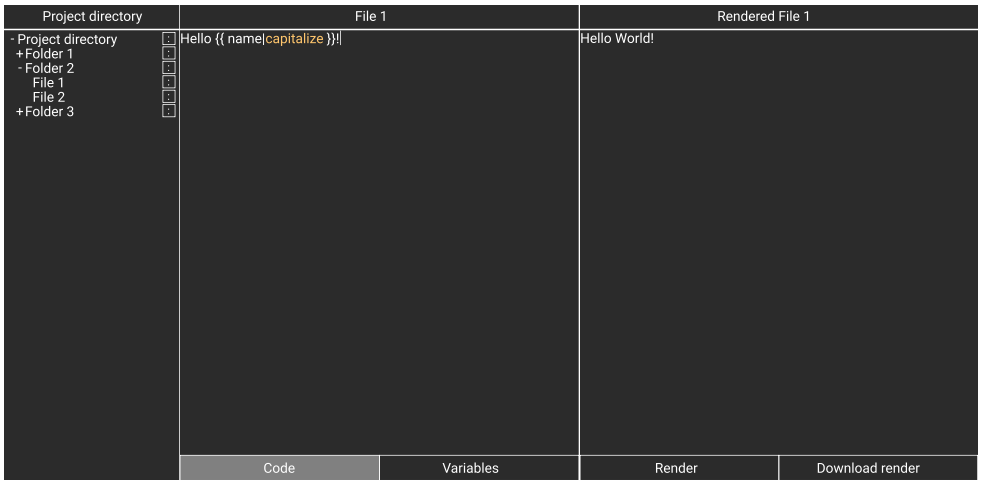
Při prvním použití této aplikace má uživatel prázdný projektový adresář, nemá tedy žádnou Jinja šablonu přístupnou k editaci ani k vykreslení. Editor a část aplikace pro zobrazení vykreslené šablony jsou kvůli tomu zašedlé a je zobrazen text, který na toto upozorňuje.

Jinja šablony je možné vytvořit v projektovém adresáři nebo do adresáře nahrát pomocí formuláře. Po vytvoření Jinja šablon je možná jejich úprava a vykreslení.

Zvolená šablona se otevře v editoru, kde se zobrazí její název a text. Proměnné zvolené šablony je možné zobrazit a následně upravit po přepnutí na editor proměnných. Vykreslená šablona se zobrazí v pravé části aplikace.

Obrázek 6.1: Aplikace při prvním zapnutí

Obrázek 6.2: Formulář k nahrání šablony



Obrázek 6.3: Zobrazení vykreslené šablony

Závěr

V rámci bakalářské práce jsem vytvořil webovou aplikaci pro editování Jinja šablon. Aplikace je dostupná na adrese www.jinjaeditor.eu.

Při tvorbě bakalářské práce jsem postupoval podle vypsaných cílů v kapitole 1. Hlavním cílem bylo vytvoření webového editoru Jinja šablon. Mezi dílčí cíle patřilo provedení stručné analýzy obdobných webových editorů, sestavení funkčních a nefunkčních požadavků, vytvoření návrhu architektury a UI aplikace a následná implementace a otestování aplikace.

Nejdříve jsem provedl stručnou rešerši webových editorů Jinja šablon a programovacího jazyka JavaScript. Z této rešerše a z požadavků vedoucího práce jsem sestavil funkční a nefunkční požadavky na aplikaci, ty jsem poté rozdělil do čtyř kategorií podle metody MoSCoW. Pomocí sestavených požadavků jsem zvolil používané technologie a také navrhl architekturu a UI aplikace. Při návrhu jsem se zaměřil na budoucí rozšiřitelnost a uživatelskou přívětivost. Podle návrhu jsem aplikaci implementoval, otestoval a nasadil. Cíle tedy shledávám za splněné.

Webová aplikace obsahuje projektový adresář, textový editor a vykreslení šablony. Projektový adresář umožňuje Jinja šablony a složky vytvořit a smazat. Jinja šablony lze dále nahrát či stáhnout. Tyto složky a šablony jsou také automaticky ukládány. Textový editor Jinja šablon zvýrazňuje a našeptává Jinja syntaxi a podporuje základní funkce k editaci textu. Mezi tyto funkce patří například vybírání textu a jeho následné odstranění či kopírování. Jinja šablonu lze vykreslit v textovém formátu. Takto vykreslenou šablonu je možné stáhnout.

Všechny požadavky vyplývající ze zadání práce se podařilo úspěšně splnit. Dále se podařilo splnit všechny požadavky zmíněné v kategoriích 1 a 2 kapitoly 2.2.3 a některé požadavky z kategorie 3 kapitoly 2.2.3. Aplikace by šla rozšířit o nesplněné požadavky zmíněné v kapitole 2.2.3 v kategoriích 3 a 4.

Požadavky, o které by šlo aplikaci rozšířit jsou jmenovitě:

F1.2.3 Přesouvání Jinja šablon a projektových složek
F1.2.4 Přejmenování Jinja šablon a projektových složek
F1.2.6 Ukládání Jinja šablon a projektových složek na cloud
F2.2 Změna barev zvýraznění Jinja syntaxe
F2.9 Find and replace
F2.10 Multiple cursor
F2.11 Zobrazení číslování řádků
F3.2 Vykreslení šablony v HTML formátu

Literatura

1. ROGALA, Przemek. *TTL255 J2Live - Live Jinja2 Parser* [online]. 2020 [cit. 2020-12-03]. Dostupné z: `https://j2live.ttl255.com/`.
2. SIQUEIRA, Rodrigo. *Online Javascript Editor* [online] [cit. 2020-12-03]. Dostupné z: `https://js.do/`.
3. HAVERBEKE, Marijn. *CodeMirror* [online] [cit. 2020-12-03]. Dostupné z: `https://codemirror.net/`.
4. P5.JS. *p5.js Web Editor* [online] [cit. 2020-12-03]. Dostupné z: `https://editor.p5js.org/`.
5. PLAYCODE.IO. *PLAYCODE - Javascript Playground* [online] [cit. 2020-12-03]. Dostupné z: `https://playcode.io/`.
6. KUHN, Janet. Decrypting the MoSCoW Analysis. *The workable, practical guide to Do IT Yourself*. 2009, roč. 5.
7. DUNKLEY, Wayne. *Split Stack Development: A Model For Modern Applications | by Future Friendly | Medium* [online]. 2016 [cit. 2021-03-12]. Dostupné z: `https://medium.com/@Future__Friendly/split-stack-development-a-model-for-modern-applications-d7b9abb47bd5`.
8. MDN WEB DOCS. *Document Object Model (DOM) - Web APIs | MDN* [online] [cit. 2021-03-16]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model`.
9. GOOGLE. *Angular* [online] [cit. 2021-03-13]. Dostupné z: `https://angular.io/`.
10. FACEBOOK INC. *React – A JavaScript library for building user interfaces* [online] [cit. 2021-03-13]. Dostupné z: `https://reactjs.org/`.
11. VUE. *Vue.js* [online] [cit. 2021-03-13]. Dostupné z: `https://vuejs.org/`.
12. FACEBOOK INC. *Jest · Delightful JavaScript Testing* [online] [cit. 2021-03-04]. Dostupné z: `https://jestjs.io/`.

13. AVA. *GitHub - avajs/ava: Node.js test runner that lets you develop with confidence* [online] [cit. 2021-03-04]. Dostupné z: `https://github.com/avajs/ava`.
14. OPENJS FOUNDATION. *Mocha - the fun, simple, flexible JavaScript test framework* [online] [cit. 2021-03-04]. Dostupné z: `https://mochajs.org/`.
15. JAN MÜHLEMANN ET AL. *Introduction - react-i18next documentation* [online] [cit. 2021-03-05]. Dostupné z: `https://react.i18next.com/`.
16. FORMATJS. *Overview | Format.JS* [online] [cit. 2021-03-05]. Dostupné z: `https://formatjs.io/docs/react-intl/`.
17. DJANGO SOFTWARE FOUNDATION. *The Web framework for perfectionists with deadlines | Django* [online] [cit. 2021-03-07]. Dostupné z: `https://www.djangoproject.com/`.
18. HELLKAMP, Marcel. *Bottle: Python Web Framework — Bottle 0.13-dev documentation* [online] [cit. 2021-03-07]. Dostupné z: `https://bottlepy.org/docs/dev/`.
19. PYLONS PROJECT. *Welcome to Pyramid, a Python Web Framework* [online] [cit. 2021-03-07]. Dostupné z: `https://trypyramid.com/`.
20. THE PALLETS PROJECTS. *Welcome to Flask — Flask Documentation (1.1.x)* [online] [cit. 2021-03-07]. Dostupné z: `https://flask.palletsprojects.com/en/1.1.x/`.
21. PYTHON SOFTWARE FOUNDATION. *unittest — Unit testing framework — Python 3.9.2 documentation* [online] [cit. 2021-03-08]. Dostupné z: `https://docs.python.org/3/library/unittest.html`.
22. KREKEL, Holger. *pytest: helps you write better programs — pytest documentation* [online] [cit. 2021-03-08]. Dostupné z: `https://docs.pytest.org/en/stable/`.
23. REENSKAUG, Trygve Mikjel H. The original MVC reports. 1979.
24. FACEBOOK INC. *Flux | Flux* [online] [cit. 2021-02-23]. Dostupné z: `https://facebook.github.io/flux/`.
25. ABRAMOV, Dan. *Redux - A predictable state container for JavaScript apps. | Redux* [online] [cit. 2021-02-23]. Dostupné z: `https://redux.js.org/`.
26. TUTORIALS POINT. *Component-Based Architecture - Tutorialspoint* [online]. 2016 [cit. 2020-12-03]. Dostupné z: `https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm`.
27. JETBRAINS. *PhpStorm: The Lightning-Smart IDE for PHP Programming by JetBrains* [online] [cit. 2021-03-17]. Dostupné z: `https://www.jetbrains.com/phpstorm/`.

28. VEPSÄLÄINEN, Juho. SurviveJS - React From apprentice to master. 2016.

29. SORHUS, Sindre. *valid-filename - npm* [online] [cit. 2021-02-18]. Dostupné z: `https://www.npmjs.com/package/valid-filename`.

30. LOOP11. *Online User Testing Made Easy | Loop11* [online] [cit. 2021-05-01]. Dostupné z: `https://www.loop11.com/`.

31. VERCEL INC. *Develop. Preview. Ship. For the best frontend teams – Vercel* [online] [cit. 2021-04-07]. Dostupné z: `https://vercel.com/`.

32. THE ECONOMIC TIMES. *What is Authentication? Definition of Authentication, Authentication Meaning - The Economic Times* [online] [cit. 2021-03-20]. Dostupné z: `https://economictimes.indiatimes.com/definition/authentication`.

33. SINCLAIR, John. *Collins COBUILD Advanced Dictionary of English*. Collins Cobuild, 2015.

34. CAPSTONE EDITING. *Find and Replace in Word: When, why and how to use it* [online] [cit. 2021-03-20]. Dostupné z: `https://www.capstoneediting.com.au/blog/what-is-find-and-replace-and-why-you-should-be-using-it`.

35. OBASEKI, Nosa. *localStorage in JavaScript: A complete guide - LogRocket Blog* [online] [cit. 2021-04-26]. Dostupné z: `https://blog.logrocket.com/localstorage-javascript-complete-guide/`.

36. JUNG, June. *How to test software: mocking, stubbing, and contract testing* [online]. 2019 [cit. 2021-03-10]. Dostupné z: `https://circleci.com/blog/how-to-test-software-part-i-mocking-stubbing-and-contract-testing/`.

37. REITAN, Erik. *URL Routing | Microsoft Docs* [online] [cit. 2021-03-20]. Dostupné z: `https://docs.microsoft.com/en-us/aspnet/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/url-routing`.

Příloha A

Seznam použitých zkratek

API Application Programming Interface.

CSS Cascading Style Sheets.

DOM Document Object Model.

HTML Hypertext Markup Language.

IDE Integrated Development Environment.

MPA Multi Page Application.

MVC Model View Controller.

ORM Objektově relační mapování.

SPA Single Page Application.

UI User interface.

XSS Cross-site scripting.

Příloha B

Slovník

autentizace Autentizace je proces rozpoznávání identity uživatele. Jedná se o mechanismus přiřazení příchozího požadavku k sadě identifikačních údajů [32] (překlad Bc. Tomáš Halama).

drag and drop Drag and drop je metoda přesouvání počítačových souborů či obrázků z jednoho místa do druhého pomocí kliknutí a přetažení myši přes obrazovku [33] (překlad Nguyen Quynh Chi).

find and replace Find and replace je funkce, která umožňuje vyhledání cílového textu a jeho nahrazení za jiný text [34] (překlad Ilona Andriyashyn).

localStorage localStorage je vlastnost, která umožňuje JavaScript stránkám a aplikacím ukládat páry klíč-hodnota ve webovém prohlížeči [35] (překlad Bc. Hana Fukalová).

mocking Mocking je metoda v testování spočívající v tvorbě zástupné verze vnější či vnitřní služby, která může nahradit službu opravdovou [36] (překlad Ondřej Kvapil).

URL routing URL routing umožňuje nakonfigurovat aplikaci tak, aby přijímala adresy URL požadavků, které se nemapují na fyzické soubory [37] (překlad Bc. Iveta Šárfyová).

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF