VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÝ VIDEO EDITOR VYUŽÍVAJÍCÍ FRAMEWORK MLT

WEB BASED VIDEO EDITOR USING MLT FRAMEWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VLADAN KUDLÁČ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Kudláč Vladan**
Program: Informační technologie
Název: **Webový video editor využívající framework MLT**
Web Based Video Editor Using MLT Framework
Kategorie: Softwarové inženýrství

Zadání:

1. Nastudujte základy MLT frameworku a principů zobrazení videa ve webovém prohlížeči (JavaScript, HTML).
2. Najděte a nastudujte existující webové editory videa. Nastudujte aplikace pro nelineární editaci videa. Navrhněte jednoduchý nelineární webový editor videa zejména pro editaci záznamů přednášek.
3. Implementujte editor, otestujte UX a UI.
4. Vyhodnoťte úspěšnost. Navrhněte směry dalšího vývoje.
5. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a část bodu 3 ze zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Cílem této práce je vytvořit nelineární editor videí poskytující uživatelům možnost editovat videa v prostředí internetového prohlížeče bez nutnosti instalace dodatečných programů. Uživatel pracuje s interaktivním grafickým rozhraním a provádí základní úpravy videa. Po dokončení se k projektu vygeneruje seznam operací a ten se předá ke zpracování na školním serveru. Dostupné možnosti úprav i použité prostředky budou voleny tak, aby bylo možné editor použít pro úpravu přednášek.

Abstract

The goal of this thesis is to create non-linear video editor, that would allow users to edit video using only a web browser, without installation of any additional software. User works with interactive graphical interface to do basic edits to video. When the editing is finished, the list of instructions is generated and submitted to proceed on faculty server. Available editing tools will be chosen especially for making tutorials and lectures recordings.

Klíčová slova

MLT framework, JavaScript, HTML, webová aplikace, video editor

Keywords

MLT framework, JavaScript, HTML, web application, video editor

Citace

KUDLÁČ, Vladan. *Webový video editor využívající framework MLT*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Webový video editor využívající framework MLT

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőke, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vladan Kudláč
15. května 2019

Poděkování

Chtěl bych poděkovat vedoucímu mé práce, Ing. Igorovi Szőke, Ph.D., který mě odborně vedl. Dále bych rád poděkoval všem pracovníkům z výzkumné skupiny Speech@FIT, na které jsem se mohl obracet s problémy při řešení této práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvořit webovou aplikaci umožňující uživatelům upravovat videa v internetovém prohlížeči bez nutnosti instalace dodatečných programů nebo rozšíření. Uživatel nahraje videa a obrázky ze svého zařízení nebo z jiné služby a poté pomocí interaktivního grafického rozhraní vytvoří požadované video. Vytvořený projekt bude popsán souborem formátu XML, který zpracuje program *MLT*[1] na školním serveru. Dostupné možnosti úprav budou voleny tak, aby bylo možné editor použít pro tvorbu výukových videí. Editor bude vytvořen tak, aby mohl být použit jako modul pro stávající projekt *prednasky.com*. Systém bude architektury klient-server. Klient bude komunikovat se serverem pomocí zdokumentovaného API, aby mohla být implementace serveru nezávislá na klientovi a mohli v budoucnu vzniknout různé implementace klientů. Klient bude poskytovat grafické uživatelské rozhraní pomocí HTML, CSS a JavaScriptu, server bude obsluhovat požadavky, generovat XML projektů a komunikovat s dalšími serverovými moduly. Díky této práci vznikne editor se svobodnou licencí, jako alternativa k webovým editorům s licencí ve formě předplatného a uzavřeným kódem.

Potřeby jednoduchého online video editoru vznikly při práci na portálu s výukovými videi *Prednasky.com*. Při hledání řešení, které by šlo začlenit do projektu bylo zjištěno, že nic takového na trhu není. Buď bylo řešení placené, nebo bylo zdarma, ale s omezenou funkčností. Ani jedno stávající řešení není s otevřeným zdrojovým kódem, které by licence dovolila použít jako komponentu projektu.

V první kapitole jsou zmíněny a srovnávány významné video editory, zejména v oblasti dostupných funkcí a licence. Dále jsou zmíněny technologie a principy uživatelského rozhraní, které byli inspirovány stávajícími editory. Druhá kapitola je věnována návrhu řešení. Objasňuje cíl práce, způsob, jakým je editor řešený, a vysvětluje zvolené technologie. Ve třetí kapitole jsou rozebírány teoretické základy pro práci s multimédii, na kterých práce staví. Je zde vysvětlen způsob práce s multimediálním frameworkem MLT a možnosti práce s multimédii přímo ve webovém prohlížeči. Ve čtvrté kapitole je popsán způsob použití zvolených technologií, členění a implementace projektu. V této kapitole jsou uvedeny požadavky a způsob nasazení a princip fungování programu. Pátá kapitola je věnována testování uživatelského rozhraní, získávání zpětné vazby a testování kódu. Závěrečná kapitola shrnuje aktuální stav projektu, nastiňuje budoucí směr vývoje a další možné aplikace, které mohou na základě této práce vzniknout.

[1] MLT – sada nástrojů pro multimediální aplikace, https://www.mltframework.org/.

Kapitola 2

Existující řešení

Při návrhu editoru jsem zkoumal existující videoeditory. Vybíral jsem editory známé od Youtuberů, všeobecně známé i na základě doporučení od mého vedoucího práce a známých. Stávající řešení se dají rozdělit do dvou skupin. Desktopové video editory a online video editory. Průzkum existujících řešení je zaměřen zejména na projekty, které mají svobodnou licenci, nebo je lze využívat bezplatně.

2.1 Desktopové editory

Mezi desktopovými videoeditory existuje řada bezplatných řešení s licencemi vhodnými pro další i komerční využití. Rozhraní aplikací bývá podobné, rozsáhlé, pro jednoduché editace využívá uživatel zlomek funkcí. Nevýhodou bývá pomalá křivka učení a nutnost provozovat aplikaci na dostatečně výkonném zařízení.

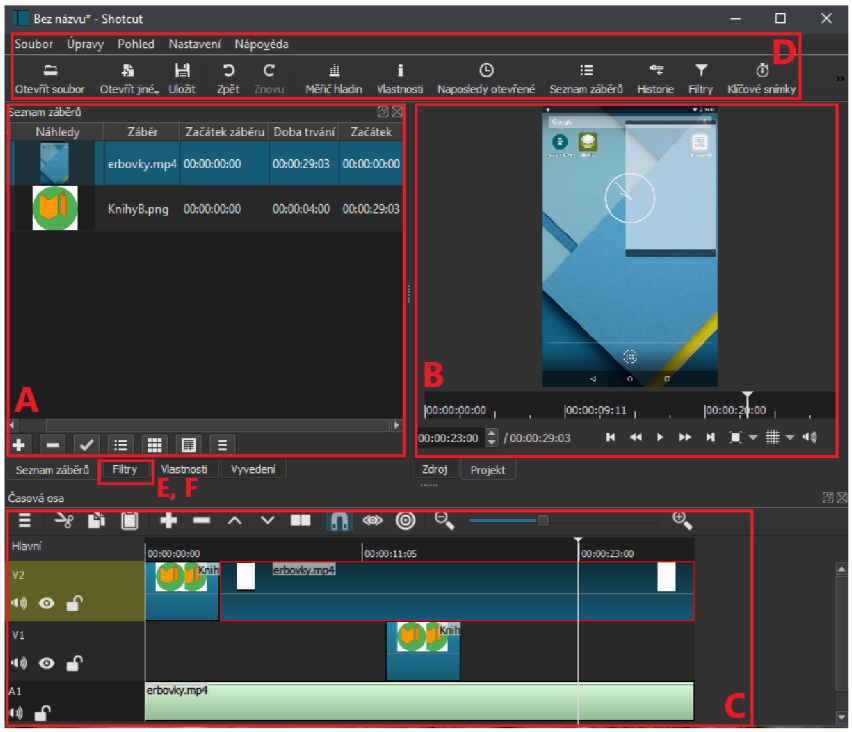
Z řešení s otevřeným zdrojovým kódem lze vyjmenovat například projekty *Shotcut*[1] a *Kdenlive*[2]. Oba projekty používají pro zpracování multimediálních souborů framework MLT. I v bezplatných aplikacích lze vytvořit libovolný projekt, oproti komerčním řešení není uživatel nijak omezován. Při testování bezplatných řešení byly zaznamenány problémy se stabilitou, základem je průběžné ukládání projektu. Rozhraní je u obou programů téměř shodné, obrázek 2.1. Na horní liště nabídek (D) jsou umístěny globální akce projektu, jako otevření projektu, uložení, vpřed, zpět a renderování projektu. Levý sloupec (A) slouží ke správě zdrojů projektu, správě filtrů a přechodů vybrané položky na časové ose a pro nastavování vlastností a parametrů. Pro přepínání zobrazení používají oba programy karty-/záložky. Pravý sloupec (B) zobrazuje náhled zdrojových souborů nebo výsledného videa. Přehrávání náhledu se ovládá prvky pod videem. Vespod je umístěna časová osa (C). Operace s časovou osou jsou umístěny nad osou. Aktuální pozici vizualizuje svislá čára. Osa obsahuje libovolné množství stop. Zvukové stopy mají zelenou barvu, video stopy barvu modrou.

Bezplatnou aplikací nevyužívající framework MLT je například program *Openshot*[3], který přešel na vlastní jádro. Jeho rozhraní je oproti předchozím dvou editorům velmi zjednodušené. Rozvržení zůstává stejné, pouze omezuje počet informací, které uživateli poskytuje. Nahoře se opět nacházejí akce k projektu, vlevo knihovna souborů, filtrů a přechodů,

[1] Shotcut – bezplatný nelineární multiplatformní editor s otevřeným zdrojovým kódem, https://www.shotcut.org/.

[2] Kdenlive – bezplatný nelineární editor pro prostředí KDE4, https://kdenlive.org/.

[3] Openshot – bezplatný nelineární editor s prostředím GTK+, https://www.openshot.org/.



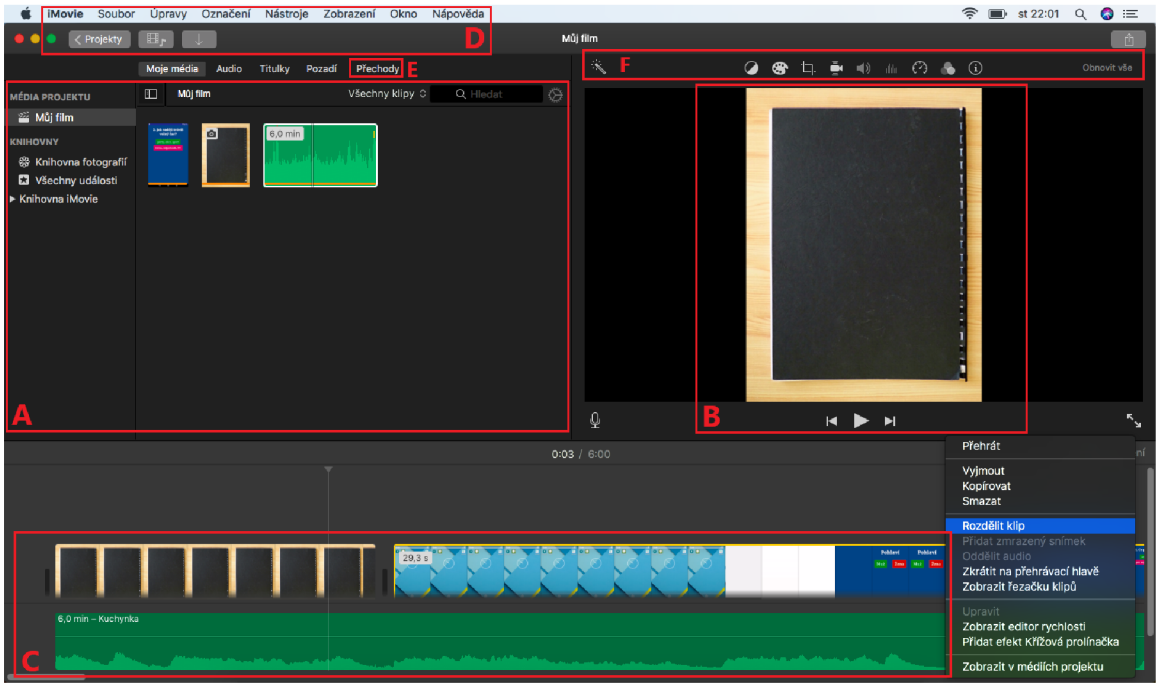
Obrázek 2.1: Uživatelské rozhraní nelineárního video editoru *Shotcut* s vyznačeným rozložením ovládacích prvků. Zdrojové soubory – A, náhled zdrojových souborů / časové osy – C, globální akce nad projektem – D, záložka filtrů a přechodů – E, F.

vpravo náhled a ve spodní části se nacházejí časové osy a nastavení vlastností filtrů/přechodů. I přes zjednodušení rozhraní zde žádné funkce nechybí.

Mezi kvalitní proprietární programy se řadí program *iMovie* od firmy Apple, *Adobe Premiere Pro* od firmy Adobe Systems a *VEGAS Movie Studio* od firmy Sony. Profesionální programy se od bezplatných liší zejména vyšší stabilitou, propracovanějšími filtry a šablonami. Licence se pohybují v řádech tisícikorun a například Adobe nabízí svůj produkt pouze ve formě předplatného. Z těchto programů jsem vyzkoušel *iMovie*[4]. Program od firmy Apple dělí pracovní prostor do stejných skupin, obrázek 2.2. Na rozdíl od předešlých programů zobrazuje v levé polovině (A) nejen soubory k aktuálnímu projektu, ale i soubory v knihovnách. V navrhovaném řešení je rovněž potřeba řešit sdílenou knihovnu s obecnými zdroji sdílenými například s uživateli fakulty. Odlišná je aplikace filtrů. Filtry se nacházejí nad náhledem videa (F). Po rozkliknutí ikony se zobrazí nabídka filtrů. Uživatel nikdy nenastavuje číselné hodnoty. Nastavení filtrů je řešeno vizuálními posuvníky, kapátky a dalšími nástroji. Zajímavě řešená je časová osa (C), která nemá pravítko s časem. Měřítko času je na ose proměnlivé. Mezi klipy je mezera, ačkoliv konec předchozího a začátek následujícího klipu je ve stejný čas. Z mého pohledu se jedná o rušivý prvek, při přehrávání ukazatel aktuálního času přeskakuje a mění svou rychlost. Na druhou stranu lze do mezer pohodlně vkládat přechody. Položky na časové ose nelze posouvat, je možné měnit pouze jejich pořadí. Pokud potřebujeme mít ve videu prázdné místo, musíme zde vložit jednolitou barvu z knihovny (záložky Pozadí). Nad časovou osou chybí lišta nástrojů. Nástroje k položkám časové osy jsou dostupné skrze kontextovou nabídku vyvolanou stiskem pravého tlačítka

[4]iMovie – proprietární nelineární videoeditor od firmy Apple, https://www.apple.com/imovie/.

myši nebo skrze klávesové zkratky. Video stopy jsou laděny do modrých barev, zvukové stopy do barev zelených. Z programu *iMovie* vychází barevné schéma tohoto projektu.



Obrázek 2.2: Uživatelské rozhraní *iMovie* s vyznačeným rozložením ovládacích prvků. Knihovna souborů – A, náhled výsledného videa – B, časová osa bez jednotného měřítka – C, globální akce – D, záložka s přechody – E, správa filtrů – F.

Zajímavým projektem mezi desktopovými aplikacemi je open source editor *fwf*[5] – lineární video editor využívající knihovnu *FFmpeg*[6]. Program je napsaný v JavaScriptovém frameworku *Vue.js*[7] a zkompilovaný pomocí nástroje *Electron*[8]. Video editor má základní funkce, není příliš spolehlivý, ale lze se u něj inspirovat s použitými technologiemi pro implementaci uživatelského rozhraní, obrázek 2.3. Program nepodporuje přechody. Na časovou osu (C) používá knihovnu *vis.js*[9]. *Timeline* je jen jedním z vizualizačních nástrojů knihovny. Nástroj slouží k zobrazení událostí od Unixové epochy (00:00:00,000 1. ledna 1970), zatímco projekt začíná časem 0. Program využívá font Source Sans Pro zveřejněnou pod svobodnou Open Font Licencí a pro ikony odnož[10] použitého fontu Material Icons od Google (licence Apache-2.0).

[5] fwf – JavaScriptový FFmpeg editor, https://github.com/daem-on/fwf.

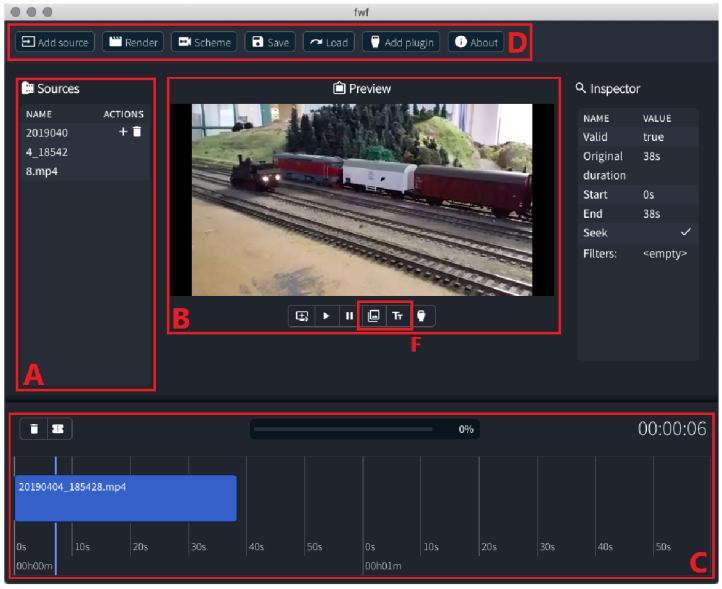
[6] FFmpeg – multiplatformní řešení pro nahrávání, konverzi a streamování audia a videa, https://ffmpeg.org/.

[7] Vue.js – open source JavaScriptový framework pro tvorbu uživatelských rozhraní, https://vuejs.org/.

[8] Electron – nástroj pro tvorbu desktopových aplikací za pomocí JavaScriptu, HTML a CSS, https://electronjs.org/.

[9] vis.js – JavaScriptová knihovna na vizualizaci dat, http://visjs.org/.

[10] Material Design Icons – odnož projektu Google, https://github.com/jossef/material-design-icons-iconfont.



Obrázek 2.3: Uživatelské rozhraní editoru *fwf*, sestavené nástrojem Electron, s vyznačeným rozložením ovládacích prvků. Zdrojové soubory – A, náhled následujících 5 sekund od aktuálního času – B, časová osa s jednou stopou – C, globální akce – D, přidání filtrů – F.

2.2 Webové editory

Na webový editor jsem nezískal žádný tip. Lidé v mém okolí raději sáhli po open source editoru či stáhli placený program obsahující crack pomocí torrentu. U webových editorů je velký rozdíl mezi placenými řešeními a neplacenými. Placené aplikace disponují mnoha funkcemi, bezplatná řešení mají zásadní omezení. Řešení s otevřeným zdrojovým kódem a svobodnou licencí jsem nenašel žádné.

Placená řešení se funkcemi i rozhraním blíží nativním aplikacím. Z placených editorů zmiňuji *WeVideo*[11], *Magisto*[12], *Loopster*[13], *Animatron*[14] a *Clipchamp*[15].

WeVideo je poskytováno v pěti cenových variantách. Nejdražší varianta neomezuje velikostí souboru ani maximální délkou. Uživatelské rozhraní umožňuje pracovat s video soubory již během nahrávání na server. Náhled je renderován v prohlížeči, výsledné video na serveru. Po dokončení renderování je uživateli zaslán email. Videa vytvořená v bezplatné variantě obsahují vodoznak.

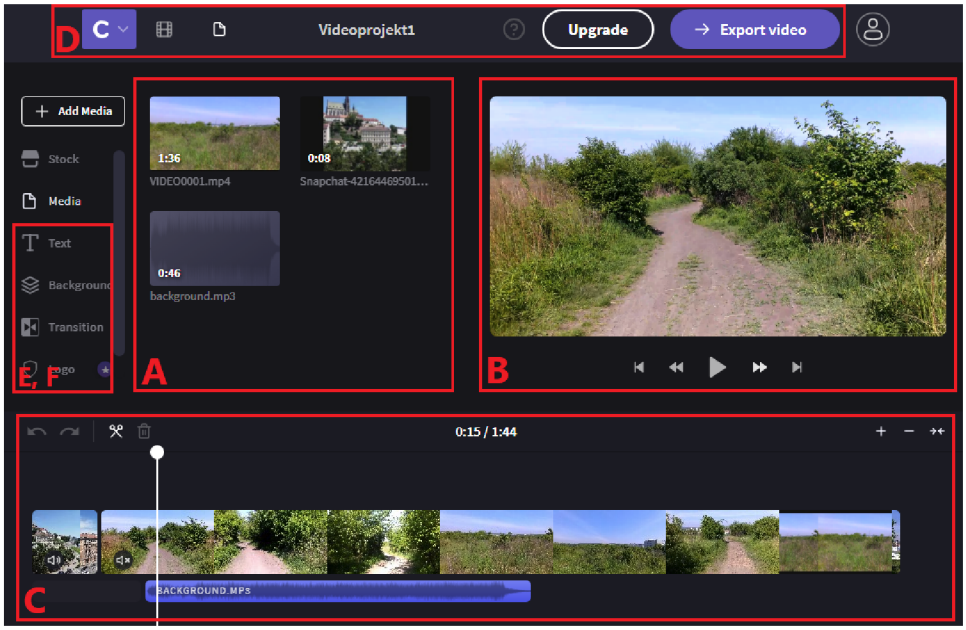
Magisto nabízí placené varianty. Nejlevnější limituje maximální délkou výsledného videa 2,5 minuty. Nejdražší limituje 10 minutami. Při vytváření projektu není k dispozici časová osa. Klipy zobrazuje jako posloupnost položek, podobně jako správci souborů. Filtry a přechody nelze nastavit pro položky zvlášť, nastavují se globálně pro celý projekt pomocí šablon. Bezplatná varianta nabízí jednoduchou 3krokovou editaci videa, ve které jsou omezené funkce.

[11] WeVideo – proprietární online editor, https://www.wevideo.com.
[12] Magisto – proprietární online editor, https://www.magisto.com/.
[13] Loopster – proprietární online editor, https://www.loopster.com.
[14] Animatron – proprietární online editor, https://www.animatron.com.
[15] Clichamp Create – proprietární online editor, který zpracovává videa v prohlížeči, https://clipchamp.com.

Loopster nabízí ve všech plánech výstupní rozlišení 1080p. Zásadním problémem pro zpracovávání přednášek je maximální délka – 30 minut u nejdražší varianty.

Animatron slouží zejména k tvorbě krátkých upoutávek a reklam. Umožňuje v nejdražší variantě vytvořit projekt o maximální délce 10 minut. Bezplatný plán neumožňuje stažení videa, pouze vystavení videa na sociálních sítích. Při práci s editorem jsem zaznamenal problém s různou orientací stejného videa napříč editorem.

Pro nenáročné uživatele by mohl být dostačující nástroj *Clipchamp Create*, který nabízí i bezplatnou variantu s omezením rozlišení videí na 480p. Velikost projektu a maximální délka omezena není, video není opatřeno reklamním vodoznakem. V nejdražší variantě umožňuje vytvářet videa o rozlišení až 1080p. Rozhraní je shodné s desktopovými programy, obrázek 2.4. Editor běží pouze v prohlížeči Google Chrome a zpracování videa probíhá na straně webového prohlížeče. Uživatel nesmí videoeditor během zpracovávání opustit. Doba zpracování videa o délce do 2 minut trvá i při rozlišení 480p cca 20 minut. Kromě nahraných souborů (A) poskytuje knihovnu s audiem a videem.



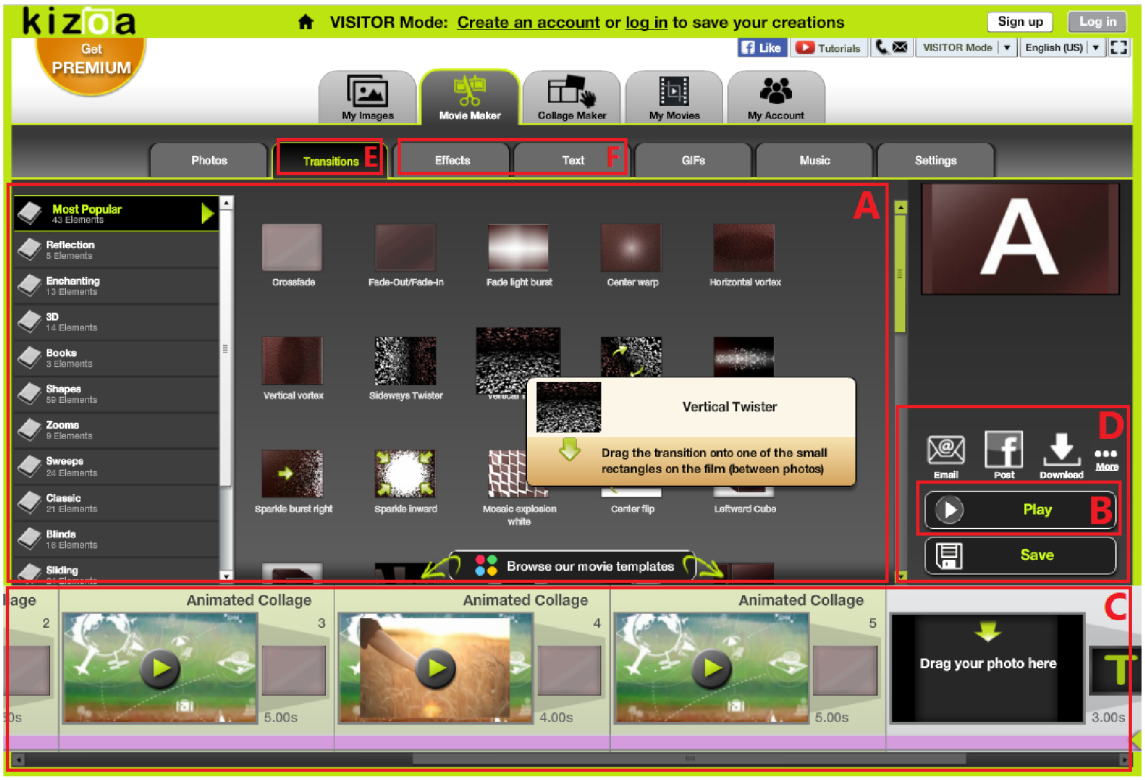
Obrázek 2.4: Uživatelské rozhraní nelineárního online video editoru *Clipchamp Create* s vyznačeným rozložením ovládacích prvků. Knihovna nahraných a poskytovaných souborů – A, náhled videa – B, časová osa s libovolným množstvím stop – C, export projektu – D, knihovna přechodů a filtrů – E, F.

Další dva editory uvádím pro úplnost, ani jeden není vyhovující – *Kizoa*[16] a *Movie Maker Online*[17].

Editor *Kizoa* je vytvořen v programu *Adobe Flash*, obsahuje dostatečné množství efektů (F), přechodů (E) a nastavení, obrázek 2.5. Kromě nahraných souborů poskytuje vlastní knihovnu s obrázky, videi a audiem. Vespod zobrazuje jednu časovou osu (C), jedná se o lineární editor. Náhled výsledného videa (B) zobrazuje v překryvném okně, při přehrávání náhledu dochází k problému se stabilitou prohlížeče. Hlavní nevýhodou je vázanost na *Adobe Flash Player*, který je dnes na ústupu a jeho podpora skončí v roce 2020 [6]. Bezplatná

[16] Kioza – bezplatný lineární online video editor, https://www.kizoa.com.
[17] Movie Maker Online – bezplatný online editor, https://moviemakeronline.com/.

varianta nabídne 2 minuty videa s rozlišením 720p, nejdražší varianta 4K video bez omezení délky.



Obrázek 2.5: Uživatelské rozhraní online lineárního video editoru *Kizoa* s vyznačeným rozložením ovládacích prvků. Knihovna nahraných a poskytovaných souborů – A, otevření okna s náhledem – B, časová osa s jednou video a jednou audio stopou – C, globální akce – D, knihovna přechodů – E, filtry a efekty – F.

Movie Maker Online představuje video editor s netradičním uživatelským rozhraním, obrázek 2.6. Nevyužívá zásuvných modulů a osu s videem zobrazuje vertikálně (C). Jedná se o nelineární editor, který má 4 stopy – zvukovou, stopu s pozadím, hlavní stopu a stopu s překryvným textem. Přidat nebo odebrat stopu není možné. Editor je řešen jako jedna stránka, na které se pod sebou nacházejí jednotlivé kroky. Nahrávání souborů probíhá v prvním kroku (A), poté se musí uživatel přesunout ke kroku 2 a provést editace. Obsahuje omezené množství filtrů (F) a z přechodů lze realizovat prolnutí pomocí dvou stop s postupným náběhem/stmíváním (E). Pro uživatele desktopových aplikací se jedná o zcela jiné rozhraní, ve kterém se budou těžko orientovat. Na rozdíl od předchozího řešení neumožňuje *Movie Maker Online* projekt uložit nebo exportovat a později opětovně použít. Program lze používat bez registrace, výsledné video o rozlišení 720p je opatřeno vodoznakem a nesmí přesáhnout 10 minut. Program by měl nabízet předplatné, aktuálně ale nelze získat.

2.3 Shrnutí

U desktopových aplikací máme široký výběr. Všechny testované programy mají podobný způsob ovládání i podobné funkce. Nemusíme se bát sáhnout po bezplatném open source řešení, i tyto programy obsahují dostatek funkcí pro kvalitní tvorbu. Neobsahují omezení

Obrázek 2.6: Uživatelské rozhraní online video editoru *Movie Maker Online* s vyznačeným rozložením ovládacích prvků. První krok s nahráváním souborů – nad oblastí A, vertikální časová osa – C, globální akce (D) se nacházejí vespod stránky, postupný náběh/stmívání – E, nastavení filtrů — F.

na maximální rozlišení nebo délku výsledného videa. Dražší programy nabídnou propracovanější filtry a vyšší stabilitu. Při návrhu online videoeditoru je dobré zachovat rozhraní, které uživatelé znají právě z těchto aplikací. Z pohledu technologií je zajímavý jednoduchý open source lineární editor *fwf*, který je napsán v JavaScriptovém framoworku *Vue.js*.

U webových řešení je také na výběr. Mezi jednotlivými aplikacemi je velký rozdíl ve funkcích a zejména v omezeních výsledného videa. Z testovaných řešení nabízí neomezenou délku videa pouze program *WeVideo* a *Clipchamp Create*, který ale zpracovává videa v prohlížeči a *Kizoa*, kterého jsem vyřadil ze srovnání kvůli vázanosti na *Adobe Flash*. V rozhraních jsou větší rozdíly než u desktopových aplikací. Uživatel desktopových editorů videa nebude mít problém s orientací v jednom z online editorů, vyjma *Movie Maker Online* se zcela odlišným rozhraním. Funkce a zásadní omezení online videoeditorů srovnávám v tabulce 2.1.

V případě, že požadujeme online editor videí, ale zároveň potřebujeme mít svá data pod kontrolou na svém serveru, připadá v úvahu pouze program *Clipchamp Create*. Program provádí veškeré operace přímo v prohlížeči. Renderování delších videí s rozlišením větší než 480p vyžaduje výkonný hardware. V tomto případě je lepší sáhnout po desktopové aplikace. Pokud chceme zpracovávat videa online, ale na našem serveru, máme smůlu.

Shotcut	platforma	desktop, Windows/macOS/Linux
	licence	GPL-3.0, open source
	délka	neomezená
	rozlišení	4K
	šablony	✓
	možnost ukládání	×
	knihovna médií	×
	video stopy	✓
	audio stopy	✓
	filtry	✓(77)
	přechody	✓(24)
Kdenlive	platforma	desktop, Linux / Windows (beta)
	licence	GPL-2.0, open source
	délka	neomezená
	rozlišení	neomezené
	šablony	×
	možnost ukládání	✓
	knihovna médií	×
	video stopy	✓
	audio stopy	✓
	filtry	✓
	přechody	✓
Openshot	platforma	desktop, Windows/macOS/Linux
	licence	GPL-3.0, open source
	délka	neomezená
	rozlišení	neomezená
	šablony	×
	možnost ukládání	✓
	knihovna médií	×
	video stopy	✓
	audio stopy	✓
	filtry	✓
	přechody	✓
iMovie	platforma	desktop, macOS
	licence	proprietární software
	délka	neomezená
	rozlišení	4K
	šablony	×
	možnost ukládání	✓
	knihovna médií	✓(audio, pozadí, titulky)
	video stopy	✓
	audio stopy	✓

	filtry	✓(~68)
	přechody	✓(24)
fwf	platforma	desktop, Windows/macOS
	licence	MIT, open source
	délka	neomezená
	rozlišení	neomezená
	šablony	×
	možnost ukládání	✓
	knihovna médií	×
	video stopy	1
	audio stopy	×
	filtry	✓(14)
	přechody	×
WeVideo	platforma	webová aplikace, moderní prohlížeče
	licence	proprietární software
	délka	neomezená
	rozlišení	4K
	šablony	✓(32)
	možnost ukládání	✓
	knihovna médií	✓(video, audio)
	video stopy	✓
	audio stopy	✓
	filtry	✓(26)
	přechody	✓(42)
Magisto	platforma	webová aplikace, moderní prohlížeče
	licence	proprietární software
	délka	10 minut
	rozlišení	1080p
	šablony	✓
	možnost ukládání	✓
	knihovna médií	✓(video, audio)
	video stopy	×
	audio stopy	×
	filtry	✓(4)
	přechody	×
Loopster	platforma	webová aplikace, moderní prohlížeče
	licence	proprietární software
	délka	30 minut
	rozlišení	1080p
	šablony	×
	možnost ukládání	✓
	knihovna médií	×
	video stopy	1
	audio stopy	1
	filtry	hlasitost audia
	přechody	✓(6)
	platforma	webová aplikace, moderní prohlížeče
	licence	proprietární software

Animatron	délka	10 minut
	rozlišení	1080p
	šablony	✕
	možnost ukládání	✓
	knihovna médií	✓(video, obrázky, audio)
	video stopy	1
	audio stopy	1
	filtry	✓(6)
	přechody	✓(8)
Clipchamp	platforma	webová aplikace, prohlížeč Google Chrome
	licence	proprietární software
	délka	neomezená
	rozlišení	1080p
	šablony	✓
	možnost ukládání	✓
	knihovna médií	✓(video, audio)
	video stopy	✓
	audio stopy	✓
	filtry	✓(13)
	přechody	✕
Kioza	platforma	Adobe Flash applet, prohlížeč s pluginem
	licence	proprietární software
	délka	neomezená
	rozlišení	4K
	šablony	✓
	možnost ukládání	✓
	knihovna médií	✓(video, audio, obrázky)
	video stopy	1
	audio stopy	1
	filtry	✓(52)
	přechody	✓(205)
Movie Maker Online	platforma	webová aplikace, moderní prohlížeče
	licence	proprietární software
	délka	10 minut
	rozlišení	720p (s vodoznakem)
	šablony	✕
	možnost ukládání	✕
	knihovna médií	✕
	video stopy	2
	audio stopy	1
	filtry	✓(36)
	přechody	✕
Navrhované řešení	platforma	webová aplikace, moderní prohlížeče
	licence	Apache-2.0, open source
	délka	alespoň 1 hodina
	rozlišení	alespoň 1080p
	šablony	✕
	možnost ukládání	✓

	knihovna médií	✓(FIT intro)
	video stopy	✓
	audio stopy	✓
	filtry	✓
	přechody	prolnutí

Tabulka 2.1: Přehled vlastností a funkcí editorů popisovaných v kapitole 2. Nejprve jsou uvedeny desktopové aplikace, poté webové a poslední srovnávanou položkou jsou očekávané vlastnosti editoru, který vzniká v rámci této práce. Knihovnou médií se rozumí soubory dostupné všem uživatelům pro použiti v projektu. Pokud jsou video/audio stopy zatržené, lze hovořit o nelineárním editoru.

Kapitola 3

Návrh řešení

Hlavním cílem práce je vytvořit videoeditor, který bude fungovat ve všech moderních prohlížečích a výsledné zpracování videa nebude probíhat na straně klienta. K dosažení těchto cílů bude nutné vytvořit webovou aplikaci. Stejně jako stávající řešení by i tento editor měl provádět veškeré úkony pomocí JavaScriptu a asynchronních požadavků, bez nutnosti znovu načítat celou stránku. Vzhledem ke komplexnosti uživatelského rozhraní není vhodné zvolit čistý JavaScript. Vyplatí se sáhnout po JavaScriptovém frameworku, usnadňují časté operace, udržují dobré návyky a pomáhají vytvořit čistý kód.

Nejpoužívanějšími frameworky jsou: React, Backbone.js, RequireJS, AngularJS, nebo třeba Vue.js (používané programem *fwf*) [4]. Z těchto frameworků jsem vybral React, který je zároveň i šablonovacím systémem a má kvalitní dokumentaci. Šablonovací systém používám zejména k rozdělení uživatelského rozhraní na komponenty.

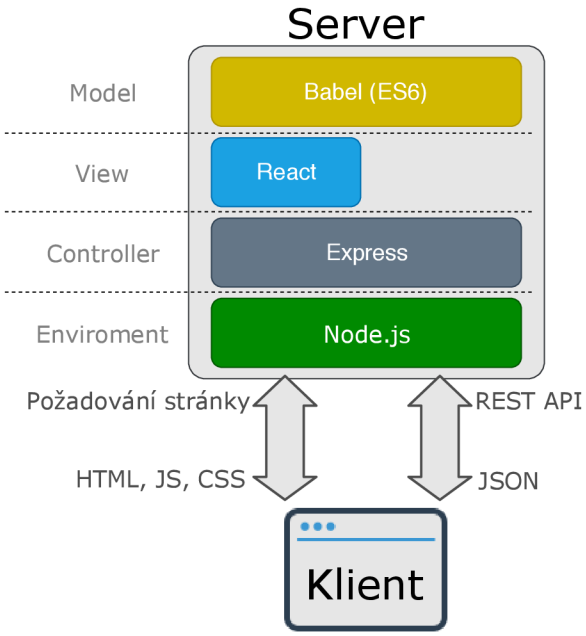
Webová aplikace bude potřebovat i serverovou část, na kterou klientská část nahraje soubory, popíše serveru operace nad soubory a dá pokyn pro vyhotovení výsledného videa. Tuto architekturu označujeme z pohledu sítě jako klient-server. Z pohledu návrhu se bude jednat o MVC (model-view-controller) architekturu. Mezi klientem a serverem je nutné komunikovat. Pro výměnu dat po síti musejí obě části používat dohodnutý serializační formát. JavaScript nejčastěji používá *XML* nebo *JSON*. *JSON* (JavaScriptový objektový zápis) je novější a zápis ve formátu *JSON* je validním kódem JavaScriptu. *XML* (rozšiřitelný značkovací jazyk) na druhou stranu umožňuje přenos binárních dat. Se serverem bude komunikovat kód v JavaScriptu, volba formátu *JSON* je tedy výhodná.

Sever bude mít za úkol 2 činnosti. Nejprve bude muset uživateli poskytnout soubory s HTML, CSS a JavaScriptem a poté reagovat na JavaScriptové asynchronní požadavky. Při návrhu jsem se rozhodoval mezi implementací serveru pomocí jazyka PHP a pomocí Node.js. Jazyk PHP vznikal v době statických stránek a osobních webů, Node.js je obvykle spojován se single page aplikacemi. Node.js jsem si vybral právě protože plánuji vytvořit single page aplikaci, kvůli možnosti používat stejný jazyk pro klientskou i serverovou část a také kvůli asynchronnímu přístupu k blokujícím operacím. Po PHP bych naopak sáhl při tvorbě e-shopu, blogu či informačního systému.

Server bude pro klienta dostupný skrze rozhraní (API). Rozhraní bude sloužit k vyvolání akcí na serveru i k získávání dat. Existuje více koncepcí pro komunikaci a tvorbu API, například *XML-RPC*, *SOAP* nebo *REST*. *XML-RPC* a *SOAP* používají jako serializační formát pro výměnu dat *XML*, kvůli JavaScriptu na straně klienta i serveru jsem ovšem zvolil *JSON*. *REST* je způsob komunikace na základě HTTP požadavků. *REST* není na HTTP vázaný, ale není praktický důvod používat odlišný protokol. Výchozím serializačním formátem pro výměnu dat je JSON, ovšem lze používat například i *XML*.

Dále bude nutné zpracovávat multimediální soubory. *Clipchamp Create* provádí zpracování na straně klienta ve webovém prohlížeči. Webový prohlížeč ani webové technologie nejsou navrženy pro renderování videí, renderování videí v prohlížeči není efektivní. Editor byl navržen, aby zpracovával videa na serveru. K tomu může sloužit program *FFmpeg* nebo *MLT*. Program *FFmpeg* používá například editor *fwf* nebo Android aplikace *Video Editor using FFmpeg*[1]. *FFmpeg* se používá skrze příkazový řádek. Každá operace představuje jeden příkaz (zkrátit video, přidat efekt, spojit videa). Na pořadí operací záleží. Projekt by byl popsán souborem s posloupností příkazů pro FFmpeg, případně jedním dlouhým příkazem. Multimediální framework *MLT* rovněž nabízí své nástroje skrze příkazový řádek. Projekt je možné popisovat posloupností příkazů a také pomocí serializačního formátu *XML*. Celý projekt se popíše entitami *XML* a programu *MLT* se poté pouze předá *XML* soubor. *XML* soubor bude uložen po celou dobu na serveru a server jej bude editovat na základě požadavků klienta. *REST API* je bezstavové, požadavek musí být úplný a stav projektu uložen v *XML* souboru projektu. Projekt nebude využívat databázový systém.

Seznam použitých technologií, jejich umístění a komunikaci zobrazuje obrázek 3.1.



Obrázek 3.1: Použité technologie z hlediska architektury klient-server i z hlediska návrhového vzoru model-view-controller.

3.1 Zvolené technologie

Následující podkapitola shrnuje použité technologie, zobrazené ve schematickém obrázku 3.1. Zmiňuje podobné technologie, vysvětluje odlišnosti, shrnuje přínos pro projekt a nastiňuje použití.

[1]Video Editor using FFmpeg – open source editor videa pro android, https://github.com/bhuvnesh123/FFmpeg-Video-Editor-Android.

3.1.1 Node.js

Node.js vytváří běhové prostředí JavaScriptu postavené na JavaScriptovém enginu V8 od firmy Google. Umožňuje používat JavaScript jako systémový jazyk, který má přístup k paměti, bufferům, procesům, souborům a soketům. Cílem Node.js je poskytovat snadnou cestu k vytvoření škálovatelných síťových aplikací. Node.js nepoužívá vlákna, je založen na událostech a asynchronním vykonávání blokujících operací, umožňuje snadnou modularitu. Právě eliminací blokujících operací pomocí asynchronní obsluhy řízenou událostmi lze dosáhnout vysokého výkonu. PHP vytváří pro každý požadavek samostatné vlákno a v případě blokující operace čeká na vyřízení, Node.js používá pro všechny požadavky jedno vlákno a namísto paralelního zpracování používá Node.js pseudoparalelní zpracování.

Pro porovnání rychlosti vykonání programu, uvedu příklad jednoduchého programu, ve kterém se v prázdném cyklu inkrementuje miliardkrát čítač. Doba běhu programu je pro jednotlivé jazyky následující – JavaScript (Node.js verze 9.3.0) 0,635 sekund, jazyk C (Apple LLVM 8.1.0) 2,398 sekund a Python (3.6.1) 31,096 sekund. Všechny testy probíhaly na zařízení MacBook Pro s procesorem Intel Core i7 2,8 GHz. Překladač V8 JavaScript kompiluje namísto interpretování, díky tomu lze využívat vysokoúrovňový jazyk bez nutnosti vzdát se rychlosti kompilovaných jazyků [10].

JavaScript na klientské straně je pro interaktivní webovou aplikaci nutnost. Díky běhu JavaScriptu na klientské i serverové části mohou programátoři používat jeden jazyk v rámci projektu. Dále se otevírá možnost používat stejný kód na serveru i na straně klienta. Stačí vhodně členit funkce do modulů a tyto moduly poté importovat jak na straně serveru, tak na straně klienta. V mé práci takto sdílím kód pro práci s časovými značkami.

3.1.2 Express framework

Pro usnadnění tvorby webového serveru jsem použil Express. Express je webový framework pro Node.js. Poskytuje knihovny užitečné pro zajištění základních funkcí webového serveru (routování, podpora šablonovacího systému pro generování stránek, zpracování dotazů, parametrů, sezení a cookies, autentifikaci, vytváření odpovědí, zpracování chyb, aj.).

3.1.3 ECMAScript, TypeScript

Jedná se o programovací jazyk se syntaxí inspirovanou jazykem C, který se překládá do JavaScriptu. TypeScript je nadstavba nad JavaScriptem, program v JavaScriptu je validním programem v TypeScriptu. Přináší silnou typovou kontrolu, třídy (dnes i v ES6), rozhraní, dědičnost. Offline překlad do JavaScriptu odhalí syntaktické chyby, a volitelně zajistí kompatibilitu se standardy ES3 a ES5 (alternativou je *Babel*, na kontrolu kódu na syntaktické chyby pak *ESLint*[2]). Dále přidává spoustu „syntaktického cukru“ – gettery/settery, výčtové typy, promises. Podporuje integraci s Angular 2 i React [12].

Na druhou stranu je typová kontrola plnohodnotná pouze pokud všechny balíčky obsahují hlavičkové soubory TypeScript. TypeScript je nutné začlenit do vývojového procesu s Node.js a React. Během vývoje se při každé změně zdrojového kódu musí provést kompilace do JavaScriptu. Pro začínajících programátory a pro malé projekty zavádí složitost, která nemusí převážit výhody zavedení.

S příchodem ES6 přejal standard z TypeScriptu deklaraci proměnných s platností v daném rámci bez vedlejších efektů (`let`), možnost vytvářet třídy, dědičnost, rest parametry, promises a další. Hlavní výhodou zůstává silné typování a kontrola syntaxe při kompilaci.

[2] ESLint – nástroj pro analýzu JavaScriptového kódu, https://eslint.org/.

Vzhledem k mým předchozím zkušenostem s jazykem PHP jsem tyto vlastnosti nepožadoval, proto jsem se rozhodl použít standard ES6 a novější revize. Zpětnou kompatibilitu klientského JavaScriptu jsem zajistil nástrojem *Babel*. Syntaktickou kontrolu jsem prováděl nástrojem *ESLint*.

3.1.4 React

React je JavaScriptová knihovna pro tvorbu uživatelských rozhraní. Je založen na komponentách, stavech komponent a na signálech mezi komponentami. Knihovna dovolí rozdělit uživatelské rozhraní do od sebe izolovaných částí kódu – komponent. Každá komponenta má svůj vlastní stav, své funkce a má plně pod kontrolou své vykreslení. Vztah komponent bývá hierarchický, skládáním komponent můžeme vytvořit komplexní rozhraní při zachování čistého kódu. Stav aplikace uchovávají jednotlivé komponenty, stav nemá jedno centrální úložiště. Někdy nám tato vlastnost nemusí vyhovovat. V takovém případě stačí vytvořit kořenovou komponentu, a původní komponenty vložit do kořenové komponenty. Úložištěm stavu poté může být kořenová komponenta. Jak může vypadat komponenta React ukazuje následující ukázka.

```
1  import React, { Component } from 'react';
2  import SourcesTableRow from './SourcesTableRow';
3  import Uploader from './Uploader';
4
5  export default class Sources extends Component {
6      render() {
7          return (
8              <div id={'sources'}>
9                  <i className="material-icons" aria-hidden="true">video_library</i>
10                 <table>
11                 <tbody>
12                     {Object.keys(this.props.items).map(key =>
13                         <SourcesTableRow
14                             key={this.props.items[key].id}
15                             item={this.props.items[key]}
16                             onRemove={this.delResource}
17                             onInsert={this.putResource}
18                         />)
19                     }
20                     <tr>
21                         <td colSpan="3">
22                         <Uploader value={{
23                             onAdd: resource => this.props.onAddResource(resource),
24                             project: this.props.project,
25                         }}
26                         />
27                         </td>
28                     </tr>
29                 </tbody>
30                 </table>
31             </div>
32         );
33     }
34     ...
```

Na ukázce je vidět hierarchie komponent. Aktuální komponentou je `Sources`, komponenta se v těle funkce pro vykreslení odkazuje na `SourcesTableRow` a `Uploader`. Komponenta `Sources` bude těmto dvěma komponentám nadřazená. Pokud by byla komponenta

`Sources` kořenová, nemá na ni kdo odkazovat. Je potřeba určit v jakém místě HTML stránky se má vykreslit. Toho docílíme v JavaScriptovém souboru, který Webpack zkompiluje do klientského JavaScriptového souboru:

```
ReactDOM.render(<Sources items={{}} onAddResource={() => {}} />, document.getElementById('app'));
```

Pokud uživatel navštíví stránku s React komponentami, získá nejprve soubor HTML s kostrou stránky. Vespod HTML bude odkaz na JavaScriptový soubor, který se stáhne, vykoná a nahradí elementy kostry vykreslenými React komponentami. V tomto případě se obsah elementu `<div id="app"></div>` nahradí vykreslenou komponentou `Sources`.

```
<!DOCTYPE html>
<html lang="cs" dir="ltr">
    <head>
        <title>React page</title>
    </head>
    <body>
        <div id="app">
            <h2>Nacitani<h2>
            Pokud se aplikace nenacte, zkontrolujte zapnuty JavaScript
        </div>
        <script src="/main.js" async></script>
    </body>
</html>
```

Pokud má být výsledný kód přehledný, vyplatí se vytvářet komponenty jako třídu, která rozšiřuje třídu `Component`. Je vhodné pro každou třídu použít samostatný soubor a třídu s komponentou určit jako výchozí export. Díky tomu můžeme ostatní komponenty použít importem souboru.

Třída komponenty může obsahovat vlastní funkce a také funkce, které mají zvláštní význam. Funkce konstruktoru třídy (`constructor`) je obvyklá. Na začátku konstruktoru musíme volat konstruktor předka. V konstruktoru se obvykle nastavuje stav komponenty, atributy třídy a také se zde fixuje kontext pro proměnnou `this` uvnitř funkcí.

Dalšími důležitými metodami zvláštního významu jsou například: `componentDidMount`, `componentDidUpdate`, `componentWillUnmount` a `render`.

Metoda `render` zajišťuje vykreslení komponenty. Funkce má za úkol vrátit JSX kód komponenty. JSX je rozšíření syntaxe JavaScriptu. JSX svou syntaxí připomíná spíše XML, než HTML. Všechny tagy jsou párové (musejí být uzavřeny). Kód musí být uzavřen do kořenového elementu. Pokud nechceme vykreslovat kořenový element, můžeme použít prázdné tagy `<></>`. JSX kombinuje HTLM a JavaScript. Příkazy a proměnné JavaScriptu jsou dostupné ve složených závorkách – `{this.props.items}`. Atributy HTML prvků se zapisují v CamelCase (notaci velkých písmen). Atributy nabývají stejných hodnot, jako JavaScriptové proměnné. Zvláštností je atribut `style`, který přijímá objekt s CSS vlastnostmi [3].

Zbývající metody se volají při vložení komponenty do virtuálního DOM stromu, který si React vytváří, při aktualizaci a při odstraňování z DOM.

Aby uživatelské rozhraní fungovalo, musejí mezi sebou komponenty komunikovat. Komponenty mají stromovou strukturu, komunikace probíhá dvěma směry – od předka k potomkovi a naopak. Komponenty mohou podřazeným komponentám při jejich vytváření předat parametry. Můžeme předávat callbacky nebo stav komponenty. V podřazené komponentě jsou tyto parametry dostupné skrze objekt `this.props`. Pokud podřazené komponentě předáme stav a podřazená komponenta se stavem přímo pracuje ve funkci `render`, dojde se změnou stavu předka i k aktualizaci a překreslení podřazené komponenty. Pokud například změníme stav kořenové komponenty, může tato změna „probublat“ ke komponentám

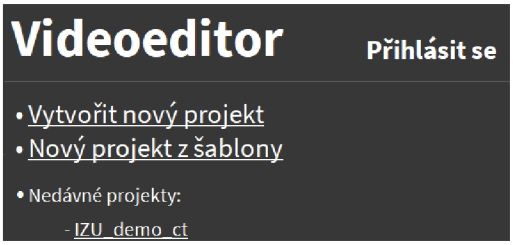
o několik úrovní níže, vše se děje automaticky. Komunikace směrem ke kořenu probíhá pomocí callbacků. Podřazená komponenta může změnit stav nadřazené pouze nepřímo. Zavolá funkci, kterou mu nadřazená komponenta zpřístupnila v `this.props`, a na základě volání nadřazená komponenta změní svůj stav a obě komponenty se překreslí.

Stav komponenty se nastavuje metodou `this.setState`. Jako parametr se předá objekt s položkami, které se mají přepsat. Předchozí a následující stav se nesmí ovlivňovat. Pokud chceme upravit jen jednu položku objektu, musíme vytvořit kopii objektu a tuto kopii nastavit jako nový stav [5].

3.2 Návrh uživatelského rozhraní

Cílovou skupinou jsou počítačově gramotní uživatelé, kteří někdy používali program na editování videa. Rozhraní je voleno tak, aby připomínalo klasické desktopové aplikace a pomohlo uživatelům se zorientovat. Typickou úlohou pro videoeditor bude sestříhat záznam přednášky do výukového videa, nebo vytvořit tutoriál ze záznamu z kamery nebo obrazovky. Aplikace bude využívána na počítačích a noteboocích, není nutné přizpůsobovat rozhraní obrazovkám mobilních zařízení. Ovládání bude pomocí klávesnice a myši, dotekové obrazovky se sice nepředpokládají, ale ani nevylučují.

Po načtení úvodní stránky se zobrazí možnost vytvořit nový projekt. Stránka bude do budoucna připravena na rozšíření o další možnosti, jako jsou šablony, autentizace uživatelů, či seznam vlastních projektů. Pro zahájení editace uživatel zvolí *Vytvořit nový projekt*, obrázek 3.2.



Obrázek 3.2: Úvodní obrazovka pro vytváření nových projektů.

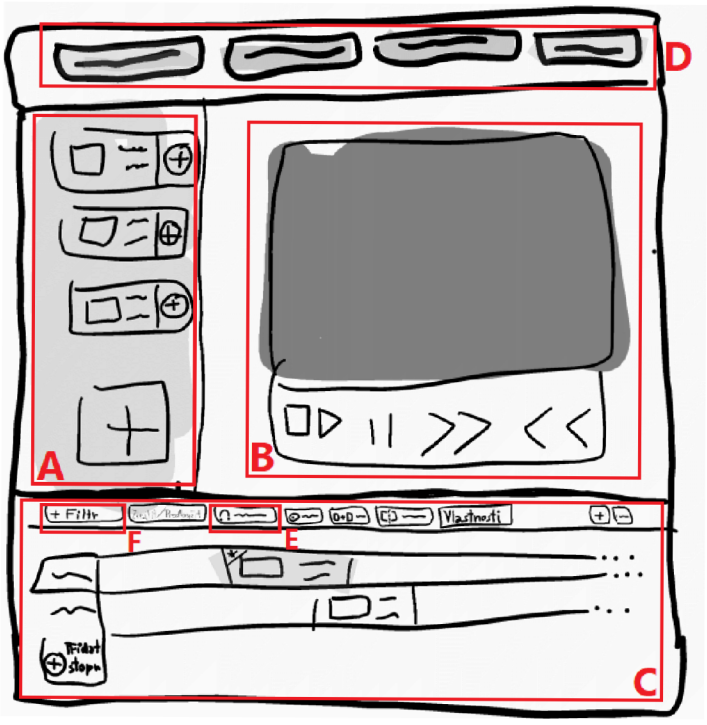
Po vytvoření nového projektu se uživatel ocitne v rozhraní pro tvorbu videí. Aplikace je řešena jako single page aplikace. Uživatel po celou dobu pracuje na jedné obrazovce a i při dialozích je editor překryt pouze částečně, tak, aby měl uživatel pocit, že má svůj projekt po celou dobu pod kontrolou. Při návrhu uživatelského rozhraní editoru jsem se nejvíce inspiroval desktopovou aplikací *iMovie*, *Shotcut* a online nástrojem pro úpravu fotografií – *Photopea*[3].

Uživatelské rozhraní editoru je rozděleno do 4 částí. Rozložení bylo navrženo a odzkoušeno na drátěném modelu, obrázek 3.3. Na základě drátěného modelu byl zhotoven prototyp hlavního uživatelského rozhraní, obrázek 3.4.

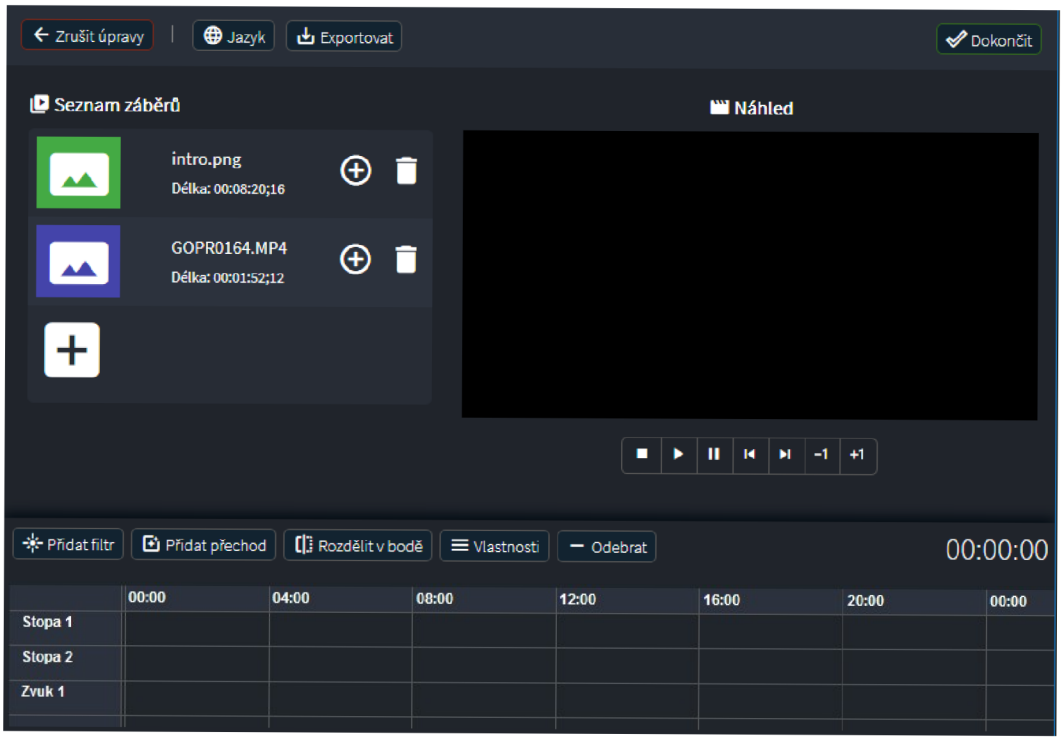
U horního okraje obrazovky se nachází panel nástrojů pro globální akce nad projektem (D) – zrušit úpravy nebo renderovat projekt. Do budoucna zde bude možnost přepínat jazykovou mutaci a možnost exportu nebo výmazu dat.

Operace nad videi jsou členěny do 3 částí, tomu odpovídá a členění uživatelského rozhraní. V levé části (A) je seznam zdrojového materiálu (videí, obrázků, aj.), který je možné

[3]Photopea.com – bezplatný editor fotografií, https://www.photopea.com/.

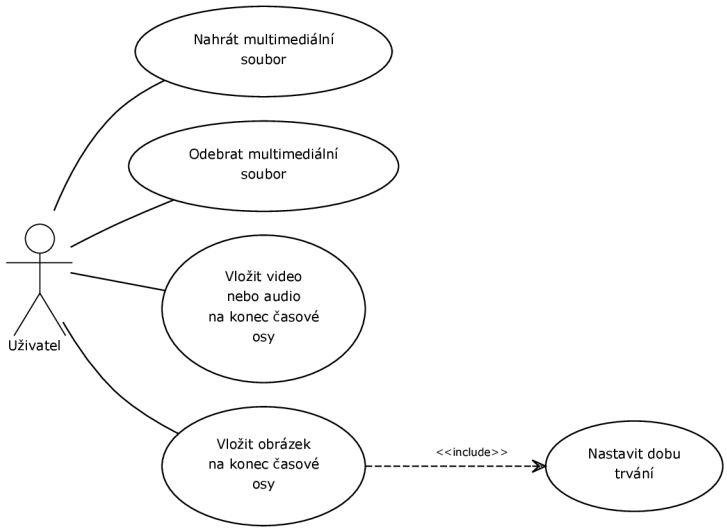


Obrázek 3.3: Drátěný model uživatelského rozhraní s vyznačeným rozložením ovládacích prvků. Seznam zdrojových záběrů – A, náhled výsledného videa – B, časové osy s ovládacími prvky – C, hlavní nabídka s globálními akcemi – D, správa přechodů a filtrů – E, F.



Obrázek 3.4: Prototyp uživatelského rozhraní. Vychází z drátěného modelu.

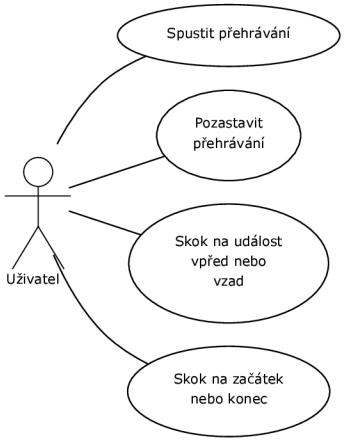
v projektu použít. S položkami seznamu se zdrojovým materiálem se pracuje přímo. Dostupné operace jsou zobrazeny po celou dobu. Pod poslední položkou se nachází tlačítko na přidání materiálu, případně je možné použít přetažení souboru do této oblasti. V seznamu zdrojů je možné nahrávat obrázkové, video a audio soubory. Podporované formáty video/audio souborů jsou shodné s formáty, které podporuje program FFmpeg. Z obrázkových souborů je podporován formát PNG a JPEG obrázek. Obrázek 3.5 zobrazuje diagram případů užití pro práci se zdrojovým materiálem.



Obrázek 3.5: Diagram případů užití pro práci se zdroji.

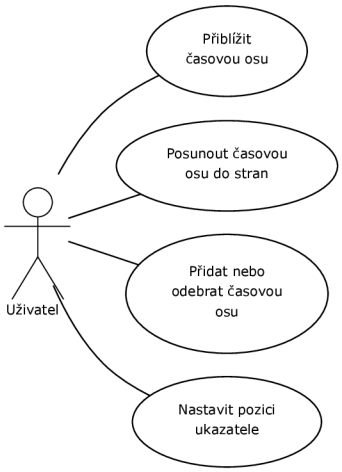
V pravé polovině (B) se nachází přehrávač předběžného náhledu výsledného videa. Přehrávač náhledu se ovládá tlačítky umístěnými přímo pod přehrávačem (přehrát, zastavit, skok na začátek nebo konec, skok na začátek nebo konec aktuální položky). Přehrávač přehrává pouze soubory umístěné na časové ose, nezobrazuje náhledy zdrojového materiálu. Aktuální pozici přehrávače udává ukazatel na časové ose. Přehrávání lze spustit původní rychlostí a pozastavit. K manipulaci s pozicí ve videu slouží skok na událost vpřed/vzad, který přesune ukazatel na nejbližší začátek nebo konec položky libovolné stopy, diagram případů užití zobrazuje obrázek 3.6.

Ve spodní části (C) je časová osa se zvukovými a obrazovými stopami. Časová osa zobrazuje pod sebou stopy v pořadí, v jakém byly vytvořeny. Stopy se přidávají a odebírají podle potřeby. Pokud uživatel uchopí položku a přetáhne ji pod poslední video/audio stopu, vytvoří se nová. Původní návrh počítal se zobrazováním názvů stop a manuálním přidávání stop. Po testování jsem se rozhodl odstínit uživatele od stop a nabídnout mu pracovní plochu, kde může přesouvat položky i vedle sebe, bez nutnosti řešit stopy. Položky se na časovou osu přidávají ze seznamu materiálu tlačítkem plus u materiálu. Posun položek na časové ose se provede uchopením prvku a přetažením na nové místo, přesun na jinou časovou se provádí přetažením na jinou osu stejného typu. Uchopením položky za okraj lze měnit začátek nebo konec. U každé položky je náhledový obrázek a textové informace o položce. S vybranou položkou lze pracovat tlačítky na liště nástrojů u horního okraje časové osy. V aplikaci jsou dvě lišty nástrojů, globální akce (D) a akce s časovou osou (E, F). Pokud je na položku aplikován alespoň jeden filtr, zobrazí se v jejím levém horním rohu „nálepka“ s ikonou filtru. Časovou osu lze přibližovat a oddalovat tlačítky v liště nástrojů časové osy.



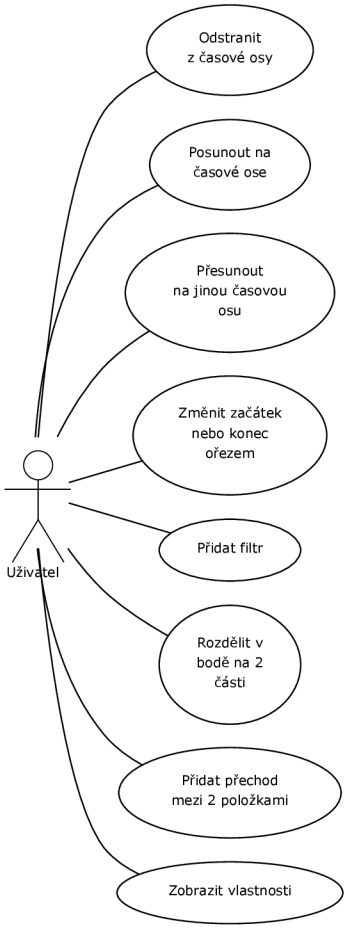
Obrázek 3.6: Diagram případů užití pro práci s přehrávačem náhledu.

Pokud je časová osa přiblížena natolik, že není možné zobrazit všechny prvky časové osy, lze osu posouvat do stran uchopením prázdného místa na ose a přetažením osy do stran. V novém projektu je k dispozici prázdná zvuková a video stopa. Ukazatel aktuálního času lze uchopit a přesunout. V čase ukazatele se bude přehrávat náhled videa a v tomto čase lze provést střih videa, diagram případů užití – obrázek 3.7.



Obrázek 3.7: Diagram případů užití pro práci s časovou osou.

Po vybrání položky na časové ose jsou k dispozici v panelu nástrojů časové osy následující akce. Položku lze smazat, okolní položky zůstávají, přechody se zruší. Lze přidat filtr obrazu (F) (text, HTML překrytí, jas, kontrast, odstín, světlost, sytost, vyvážení bílé, otočit, velikost, poloha, ořezání, průhlednost, ostrost) nebo filtr zvuku (ztišit, zesílit/zeslabit). Přidat přechod prolnutí mezi 2 položkami (E), položky s přechody se při přesouvání chovají jako celek. Dále lze zobrazit souhrnné vlastnosti. Začátek nebo konec video/audio souboru či délku obrázku lze změnit uchopením okraje položky a táhnutím na požadovanou pozici. Akce s vybranou položkou zobrazuje diagram případů užití na obrázku 3.8.



Obrázek 3.8: Diagram případů užití pro práci s položkami časové osy.

Pokud je potřeba upřesnit, jakou akci chce uživatel provést (výběr filtru), zobrazí se rozbalovací nabídka. Pokud je potřeba zadat hodnotu (například filtru), zobrazí se modální okno. Modální okna jsou využívána i pro zobrazení vlastností. Nápověda k prvkům je řešena pomocí informačních bublin (někdy též tooltip).

Pokud má nový systém nahradit stávající proces zpracování videa, musí zvládat všechny kroky úprav videa. V Tabulce 3.1 uvádím v současnosti nejčastější případ užití, kdy je potřeba oříznout začátek a konec videa a před video vložit úvodní obrazovku s podmínkami použití.

3.3 Návrh API

Stejně jako uživatelské rozhraní, i serverové rozhraní je potřeba navrhnout a otestovat. Při návrhu je třeba dbát pravidel architektury *REST*. *REST* definuje, jakým způsobem lze pracovat s daty na serveru. K práci se využívají 4 metody – vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání dat (Delete). Tato čtveřice bývá označování zkratkou CRUD [9].

Případ užití: Vložit intro, oříznout videozáznam
ID: UC1
Účastníci: Zaměstnanec FIT Systém
Vstupní podmínky: 1. Zaměstnanec má k dispozici obrázek s intro a video. 2. Zaměstnanec otevřel internetové stránky s editorem.
Tok událostí: 1. Případ užití začíná nahráním videa a obrázku intra. 2. Uživatel vloží obrázek na časovou osu, v dialogu nastaví délku zobrazení intra. 3. Uživatel vloží video na časovou osu. 4. Uživatel si přehraje náhled videa a v místě plánovaného začátku a konce videa přehrávání pozastaví a provede rozdělení na 2 části. 5. Uživatel zvolí část po plánovaném konci a odstraní ji. 6. Uživatel zvolí část před plánovaným začátkem a odstraní ji. Ořezané video uchopí a přetáhne jej na časové ose hned za konec intra. 7. Uživatel potvrdí změny a vyplní parametry výsledného videa. 8. Systém generuje XML a nabízí jej ke stažení nebo zpracování na serveru.

Tabulka 3.1: Textová specifikace případů užití pro nejčastější úpravy záznamů přednášek.

Nejprve bylo potřeba sepsat operace, které bude API umožňovat. Jedná se o následující operace: vytvořit nový projekt, získat stav existujícího projektu, nahrát soubor ze zařízení, vložit nahraný soubor na časovou osu, přidat stopu, odebrat stopu, přidat filtr, odebrat filtr, přidat přechod, odebrat přechod, posunout položku, rozdělit položku na 2 části, odebrat položku z časové osy, smazat nahraný soubor, renderovat projekt, případně smazat projekt.

Dalším krokem bylo identifikovat zdroje – entity. Jednou z entit bude beze sporu **projekt**, dále **soubor** a **položka časové osy**. Dalšími entitami jsem určil **přechod**, **filtr** a **stopu**. Operace je možné provádět jen nad zdroji, po identifikaci zdrojů je potřeba přiřadit operace ke zdrojům.

K projektu jsem přiřadil: vytvořit nový projekt, získat stav existujícího projektu, renderovat projekt, smazat projekt. Nad souborem bude možné provádět: nahrát soubor ze zařízení, vložit nahraný soubor na časovou osu, smazat nahraný soubor. Položka časové osy bude umožňovat: posunout položku, rozdělit položku na 2 části, odebrat položku z časové osy. Filtr bude možné přidat a odebrat, stejně tak přechod a stopu bude možné přidat a odebrat.

REST rozlišuje 4 operace. Nyní je potřeba přiřadit operacím jednu z následujících HTTP metod: POST (Create), GET (Retrieve), PUT (Update), DELETE (Delete). Nové prostředky se vytváří při vytváření nového projektu, nahrávání souboru ze zařízení, přidání filtru, přechodu a stopy, pro tyto operace byla zvolena metoda POST. Získávání stavu projektu je jednoznačně metoda GET. Pro rušení prostředků slouží metoda DELETE – odebrání filtru, přechodu, stopy, položky z časové osy, smazání nahraného souboru, smazání projektu. Na zbývající operace lze použít metodu PUT – vložit nahraný soubor na časovou osu, posunout položku, rozdělit položku na 2 části.

I ideálním případě nevzniknou kolize a každý zdroj bude mít přiřazenou maximálně jednu operaci k jedné HTTP metodě. V tomto případě má zdroj **položka na časové ose**

dvě metody PUT – posunout položku, rozdělit položku na 2 části. V této chvíli je potřeba operace dále rozlišit, přesun identifikovat jako `move` a rozdělení položek jako `split`.

Aby nemohlo dojít ke kolizi mezi URL určených pro webové stránky a pro API, mají veškeré požadavky na API prefix `api`. Do budoucna to umožní například verzovat API. Výsledný návrh API je uveden v následujícím seznamu:

- project
  - POST: `/project`
  - GET: `/project/{projectID}`
  - DELETE: `/project/{projectID}`
  - PUT: `/project/{projectID}`
- file
  - POST: `/project/{projectID}/file`
  - DELTE: `/project/{projectID}/file/{fileID}`
  - PUT: `/project/{projectID}/file/{fileID}`
- filter
  - POST: `/project/{projectID}/filter`
  - DELETE: `/project/{projectID}/filter`
- transition
  - POST: `/project/{projectID}/transition`
  - DELETE: `/project/{projectID}/transition`
- track
  - POST: `/project/{projectID}/track`
  - DELETE: `/project/{projectID}/trac`
- item
  - DELETE: `/project/{projectID}/item`
  - PUT: `/project/{projectID}/item/move`
  - PUT: `/project/{projectID}/item/split`

Požadavek na *REST* API musí být úplný, *REST* API je bezstavové. Například požadavek DELETE `/project/{projectID}` je úplný, serveru říkáme, že má smazat projekt a zároveň i uvádíme identifikátor projektu. U některých požadavků je ale potřeba blíže specifikovat parametry požadavku. Například u požadavku POST: `/project/{projectID}/filter` nevíme, jaký filtr se má aplikovat a ani nevíme, na jakou položku. Vyjma metody GET mohou mít požadavky tělo s parametry. Parametry mohou být předány v libovolném formátu, v projektu používám *application/json* a pro nahrávání souboru *multipart/form-data*. Předchozí požadavek je upřesněn parametry `track`, `item`, `filter` a `params` (parametry filtru). Kompletní specifikace navrženého API se nachází v dokumentaci API. API jsem zdokumentoval dle specifikace *OpenAPI 3.0*[4] pomocí online nástroje *Swagger Editor*[5].

[4]OpenAPI 3.0 – komunitní specifikace pro popis REST API, https://www.openapis.org/.

[5]Swagger Editor – open source online editor pro dokumentování API, https://editor.swagger.io/.

Kapitola 4

Práce s multimédii

Následující kapitola se zaměřuje na teoretické základy práce s multimédii. V projektu se s multimédii pracuje jak na serverové části, tak v části klientské (ve webovém prohlížeči). První část kapitoly se věnuje multimediálnímu frameworku *MLT*, který se používá v serverové části k renderování výsledného videa. Druhá část je věnována práci s multimédii ve webovém prohlížeči, zejména s ohledem na možnost zobrazení předběžného náhledu výsledného videa.

4.1 MLT framework

Jedná se o sadu nástrojů s otevřeným zdrojovým kódem s licencí LGPL-2.1, která byla původně vytvořena pro televizní vysílání. Poskytuje nástroje pro streamování, videoeditory, multimediální přehrávače, konvertory a další typy multimediálních aplikací. Framework MLT umožňuje vytvářet, spravovat a přehrávat audio nebo video projekty s více stopami. Je používán jako základní komponenta v nelineárních videoeditorech. Využití nalezne i mimo klasické desktopové videoeditory, může být použit jako součást většího systému nezávisle na použitém programovacím jazyce.

Framework lze používat prostřednictvím programu `melt` v příkazovém řádku nebo prostřednictvím *MLT API* knihovny v jazyce C. Pro napojení API na jiné programovací jazyky je nutné použít *SWIG*[1], nástroj, který překládá volání funkcí API na volání funkcí v C++. V C++ funkcích se následně volají funkce programu MLT v jazyce C. Druhou možností je provádět serializaci/deserializaci dat vlastnoručně a s MLT komunikovat prostřednictvím formátu XML. Vnitřní struktura, kterou *MLT API* vytváří je blízká formátu XML. V této práci se zabývám využitím MLT XML.

4.1.1 Zprovoznění

Framework nemá žádné jiné závislosti, než standard C99 a knihovny standardu POSIX. Zprovoznění zahrnuje nainstalování programu `melt` (v případě Ubuntu z repozitáře) a nainstalovaní programů a kodeků pro práci s multimédii.

```
$ sudo apt install melt
$ sudo apt install ladspa-sdk ffmpeg
$ sudo apt install vlc #Volitelne, pro prehrani vystupu
```

[1] SWIG – nástroj pro vývoj softwaru umožňující propojit program napsaný v jazyce C/C++ s jiným jazykem, http://swig.org/.

4.1.2 Základní příkazy

Jednoduché úkony lze provádět volbou parametrů při spuštění programu. Pro složitější projekty bude potřeba použít serializaci dat a formát XML. MLT umí soubory XML načítat i generovat.

```
#Prehrani na obrazovce
$ melt VIDEO0004.mp4 -consumer sdl
#Predpis pro vytvoreni sedotonoveho videa serializovany v~XML souboru
$ melt VIDEO0004.mp4 -filter greyscale -consumer xml > task.mlt
#Renderovani souboru output.avi na zaklade XML souboru
$ melt task.mlt -consumer avformat:output.avi acodec=libmp3lame vcodec=libx264
```

4.1.3 Používání formátu XML

Nejprve je nutné uvést seznam vstupních mediálních souborů. Vstupný soubor je označen jako `producer`. U vstupu se v `property` uvádí hodnota `resource`, s cestou k souboru. Tento jednoduchý soubor po spuštění přehraje celý soubor VIDEO001.mp4. Toto XML může být užitečné, pokud chceme soubor pouze zkonvertovat do jiného formátu.

```
1 <?xml version="1.0"?>
2 <mlt>
3   <producer id="producer0">
4     <property name="resource">/home/vladan/VIDEO0001.mp4</property>
5   </producer>
6 </mlt>
```

Obvykle potřebujeme pracovat s více soubory. K tomuto účelu slouží `playlist`. Stanovíme vstupy (`producer`) s unikátním `id`. V tuto chvíli by se přehrál pouze poslední soubor, který uvedeme, neboť framework MLT přehraje vždy poslední element uvnitř kořenového elementu `<mlt>` . Definujeme `playlist` obsahující položky `entry`. Vstupní soubory jsou na výstup řazeny ve stejném pořadí v jakém jsou uvedeny v playlistu. U položky playlistu můžeme uvést čas (od/do) pokud chceme vložit část vstupního souboru. Časové údaje lze zadávat buď jako počet snímků nebo jako čas ve formátu `00:00:00,000`, kde první dvojice číslic jsou hodiny a poslední tři číslice udávají milisekundy. Omezení se ignoruje, pokud uvedeme větší číslo, než je počet snímků, či délka souboru. Vkládáme-li obrázek, pak hodnota `out` udá délku zobrazení obrázku. S tímto XML zvládneme sestříhat více souborů do jednoho, vystřihnout určitou pasáž, vložit intro a outro.

```
1  <?xml version="1.0"?>
2  <mlt>
3    <producer id="producer0">
4      <property name="resource">/home/vladan/VIDEO0004.mp4</property>
5    </producer>
6    <producer id="producer1">
7      <property name="resource">/home/vladan/VIDEO0001.mp4</property>
8    </producer>
9    <producer id="producer2">
10     <property name="resource">/home/vladan/lego1.png</property>
11   </producer>
12   <playlist id="playlist0">
13     <entry producer="producer2" in="0" out="50"/>
14     <entry producer="producer1" in="0" out="999"/>
15     <entry producer="producer0" in="0" out="100"/>
16   </playlist>
17 </mlt>
```

Chceme-li na video aplikovat filtry, musíme vytvořit element `tractor` obsahující `multitrack` se seznamem tracků (odkazy na playlisty). V tractoru definujeme filtry. Filtry aplikujeme na `track` (playlist). Pokud budeme chtít vzít vstupní video, aplikovat na něj černobílý filtr, musíme vytvořit `playlist` s jedním zdrojem a `tractor` s jedním trackem.

```
<?xml version="1.0"?>
<mlt>
  <producer id="producer0">
    <property name="resource">/home/vladan/VIDEO0002.mp4</property>
  </producer>
  <playlist id="playlist0">
    <entry producer="producer0"/>
  </playlist>
  <tractor id="tractor0">
    <multitrack>
      <track producer="playlist0"/>
    </multitrack>
    <filter mlt_service="greyscale" track="0"/>
  </tractor>
</mlt>
```

Pokud potřebujeme aplikovat filtr na část videa, musíme vytvořit dva playlisty – jeden pro část na kterou bude aplikován filtr a druhý playlist pro část bez filtru. Pro synchronizaci playlistů použijeme `blank`, v jednu chvíli se může zobrazovat pouze jeden z playlistů, přednost má ten později definovaný. V následujícím případě se v „playlist1“ prvních 100 snímků přehrává „VIDEO0002“, v „playlist2“ je po tuto dobu nastaven stav `blank`. Pokud chceme zobrazovat statickou překryvnou vrstvu, můžeme využít filtru `watermark`. Pokud je filtr přizpůsobitelný parametry, uvedou se uvnitř elementu `filter` stejným způsobem, jako se uvádí parametry u elementů `producer`.

```
<?xml version="1.0"?>
<mlt>
  <producer id="producer0">
    <property name="resource">/home/vladan/VIDEO0002.mp4</property>
  </producer>
  <playlist id="playlist0">
    <entry producer="producer0" in="0" out="99"/>
    <blank length="50"/>
    <entry producer="producer0" in="150"/>
  </playlist>
  <playlist id="playlist1">
    <blank length="100"/>
    <entry producer="producer0" in="100" out="149"/>
  </playlist>
  <tractor id="tractor0">
    <multitrack>
      <track producer="playlist0"/>
      <track producer="playlist1"/>
    </multitrack>
    <filter mlt_service="watermark" track="0">
      <property name="resource">transparent.png</property>
    </filter>
    <filter mlt_service="greyscale" track="1"/>
  </tractor>
</mlt>
```

Více stop (`track`) může být přehráváno současně pouze použijeme-li filtr nebo přechod. Přechody (`transition`) se aplikují stejně jako filtry uvnitř tractoru. U přechodu nasta-

víme snímek náběhu a dokončení přechodu (`in` a `out`) a dále specifikujeme typ přechodu (`mlt_service`) dle https://www.mltframework.org/plugins/PluginsTransitions/, počáteční stopu přechodu (`a_track`) a cílovou stopu přechodu (`b_track`). Tímto způsobem můžeme vytvořit intro nebo outro s přechody, či prolínání jednotlivých video souborů. Obě stopy by se měly překrývat pouze po dobu přechodu. Pokud by například přechod skončil dříve, než by začala druhá stopa, docházelo by ke krátkodobému probliknutí první stopy.

```
<mlt>
  <producer id="producer0">
    <property name="resource">lego1.png</property>
  </producer>
  <producer id="producer1">
    <property name="resource">VIDEO0001.mp4</property>
  </producer>
  <playlist id="playlist0">
    <entry producer="producer0" in="0" out="74"/>
  </playlist>
  <playlist id="playlist1">
    <blank length="50"/>
    <entry producer="producer1"/>
  </playlist>
  <tractor id="tractor0">
    <multitrack id="multitrack0">
      <track producer="playlist0"/>
      <track producer="playlist1"/>
    </multitrack>
    <transition mlt_service="luma" in="50" out="74" a_track="0" b_track="1"/>
  </tractor>
</mlt>
```

S pomocí těchto konstrukcí jsme schopni nadefinovat nejčastější operace, které se při zpracovávání přednášek provádí.

4.2 Práce s videi v prohlížeči

S příchodem HTML5 prvků `<video>` a `<audio>` a jejich JavaScriptového API vzniká možnost práce s multimédii na webu. Práci s multimediálními prvky implementovali od roku 2010 všechny moderní prohlížeče. Za moderní desktopové prohlížeče považuji program Google Chrome, Safari, Mozilla Firefox, Opera, Microsoft Edge. Prohlížeč Internet Explorer má sice dle serveru StatCounter 5,48% zastoupení, což jej řadí mezi Microsoft Edge a Operu [13], avšak i Microsoft jej v únoru označil za „řešení pro kompatibilitu“, nikoliv webový prohlížeč [8]. Po představení prohlížeče Microsoft Edge získává Internet Explorer bezpečnostní záplaty, ale neprobíhá implementace nových webových standardů. Při vývoji nových projektů by tedy nekompatibilita s Internet Explorer neměla být brána v potaz. Následující kapitoly čerpají z knihy *HTML5 - audio a video: kompletní průvodce* [11] a z komunitního projektu MDN Web Docs[2].

4.2.1 Kódování mediálních zdrojů

Veškerá data v počítači je potřeba kódovat. Na vyšší úrovni pohlížíme na kódování jako na formát dat. Formáty kódování audia jsou například MP3, AAC, Vorbis, FLAC a formáty

[2]MDN Web Docs – portál s webovou dokumentací provozovaný společností Mozilla, https://developer.mozilla.org/.

kódování videa například MPEG-2, MPEG-4 , H.264, HEVC, Theora, VP9, AV1. Z pohledu uživatele se liší zejména licencí pro použití a způsobem komprese dat. Multimediální soubory mohou obsahovat více video a audio stop, titulky či jiná podpůrná data, proto se multimediální soubory vždy obalují do multimediálních kontejnerů, jako například AVI, MP4, FLV, RealMedia, Matroška. Výběr formátu videa tak typicky zahrnuje výběr multimediálního kontejneru, formátu kódování videa a formátu kódování audia. Kodek je zařízení nebo počítačový program, který dokáže převádět zakódovaná data do jiné podoby, v případě webových aplikací je kodekem internetový prohlížeč. V době, kdy organizace W3C[3] vytvářela specifikaci pro HTML5 prvky `<video>` a `<audio>`, byla snaha standardizovat formát, který by byl podporován ve všech prohlížečích, neúspěšná kvůli požadavku na royalty free licenci [11]. Od té doby se nejvíce rozšířil formát MP4 H.264, který ovšem odmítá Mozilla, a Theora, který odmítá Apple. Pokud se podíváme na srovnání podporovaných kodeků dnes, tabulka 4.1, zjistíme, že situace není stále ideální.

Kontejner	Video	Audio	Chrome	Firefox	Edge	Opera	Safari
WebM	VP8	Vorbis	ano	ano	s 2 rozšířeními	ano	ne*
WebM	VP9	Opus	ano	ano	rozšíření, HW dekodér	ano	ne
WebM	AV1	Vorbis	ano	ano	s beta rozšířením	ano	ne
WebM	AV1	Opus	ano	ano	s beta rozšířením	ano	ne
Ogg	Theora	Vorbis	ano	ano	s rozšířením	ano	ne
MP4	H.264	MP3	ano***	ano**	ano	ano	ano
MP4	H.264	AAC	ano***	ano**	ano	ano	ano

Tabulka 4.1: Přehled široce podporovaných formátů videa internetovými prohlížeči. *Safari vyžaduje pro VP8 program *Perian* (vývoj ukončen). **Mozilla Firefox pro Linux vyžaduje pro H.264 program *GStreamer* nebo *FFmpeg*. ***Google Chrome H.264 podporuje, ale Chromium vyžaduje FFmpeg.

Pokud bychom chtěli použít pouze jeden formát, můžeme zvolit H.264 video s AAC audiem uvnitř MP4 nebo VP8 video s Vorbis audiem uvnitř WebM. V prvním případě budou muset uživatelé prohlížeče Firefox na Linuxových systémech a uživatelé prohlížeče Chromium nainstalovat program *FFmpeg*, v druhém případě budou muset uživatelé Safari nainstalovat program *Perian* a uživatelé Microsoft Edge *Rozšíření pro webová média* a *Rozšíření pro video VP9*.

Chceme-li mít jistotu, že uživatel video přehraje, pak musíme zvolit formáty dva – H.264 s MP3/AAC uvnitř MP4 pro uživatele Safari a Edge a pro ostatní uživatele variantu Theora s Vorbis uvnitř Ogg.

4.2.2 Použití prvku video

Prvek `<video>` přinesla specifikace HTML5 a umožňuje přehrávat video na webové stránce v režii prohlížeče. Pokud nespecifikujeme jinak, prohlížeč zajistí načtení zdroje a vykreslení ovládacích prvků. S videem můžeme pracovat skrze atributy a HTMLMediaElement API. Prvek je podporován všemi rozšířenými prohlížeči s výjimkou Opera Mini (kvůli spořiči přenesených dat).

Video prvek má celkem 6 atributů. Atributy `autoplay`, `loop` a `controls` mohou nabývat booleovských hodnot, a proto jejich uvedení znamená logickou 1, pokud atributy neu-

[3] World Wide Web Consortium (W3C) – mezinárodní sdružení vytvářející webové standardy, https://www.w3.org/.

vedeme, jsou atributy s hodnotou logické 0. Pro potřeby náhledu videa atribut `autoplay` nepoužijeme, video budeme přehrávat skrze API metody `play`, `loop` rovněž není žádoucí, video by se přehrávalo ve smyčce, při neuvedení se přehrávání na konci zastaví. Atribut `controls` zobrazí ovládací prvky, což je nežádoucí, použijeme vlastní ovládací prvky. Atribut `preload` navrhuje prohlížeči, kolik dat se má načíst ze zdroje, dostupné hodnoty jsou „none“, „metadata“ a „auto“. Hodnota „none“ nenačte nic, „metadata“ zajistí načtení délky videa a prvního snímku, „auto“ nechá volbu na prohlížeči. V případě editoru postačuje výchozí hodnota „auto“. Atribut `poster` slouží k nahrazení prvního snímku zástupným obrázkem, v projektu není potřebný. Praktické jsou elementy `width` a `height`, udávají šířku a výšku videa. Rozměry jsou chápány jako maximální možné, pokud je rozměr videa větší, zmenší se ve výchozím nastavení se zachováním poměru stran. Zarovnání videa v boxu určuje CSS vlastnost `object-position` s dvojicí hodnot pro vodorovnou a svislou pozici. Jak se má měnit měřítko obsahu videa udává vlastnost `object-fit` s hodnotami „fill“, „contain“ a „cover“.

Zdroj videa lze nastavit dvěma způsoby. Atributem `src` nebo vnořenými prvky `<source>`. Vnořené prvky `<source>` poskytují více možností skrze vlastní atributy. Zdrojů můžeme uvést více s různými formáty – atribut `type` (MIME typ mediální zdroje) nebo `media` (mediální zdroj pro konkrétní typ zařízení). Prohlížeč sám vybere pro něj nejlepší zdroj. Pokud měníme zdroje dodatečně, musíme po změně zavolat metodu `load`, která provede inicializaci video prvku. Druhou možností je uvést zdroj v atributu `src` prvku `<video>`. Takto je možné nastavit jen jeden zdroj současně, a proto si musíme kompatibilitu formátu sami skrze metodu `canPlayType` nebo knihovny Modernizer[4]. Druhá metoda je vhodná pokud používáme jeden zdroj.

4.2.3 Použití prvku audio

Prvek pro přehrávání audia v prohlížeči je možné rovněž upravovat pomocí HTMLMediaElement API a pomocí atributů. Prvek `<audio>` narozdíl od videa nemá atributy `poster`, `width`, `height`, protože jsou v prohlížeči vykresleny pouze ovládací prvky bez vizualizace audia. Audio prvek lze využít pro zvukové stopy. Nastavení a význam hodnot atributů je stejný jako u prvku `<video>`.

4.2.4 Ovládání videa

Přehrávání videa lze ovládat pomocí JavaScriptu skrze HTMLMediaElement API. Stejné API používají i prvky `<audio>`, lze pro ně použít stejný postup. Základními metodami pro přehrávání jsou `play` a `pause`. Přehrávání videa se na konci zastaví (pokud není nastaven atribut `loop`). Pokud po dokončení přehrávání zavoláme znovu metodu `play`, přehrávání se spustí od začátku videa.

Pro skok na konkrétní pozici slouží atribut `currentTime`. Atribut je určen pro čtení (získání aktuálního času) i pro zápis (skok na pozici). Celkovou délku zjistíme přečtením atributu `duration`. Délku musíme zjišťovat až po získání metadat – vyvolání události `loadedmetadata`.

Další užitečnou událostí je `timeupdate`, která se volá v intervalech 15 ms až 250 ms (Mozilla Firefox 100-250 ms) po čas přehrávání a umožňuje průběžně efektivně pracovat s atributem `currentTime` a například zobrazovat aktuální pozici přehrávání.

[4]Modernizer – sada nástrojů pro testování funkcí prohlížeče, https://modernizr.com/.

Použití událostí a metod pro jednoduchý přehrávač videa řízený JavaScriptem ukazuje obrázek 4.1.


```
function onLoadMetadata() {
    duration.innerText = video.duration;
    time.innerText = video.currentTime;
}

function onTimeUpdate() {
    time.innerText = video.currentTime;
}

function toggleReset() {
    video.pause();
    video.currentTime = 0;
}

function togglePlay() {
    video.play();
}

function togglePause() {
    video.pause();
}

function togglePrev() {
    video.currentTime = 0;
}

function toggleNext() {
    video.currentTime = video.duration;
}
```

Obrázek 4.1: Jednoduchý přehrávač videa demonstrující využití HTMLMediaElement API.

4.2.5 Posloupnost videí

Při dosažení konce média je vyvolána událost `ended`. Této události lze využít pro vytváření posloupnosti videí. Plocha s náhledem je tvořena dvojicí videí. Na popředí je aktuálně přehrávající se videa a na pozadí je skryté druhé video, které se s předstihem nastaví na nový zdroj, počáteční pozici a načtou se metadata. Jakmile přehrávání videa v popředí skončí a je vyvolána událost `ended`, dojde k prohození videa v popředí s videem v pozadí a spustí se nové video v popředí. Přístup dvou scén, kdy jedna je viditelná a druhá se připravuje, je inspirován programem OBS Studio[5], který slouží na nahrávání a streamování videí. Ve webovém prohlížeči určuje pořadí objektů CSS vlastnost `z-index`. Hodnotou je celé číslo, objekty s vyšší hodnotou překryjí objekty s nižší hodnotou. Prvky se musejí překrývat, to lze zajistit absolutním pozicování uvnitř kontejneru. Následující kód obsahuje funkci pro přehození video souborů a pro resetování přehrávače do původního stavu.

```
let videoMain = video1;
function videoSwap() {
    videoMain = video2;
    video1.style.zIndex = 1;
    video2.style.zIndex = 2;
    togglePlay();
}

function toggleReset() {
    videoMain.pause();
    video1.currentTime = 0;
    video2.currentTime = 0;
    video1.style.zIndex = 2;
    video2.style.zIndex = 1;
```

[5]OBS Studio – open source software pro nahrávání a streamování videí, https://obsproject.com/.

```
15      videoMain = video1;
16  }
```

Pro posloupnost více videí je zapotřebí „podavač“ videí, který přehrávači předá další klip.

4.2.6 Přechod mezi videi

Při stříhání videí se může hodit efekt prolnutí mezi 2 klipy. Při tomto přechodu se určitý čas před koncem přehrávaného videa začne přehrávat i video následující a současné video se postupně zprůhlední. Průhlednost elementů lze nastavit CSS stylem `opacity`, přičemž hodnota 1 značí neprůhledný objekt a hodnota 0 zcela průhledný objekt. Samotné prolnutí (zvyšování průhlednosti) lze provést CSS animacemi nebo snižováním hodnoty `opacity` s každým vyvoláním události `timeupdate`. Použití animací sice vytvoří plynulý efekt, ale nastává problém při pozastavení a obnovení videa v místě přechodu, neboť by musela být animace pozastavena, aktuální stav uložen a posléze vytvořena nová animace pro zbývající úsek. Řešení pomocí `timeupdate` poskytuje pro náhled dostatečnou kvalitu a provést pozastavení či skok je v tomto případě snadné. Pokud chceme aplikovat prolnutí, stačí nastavit délku přechodu a poté zaregistrovat následující funkci pro obsluhu události `timeupdate`.

```
1  videoMain.addEventListener('timeupdate', transitionLuma, false);
2  function transitionLuma() {
3      const timeToEnd = videoMain.duration - videoMain.currentTime;
4      if (timeToEnd <= transitionDuration) {
5          videoMain.style.opacity = (timeToEnd / transitionDuration);
6          if (!transition) {
7              videoBack.play();
8              transition = true;
9          }
10     }
11 }
```

Na každou událost lze zaregistrovat více obslužných funkcí, po skončení přechodu je nutné obsluhu funkcí `transitionLuma` odregistrovat. Pokud je zbývající čas menší nebo roven délce přechodu, nastaví se `opacity` na hodnotu rovnou podílu zbývajícího času s délkou přechodu (klesá rovnoměrně od 1 k 0). Ve webovém prohlížeči lze sestrojit i jiné přechody s odsunutím, zvětšováním, postupným odkrýváním, ale tyto filtry nejsou pro tvorbu přednášek vhodné a proto je není potřeba implementovat.

Stejnou technikou se sestrojí roztmívání a zatmívání obrazu (přechod z černé / do černé), pouze bude druhá scéna až za podkladovou černou vrstvou (`z-index se zápornou hodnotou`).

4.2.7 Filtry

Pokud potřebujeme aplikovat filtr obrazu, nabízí se dvě možnosti – řešení pomocí CSS filtrů a řešení pomocí vykreslování do Canvas. Obě varianty mají své přednosti, CSS najde uplatnění u jednoduchých filtrů, vykreslování do Canvas nám dá absolutní kontrolu nad výsledným videem.

Obrazové filtry pomocí CSS

HTML5 obsahuje pracovní návrh CSS vlastnosti `filter`. Vlastnost aplikuje grafické efekty na libovolné objekty, včetně obrázků a video elementů. CSS vlastnost má několik definovaných filtrů a možnost odkazovat na SVG filtry. Ačkoliv se jedná o pracovní návrh, CSS

filtry jsou podporovány všemi rozšířenými prohlížeči (Internet Explorer je nepodporuje) [2]. Microsoft Edge podporuje předdefinované filtry, ale kvůli chybě nepodporuje odkazování na SVG filtry. Největší předností CSS filtrů je způsob aplikování na video. Filtr je uložen v CSS třídě, případně lze přiřadit přímo k elementu. U `<video>` elementu stačí nastavit třídu filtru a filtr se aplikuje, vykreslování zajistí prohlížeč. Jaké předdefinované filtry vlastnost `filter` nabízí, ukazuje tabulka 4.2.

Funkce filtru	Výchozí	Možné hodnoty	Chování filtru
brightness	1	>=0% / >=0	0% černý obraz, >100% větší jas
contrast	1	>=0% / >=0	0% šedý obraz, >100% větší kontrast
saturate	1	>=0% / >=0	0% vybledlý, >100% větší sytost
grayscale	0	0%-100% / 0-1	0% původní obraz, 100% šedotónový
invert	0	0%-100% / 0-1	0% původní obraz, 100% invertovaný
sepia	0	0%-100% / 0-1	0% původní obraz, 100% sépiový
opacity	1	0%-100% / 0-1	0% průhledný, 100% neprůhledný
hue-rotate	0	úhel (deg/grad/rad/turn)	posun barev (s periodou 360°)
blur	0	vzdálenost, vyjma %	rozmazání o danou vzdálenost
drop-shadow	none	viz vlastnost `box-shadow`	efekt zrcadlení

Tabulka 4.2: Seznam definovaných filtrů. Klíčovými filtry jsou jas, kontrast a sytost.

Poslední možnou hodnotou, které může `filter`, je „url()“. V tomto případě se nejedná o filtr, ale o odkaz na SVG element. SVG element se používá jako kontejner pro SVG filtry. SVG filtry jsou z pohledu standardu ve stavu „doporučení“. Aktuálně kvůli chybě nefungují v prohlížeči Microsoft Edge. Po aplikování filtrů bylo patrné snížení počtu snímků za sekundu, při aplikování jednoho filtru bylo přehrávání videa dostatečně plynulé. Pokud bychom chtěli aplikovat více obrazových filtrů současně (4 a více), vyplatí se realizovat filtry pomocí Canvas. K dispozici je 17 filtrů, jejich přehled uvádím v tabulce 4.3. Pro potřeby úprav videí nevyužijeme zdaleka všechny funkce, lze se tedy omezit na předdefinované filtry, které fungují i v prohlížeči Microsoft Edge.

JavaScript umožňuje výpočetní operace nad daty pro zvýšení výkonu přesunout na pozadí do samostatného vlákna, pro tuto techniku se používá termín Web Workers[6]. Přímo v prohlížeči můžeme například analyzovat obsah videa a detekovat v něm objekty (například rozpoznávat obličeje). Vytváření vláken přináší režii, která se vyplatí až při náročných skriptech, aplikování filtrů na videa pomocí Web Workers počet zpracovaných snímků za sekundu zvýší minimálně, ve většině prohlížečů spíše způsobí snížení (v závislosti na implementaci Web Workers).

Obrazové filtry pomocí Canvas

Zatímco CSS a SVG filtry umožňují manipulovat s výsledným obrazem na základě přiřazení CSS stylů a tříd, Canvas přestavuje kreslící plátno složené z 2D matice pixelů. Jedná se o bitmapu, jejíž vykreslení musíme sami zajistit. Vykreslováním do canvasu lze dosáhnout většího počtu vykreslených snímků za sekundu, než při použití CSS/SVG filtrů. Canvas je vhodné použít tehdy, pokud nepotřebujeme přístup k objektům filtrů pomocí DOM. Rychlost vykreslení závisí na velikosti plátna. Při velkém formátu je výhodnější SVG než canvas. Obě techniky je možné kombinovat.

[6]Web Workers – standardizované API pro běh skriptů na pozadí, https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers.

Filtr	Popis filtru	Možné použití
feBlend	míchání barev objektů/bitmap	přidání vodoznaku
feColorMatrix	změna barev pixelů transformační maticí	barevný odstín, sytost
feComponentTransfer	úprava barevných složek	jas, kontrast
feConvolveMatrix	aplikování konvoluční matice na obraz	rozmazání, doostření
feGaussianBlur	Gaussovské rozmazání obrazu	rozmazání
feComposite	operace nad skládáním SVG objektů	
feDiffuseLighting	aplikace difúzní složky osvětlovacího modelu	
feDisplacementMap	nahrazení pixelů pixely z jiného zdroje	
feDropShadow	vytvoření efektu stínu/zrcadlení	
feFlood	vyplnění objektu konstantní barvou	
feImage	převádí externí zdroj na bitmapu	
feMerge	aplikuje filtry současně, ne postupně	
feMorphology	zvýšení/snížení tloušťky objektů	
feOffset	posun objektů v ose x, y	
feSpecularLighting	aplikace lesklé složky osvětlovacího modelu	
feTile	vyplnění objektu texturou	
feTurbulence	generování textury pomocí Perlinova šumu	

Tabulka 4.3: Seznam SVG filtrů s možným využitím ve videoeditoru.

Pro zobrazení videa skrze `<canvas>` bude zapotřebí jak `<canvas>`, tak i `<video>` element. Z video elementu se budou vzorkovat snímky a vykreslovat jako bitmapy v `<canvas>`. Ovládání videa bude stále skrze JavaScriptové API `<video>` elementu, ale zobrazení obstará `<canvas>`. Z toho důvodu je nutné video element skrýt pomocí CSS vlastnosti `display: none`.

Během přehrávání je potřeba zajistit pravidelné překreslování Canvasu. Lze použít události `timeupdate`, ale tato událost je vyvolána jen každých 15 ms až 250 ms, což na pohodlné přehrávání nestačí. Druhou možností je nastavit obsluhu události `play` na funkci, která bude zajišťovat překreslování. Na konci funkce pro překreslení se otestuje, zda se video stále přehrává (`video.paused || video.ended`) a případně se naplánuje opětovné zavolání funkce za 0 milisekund – `setTimeout(repaint, 0)`. V některých prohlížečích se událost `timeupdate` nevyvolá po načtení metadat a prvního snímku, pro tyto prohlížeče pomůže funkci vykreslení zavolat i při události `loadedmetadata`. Následující ukázka získá aktuální snímek z `<video>` elementu, tyto data si zkopíruje do pomocného pole a poté postupně projde a modifikuje jednotlivé pixely (v tomto případě sníží jas na polovinu). Pole pixelů je organizováno po řádcích, přičemž každý pixel se skládá ze 4 po sobě jdoucích složek (RGBA) – každý pixel má v poli velikost 4. Po modifikaci pixelů dojde k vyčištění původního plátna (nastaví se průhledné plátno) a poté se vykreslí modifikovaný obraz. Pokud se video stále přehrává, naplánuje se ihned další spuštění funkce `paintFrame`.

```
1 video.addEventListener('play', paintFrame, false);
2 const context = canvas.getContext('2d');
3
4 function paintFrame() {
5     context.drawImage(video, 0, 0, 480, 270, 0, 0, 480, 270);
6     const frame = context.getImageData(0, 0, 480, 270);
7     const output = context.createImageData(480, 270);
8
9     for (let x = 0; x < 480; x++) {
```

```
10          for (let y = 0; y < 270; y++) {
11              let index = 4 * (x + 480 * y);
12              output.data[index + 0] = frame.data[index + 0] / 2; // R
13              output.data[index + 1] = frame.data[index + 1] / 2; // G
14              output.data[index + 2] = frame.data[index + 2] / 2; // B
15              output.data[index + 3] = frame.data[index + 3]; //
16          }
17      }
18      context.clearRect(0, 0, 480, 270);
19      context.putImageData(output, 0, 0);
20      if (!video.paused && !video.ended) {
21          setTimeout(paintFrame, 0);
22      }
23  }
```

V ukázce není vyřešeno škálování. Metoda `drawImage` vezme originální obraz, nikoliv zmenšený obraz `<video>` elementu. Pro bezpečné použití by bylo zapotřebí zjišťovat větší rozměr videa a ten přizpůsobit velikosti plátna. Dále by muselo být video ručně centrováno. V prohlížeči Mozilla Firefox se videa natáčená na výšku zobrazují v `<canvas>` otočená o 90°. Z tohoto důvodu jsem od filtrů pomocí Canvas opustil.

Canvas najde uplatnění zejména v případech, kdy pracujeme s důvěryhodnými daty a provádíme nad nimi náročné operace.

4.2.8 Shrnutí

HTML5 je živý standard, který neustále rozšiřuje možnosti tvorby webového obsahu. Práce s multimédii je v HTML5 prakticky neomezená, ale ne všechny funkce jsou standardizovány. Při implementaci je nutné sledovat stav standardu a rozšířenost v prohlížečích. Vždy je nutné začít výběrem formátu multimédií. Pokud chceme mít jistotu, že každý uživatel bude mít prohlížeč s kompatibilním kodekem, musíme prohlížeči videa nabízet alespoň ve 2 formátech. Jednoduché úpravy obrazu provádíme pomocí CSS filtrů, na složitější filtry použijeme SVG filtry. Chceme-li přistupovat přímo k obrazovým datům, použijeme Canvas. Jednoduchý videoeditor si pro náhledové video vystačí s CSS filtry – dostatečně rychlé, na implementaci jednoduché a s podporou v moderních prohlížečích.

Kapitola 5

Realizace

V úvodu této kapitoly nastíním strukturu projektu, části, ze kterých se projekt skládá. Dále popisuji způsob instalace a zprovoznění, popisuji důležité soubory z hlediska nasazení a údržby projektu. Ve druhé části se věnuji roli jednotlivých částí při obsluze požadavku a poté jednotlivé části projektu podrobně rozebírám.

5.1 Struktura projektu

V kořenovém adresáři projektu se nacházejí konfigurační soubory, soubory správce balíčků a hlavní soubor pro spuštění projektu. Zdrojové soubory jsou uloženy v podadresářích, ze struktury projektu je patrná architektura MVC, obrázek 5.1.

```
/
├─ controllers      (zpracování požadavků, odeslání odpovědí)
├─ examples         (ukázkové soubory ke kapitole 4.2)
├─ models           (moduly pro práci s daty, s XML)
├─ public           (veřejný adresář: CSS, JS, fonty, ikony)
├─ react            (React komponenty, součástí View)
├─ views            (statické šablony – kostra GUI)
├─ WORKER           (adresář s MLT projekty)
├── .babelrc
├── .eslintrc.js
├── config.js
├── package.json
├── router.js
├── server.js
└── webpack.config.js
```

Obrázek 5.1: Struktura projektu.

Významu konfiguračních souborů a souborům v kořenovém adresáři se budu věnovat v následujících kapitolách. Jednotlivé části projektu rozebírám v druhé polovině této kapitoly.

5.2 Nastavení, konfigurační soubory

Po spuštění aplikace se začne vykonávat kód v souboru `/server.js`. Nejprve se nastaví prostředí pro zpracovávání požadavků. K tomu se využívají další soubory – `/config.js`, `/router.js` a soubory nainstalovaných balíčků.

5.2.1 Server

Soubor `/server.js` je hlavním souborem projektu. Při spuštění projektu začne interpret Node.js interpretovat právě tento soubor. Soubor má na starost inicializaci prostředí, konkrétně má na starost následující:

1. Inicializuje JavaScriptový framework *Express*.
2. Inicializuje rozšíření pro zpracování nahrávaných souborů (*busboy*).
3. Inicializuje rozšíření pro zpracování těl požadavků ve formátu *JSON*.
4. Nastaví šablonovací systém, z pohledu *Express* čisté HTML ve složce `/views`.
5. Nastaví router na `/router.js`.
6. Nastaví `/public` jako veřejný adresář, přístupný ze sítě.
7. Nastaví server, aby naslouchal na portu a adrese uvedené v souboru `config.js`.

Tyto činnosti se vykonávají pouze při spuštění serveru. Jakmile se provedou všechny úkony, server naslouchá požadavkům a příchozí požadavky předává routeru.

5.2.2 Config

Soubor `/config.js` je centrem všech konfiguračních direktiv. Před nasazením v novém prostředí je nutné hodnoty v souboru `/config.js` upravit dle aktuálních údajů. Prvních sedm údajů je závislých na prostředí, ve kterém je plánováno nasazení:

- `port` – číslo portu, na kterém bude server spuštěn. Port musí být volný. Porty s číslem menším než 1024 mohou vyžadovat oprávnění správce. Výchozí hodnotou je „8080“.
- `host` – doménové jméno. Například „localhost“ nebo „eva.fit.vutbr.cz“. Výchozí hodnotou je „localhost“.
- `emailServer` – doménové jméno SMTP serveru pro zasílání emailových upozornění, výchozí hodnotou je „smtp.stud.fit.vutbr.cz“.
- `emailPort` – číslo portu SMTP serveru pro zasílání emailových upozornění, výchozím portem je „465“.
- `emailUser` – uživatelské jméno účtu pro autentizaci na SMTP serveru. Z tohoto účtu jsou emailová upozornění zasílána. Výchozí hodnotu „xkudla15“ je nutné změnit.
- `emailPasswd` – heslo k účtu pro autentizaci na SMTP serveru. Výchozí hodnota nemá význam.
- `adminEmail` – emailová adresa administrátora. Adresa slouží k zasílání chyb administrátorovi, ke kontaktování. Výchozí hodnota je „xkudla15@stud.fit.vutbr.cz“.

Následuje položka `projectPath`, která udává umístění adresáře s podadresáři MLT projektů. Podadresáře obsahují jak nahrané soubory, tak i XML soubor pro program MLT. Výchozí hodnotu „WORKER“ není nutné měnit, pokud nepotřebujeme mít soubory například na jiném disku.

Poslední položkou je objekt `mapFilterNames`. Objekt slouží k mapování vlastních názvů filtrů (které nejsou součástí MLT) na názvy filtrů, které jsou součástí MLT. Objekt není nutné upravovat, pokud neimplementujeme další alias pro názvy filtrů.

Součástí konfiguračních direktiv jsou i funkce `serverUrl` a `apiUrl`, které zjednodušují přístup k adrese napříč projektem. Hodnoty se používají například při vytváření asynchronních požadavků v React komponentách. Funkce vracejí hodnoty pro protokol HTTP, v případě nasazení HTTPS je nutné tyto dvě funkce upravit.

5.2.3 Router

Routování zajišťuje soubor `router.js`, který se skládá z routovacích a middleware pravidel. Pravidla překládají adresu URL na JavaScriptovou funkci (controller), která zajistí obsluhu požadavku. Router z pohledu architektury MVC též spadá mezi controllery. Routovací pravidlo se obvykle aplikuje první, odpovídající URL požadavku, middleware pravidla se vykonají před routovacími pravidly a pokud v nich na konci zavoláme funkci `next`, vykonají se i pravidla následující. Middleware pravidlo je využito pro logování všech přístupů, do budoucna bude využito i pro autorizaci uživatele. Funkce pro obsluhu požadavku obdrží objekt s požadavkem, objekt pro budoucí odpověď a funkci `next`. V následující ukázce je první pravidlo middleware a druhé je routovací.

```
1 // Log access
2 router.use((req, res, next) => {
3     console.info(new Date(), ' @ ${req.originalUrl}');
4     next(); // go to the next routes
5 });
6 // API route
7 router.post('/api/project/:projectID/file', apiController.projectFilePOST);
```

V případě, že router nenalezne v seznamu pravidel odpovídající routovací pravidlo, vrací odpověď s kódem 404. Tělo odpovědi obsahuje HTML s textem „Cannot GET /fuu“, pro požadavek metody „GET“ s adresou „/fuu“.

Posledním middleware pravidlem je pravidlo pro odchytávání chyb, které se použije, pokud během řádné obsluhy požadavku dojde k chybě a je zavolána funkce `next`. Chybu zpracuje `errorController`.

```
1 // Error handling
2 router.use(errorController.default);
```

5.2.4 npm

Soubory `/package.json` a `/package-lock.json` slouží k uložení informací o projektu. Soubory používá program *npm*. V souboru `/package.json` jsou uloženy informace o projektu, jako je název projektu a popis, verze, GitHub repozitář, informace o autorovi a licenci, URL pro hlášení chyb a URL stránky projektu. Dále je zde uložen název hlavního JavaScriptového souboru (`server.js`) a skripty. Například pokud napíšeme v kořenovém adresáři příkaz `npm run dev-build`, vykoná se příkaz `./node_modules/.bin/webpack -wd`.

Nejdůležitější částí souboru je seznam závislostí. Pokud potřebujeme rozšířit funkcionalitu projektu a další funkce, můžeme sáhnout po balíčkovém systému *npm*[1]. K 22. dubnu 2019 bylo v systému registrováno téměř 810 tisíc balíčků [7]. Správce *npm* je výchozí balíčkový systém Node.js, jedná se obdobu správce závislostí `Composer` pro PHP. Příkazem `npm install <balicek> -save` provedeme stažení a nainstalování závislosti a zároveň je tato závislost uložena v souboru `/package.json`.

Pro zjednodušení soubor `/package-lock.json` neuvádím ve struktuře projektu. Soubor není pro funkčnost aplikace vyžadován. Pokud soubor neexistuje, *npm* si jej vytvoří. V tomto souboru je uložen aktuální stav nainstalovaných balíčků, zatímco v `/package.json` je seznam balíčků, které „měli“ být nainstalovány. Například pokud by měl být nainstalován balíček `react` ve verzi ^16.8.6, ve skutečnosti mohl být nainstalován balíček ve verzi 16.8.9. Díky tomu, že závislosti trvají pouze na hlavní verzi, mohou být instalovány nové verze, které jsou zpětně kompatibilní (například opravy chyb).

5.3 Zprovoznění projektu

Pro běh aplikace je nezbytné mít v systému nainstalované *Node.js*. Postup instalace je uveden na oficiálních stránkách[2]. Aplikace vyžaduje verzi Node.js 8 nebo novější. Nainstalovanou verzi si lze ověřit příkazem `node -version`. Pro získání závislostí je nutné mít nainstalovaný program *npm*. Měl by být součástí instalace *Node.js*. Nainstalovanou verzi lze ověřit příkazem `npm -version`.

Nejprve je potřeba projekt zkopírovat, nebo stáhnout z repozitáře na Github.com[3] tak, aby adresářová struktura projektu odpovídala obrázku 5.1. Adresář, ve kterém se nacházejí soubory projektu dále označuji jako *kořenový adresář* se symbolem „/“. Tímto adresářem se nemyslí kořen souborového systému. Projekt se může nacházet v libovolném adresáři, pro který máte oprávnění.

Externí závislosti nejsou součástí archivu (některé závislosti jsou závislé na architektuře procesoru). Seznam závislostí je uveden v souboru `package.json`. Stažení všech závislostí se provede příkazem `npm install` v kořenovém adresáři.

Před spuštěním je potřeba zkontrolovat a upravit nastavení v konfiguračním souboru. Popis konfiguračního souboru nabízí kapitola 5.2.2.

Další postup se odvíjí od účelu, za jakým bude server spuštěn. Postup zprovoznění pro účely testování je odlišný od postupu pro produkční nasazení na server. Rozlišení je důležité z hlediska výkonu a spolehlivosti serveru.

5.3.1 Spuštění ve vývojářském režimu

Pro spuštění se používají 2 následující příkazy.

```
npm run dev-build
npm run dev-start
```

První příkaz (`npm run dev-build`) kompiluje soubor s kaskádovými styly `views/style.scss` do souboru`/public/style.css` a také veškerý JavaScript a React komponenty ve složce `/react` do souboru `/public/main.js`. Po zadání příkazu provede program *Webpack* kompilaci, dokončení je indikováno následujícím výpisem:

[1] npm – balíčkový systém pro Node.js, https://www.npmjs.com/.
[2] Oficiální stránky Node.js, https://nodejs.org/en/download/.
[3] GitHub repozitář kudlav/videoeditor, https://github.com/kudlav/videoeditor.

```
Version: webpack 4.29.6
Time: 13391ms
Built at: 2019-05-01 12:00:00
...
```

V případě, že plánujeme upravovat soubor `views/style.scss` nebo soubory ve složce `/react`, necháme skript dále běžet. Skript bude hlídat soubory a v případě změny provede novou kompilaci. Pokud soubory neplánujeme upravovat, lze skript po první úspěšné kompilaci ukončit (`CTR+C`). Webpack lze konfigurovat v souboru `/webpack.config.js`. V souboru se nastavují vstupní a výstupní soubory a pravidla pro konverzi vstupů na výstupy.

Druhým příkazem spustíme server – `npm run dev-start`. Server bude běžet, dokud skript neukončíme nebo nedojde k neošetřené výjimce. Pokud dojde k chybě, server se zastaví a čeká na změnu souborů. Server lze restartovat ručně zadáním příkazu `rs`. Při změně JavaScriptových souborů (vyjma složky `/public`) je server automaticky restartován. Toto chování je dobré na vývoj, ale nevhodné pro produkční nasazení. Kompatibilitu kódu zajišťuje *Babel*, konfiguraci ukládá do souboru `/.babelrc`.

5.3.2 Produkční nasazení

Při prvním spuštění je nejprve potřeba zkompilovat JavaScriptový soubor, který je zasílán webovému prohlížeči uživatele. Klientský JavaScriptový soubor obsahuje hodnoty z konfiguračního souboru, proto je nutné provést překompilování zdrojů po každé změně konfigurace. JavaScriptové soubory se pro produkční prostředí liší od těch, generovaných pro testovací sestavení. Zkompilování zdrojů provede následující příkaz:

```
npm run build
```

Dále je vhodné nastavit v prostředí operačního systému proměnnou `NODE_ENV`. Díky tomuto nastavení bude *Express* používat cache pro šablony a změní případná chybová hlášení. Dle oficiálních stránek může toto nastavení zvýšit výkon trojnásobně. Proměnnou můžeme nastavit před spuštěním aplikace následujícím příkazem:

```
export NODE_ENV=production
```

Další možností je nastavovat proměnnou automaticky při startu systému v systémovém souboru `/etc/init/env.conf` nebo `/etc/systemd/system/myservice.service` [1].

Nyní lze spustit samotný server (příkaz je alias pro `npm run start`):

```
npm start
```

JavaScriptový kód se spustí ve výchozím režimu kompatibility nástroje *Babel*. Nástroj lze konfigurovat souborem `/.babelrc`.

5.4 Model

Model je část serveru, která má na starost práci s daty, operace nad soubory a také poskytuje například emailové služby. Tato část je odstíněna od požadavků a odpovědí, jednotlivé moduly lze používat samostatně, potřebná data se funkcím předají v parametrech. Funkce mohou, ale nemusí, vracet hodnotu.

Modul `emailManager` obsahuje funkci `sendProjectFinished`. Funkce se používá při dokončení renderování výsledného videa. Jako parametry požaduje emailovou adresu příjemce, případně více adres oddělených čárkou, ID projektu a výsledek zpracování (`true` pro úspěch, `false` při chybě). Funkce zasílá emaily skrze SMTP server v konfiguraci. Aplikace

díky tomu nevyžaduje vlastní emailový server. V případě úspěchu se email zašle pouze vlastníkovi projektu. V případě, že během zpracování došlo k chybě, je email zaslán vlastníkovi i administrátorovi. K odesílání emailů je využívána knihovna *nodemailer*[4].

Modul `fileManager` má na starost získávání informací o nahraných souborech. Obsahuje funkci `getDuration`, která je asynchronní. Funkce vrací *Promise*[5]. V případě, že se dotážeme na délku video nebo audio souboru, vytvoří se potomek procesu a v něm se spustí příkaz *FFmpeg* pro zjištění délky média. Jakmile je délka zjištěna, vrací se hodnota funkcí `resolve`. Dotaz na jiný typ souboru vrací hodnotu `null`. Více o vytváření procesů v kapitole 5.4.2.

Modul `mltxmlManager` usnadňuje práci s *XML* souborem generovaným pro program *MLT*. Modul má na starost ukládání souboru `project.mlt`, jako parametry vyžaduje ID projektu a obsah kořenového elementu `<mlt>`, včetně samotného `<mlt>` tagu. Funkce doplní záhlaví XML a celý řetězec zapíše do souboru projektu. Dále obsahuje funkce na získání relativní cesty k MLT souboru (`getMLTpath`) a k adresáři projektu (`getWorkerDir`). Zbývající funkce pracují přímo s XML, zjednodušují například vytváření playlistů nebo tractorů.

Modul `timeManager` slouží k práci s časovými značkami. V projektu používám pro označení časových okamžiků a k označení doby trvání čas ve formátu „00:00:00,000“, přičemž doba trvání nesmí být nulová. Tento modul obsahuje funkce na sčítání a odčítání času, na půlení časových intervalů a kontrolu, zda je formát doby trvání validní (`isValidDuration`). Modul je používán jak na straně serveru, tak v React komponentách.

5.4.1 Import modulů

Moduly zastřešují související funkce do jednoho celku. Moduly obsahují proměnné a funkce, které se zpřístupní importováním. Aby je bylo možné importovat, je nutné v modulu uvádět klíčové slovo `export`. Pokud máme v modulu více funkcí, musíme uvést, kterou funkci chceme importovat. Při neuvedení se importuje výchozí export. Pokud chceme používat více funkcí jedním importem, je nutné v modulu veškeré funkce a proměnné obalit do společného objektu, který bude výchozí export. Následující ukázka využívá jednoho hlavního objektu, který je výchozím exportem.

```
1  export default {
2      subDuration(durationA, durationB) {
3          // ...
4          return subResult;
5      },
6      addDuration(durationA, durationB) {
7          // ...
8          return addResult;
9      },
10     // ...
11 }
```

Pokud chceme moduly použít, je postup jak na straně serveru, tak i na straně klienta (v React) stejný:

```
1 import timeManager from '../../models/timeManager';
2 let actualTime = timeManager.addDuration(timeA, timeB);
```

[4]nodemailer – npm balíček pro snadné odesílání emailů s podporou HTML i Unicode, https://github.com/nodemailer/nodemailer.

[5]Promise – objekt, který reprezentuje budoucí úspěšné/neúspěšné dokončení asynchronní události a jeho budoucí hodnotu, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

Moduly jsou používány v rámci jednoho projektu, nejedená se o *npm* balíčky. Jsou novější alternativou k používání příkazu `require`.

5.4.2 Práce s procesy

Díky událostem a asynchronnímu přístupu k blokujícím požadavkům není problém čekat na mnohem náročnější operace, než je práce se souborovým systémem. V projektu je potřeba zpracovávat, konvertovat a získávat informace o multimédiích. Pro JavaScript dokáže vytvořit potomka, který zpracuje videosoubor, a po skončení potomka pracovat s jeho výstupem. U PHP by to byl problém a čekající procesy by mohly způsobit vyčerpání prostředků serveru. Práci s procesy zpřístupňuje modul Child Processes (`child_process`). Z něj využívám funkci `exec`, která vytvoří shell a umožní v něm vykonávat příkazy. Po dokončení posledního příkazu je k dispozici obsah standardního výstupu a standardního chybového výstupu. Použití příkazu `exec` demonstruje následující ukázka.

```
1 exec(‘ffmpeg −i ${filepath} 2>&1 | grep Duration | cut −d \’ \’ −f 4 | sed s/,// | sed s/\\\\.//‘,
2     (err, stdout, stderr) => {
3         if (err) console.error(err);
4         else {
5             console.log(stdout.trim());
6         }
7 });
```

Funkce `exec` vytvoří potomka a v rámci něj získá informaci o délce souboru. Délku vypíše v této ukázce na obrazovku, ale pomocí *Promises* je možné vytvořit funkci `getDuration` a délku asynchronně vracet.

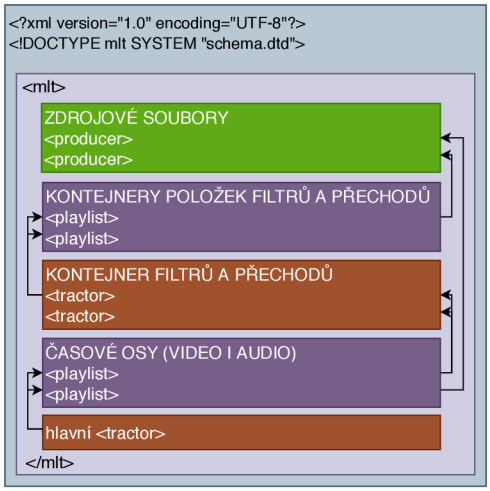
Při požadavku na renderování se zkontroluje, zda v projektu již neexistuje soubor `processing`. Pro renderování projektu se obdobným způsobem, pomocí `exec`, volá program `mlt`. Po dokončení je odstraněn příznak pro probíhající zpracování (`processing`) a je odeslán email o úspěchu. Pokud dojde k chybě, je odeslán chybový email. Standardní výstup a standardní chybový výstup je přesměrován do souborů `stdout.log` a `stderr.log`.

```
1 exec(‘cd ${projectPath} && melt project.mlt −consumer avformat:output.mp4 acodec=aac
2 vcodec=libx264 > stdout.log 2> stderr.log‘, (err) => {
3     if (err) console.error(‘exec error: ${err}‘);
4     fs.unlink(path.join(projectPath, ’processing’), (err) => {
5         if (err) console.error(err.stack);
6     });
7     if (isset(req.body.email)) {
8         emailManager.sendProjectFinished(req.body.email, req.params.projectID, !(err));
9     }
10 });
```

5.4.3 Generování XML

Se soubory XML se pracuje v modelu i controlleru. Jelikož se jedná o práci s daty, je struktura generovaných souborů popsána v této kapitole. Struktura je znázorněna graficky na obrázku 5.2.

Se založením nového projektu je vytvořen soubor `project.mlt` obsahující hlavičku XML a odkaz na DTD schéma umístěné v repozitáři projektu MLT. Poté následuje kořenový tag `<mlt>`. Uvnitř se nachází jeden playlist pro výchozí stopu `videotrack0` a hlavní kontejner `<track>` obsahující odkazy na všechny stopy v projektu. Tento hlavní kontejner musí být



Obrázek 5.2: Obecné schéma prvků generovaných souborů s vyznačenými vazbami. První řádky obsahují deklaraci XML, odkaz na DTD. Uvnitř kořenového tagu se nacházejí zdrojové soubory, kontejnery položek filtrů a přechodů, kontejnery filtrů a přechodů, časové osy a hlavní tractor. Pořadí těchto skupin je nutné dodržet.

vždy poslední položkou v kořenovém `<mlt>`. Následující ukázka kódu demonstruje obsah souboru `project.mlt` po vytvoření nového projektu:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mlt SYSTEM "https://raw.githubusercontent.com/mltframework/mlt/master/src/modu...
3  <mlt>
4      <playlist id="videotrack0"/>
5      <tractor id="main">
6          <multitrack>
7              <track producer="videotrack0" />
8          </multitrack>
9      </tractor>
10 </mlt>
```

Uvnitř kořenového elementu se nejprve nacházejí zdrojové soubory (elementy `<producer>`) u kterých jsou poznamenány následující vlastnosti – absolutní cesta k souboru (`resource`), typ souboru (`musecut:mime_type`), původní název souboru (`musecut:name`) a v případě video souboru délka v milisekundách (`length`). Zdrojové soubory mají id s prefixem „producer“ a 20 znaky náhodně generovanými při nahrávání souboru. Po posledním elementu `<producer>` jsou umístěny pomocné playlisty.

Pomocné playlisty představují kontejnery pro položky filtrů a přechodů. Uvnitř těchto playlistů se nachází vždy jedna položka `<entry>`. Před touto položkou může být v případě playlistu sloužící pro přechod položka `<blank>`. Pomocné playlisty jsou identifikovány jako „playlist“ a pořadové číslo playlistu. Nové playlisty jsou vkládány na začátek této sekce.

Po playlistech následují tractory (elementy `<tractor>`). Tyto elementy představují kontejnery pro filtry a přechody. Nesou informace o aplikovaných filtrech a přechodech. Přechody a filtry aplikují na pomocné playlisty definované výše. Tractor může obsahovat zároveň filtry i přechody. Identifikovány jsou jako „tractor“ a pořadové číslo tractoru. Nové tractory jsou vkládány na konec sekce, před stopu s id „videotrack0“;

Předposlední sekcí v generovaných souborech je část určená stopám. Stopy jsou realizovány jako playlisty. Rozlišují se video a audio stopy. Video stopy jsou identifikovány jako

„videotrack“ a pořadové číslo video stopy, audio stopy jako „audiotrack“ a pořadové číslo audio stopy. Nové stopy jsou vkládány na konec této sekce, před hlavní tractor s id „main“. Playlisty obsahují posloupnost položek časové osy. Mohou obsahovat `<entry>` s odkazem na `<resource>`, v případě jednoduché položky, nebo `<entry>` s odkazem na `<tractor>` v případě aplikovaného filtru nebo přechodu. Mezery mezi položkami řeší vložení elementu `<blank>` mezi položky.

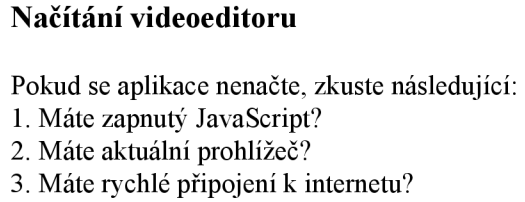
Posledním elementem v kořenovém elementu `<mlt>` musí být vždy hlavní tractor s id „main“. Tento tractor obsahuje odkaz na všechny stopy. Tento element je pouze jeden a nemůže být smazán. Načítá jej program *MLT*, z tohoto elementu jsou skrze odkazy dosažitelné všechny ostatní elementy.

5.5 View

View je část zodpovědná za zobrazení HTML stránek uživateli. Nepoužívá se při dotazech na API, to je určeno ke strojovému zpracování, ne ke komunikaci s koncovým uživatelem. View se nachází ve složkách `/views` a `/react`. Ve složce `/views` se nacházejí šablony. Složka `/react` obsahuje React komponenty.

5.5.1 Šablony

Framework *Express* poskytuje nástroje pro používání různých typů šablon. Protože pro tvorbu uživatelského rozhraní jsou použity React komponenty, zvolil jsem jednoduché HTML soubory. V těchto souborech je kostra pro namapování React komponent na DOM HTML. V souborech se rovněž odkazuje na CSS styly a na JavaScriptový soubor, který obsahuje React elementy. Aktuálně se používají 3 šablony – `main.html`, `project.html`, `finished.html`. Šablona `main.html` slouží k zobrazení úvodní obrazovky. Obsahuje přípravu na modální okno pro vytvoření nového projektu. Šablona `project.html` poskytuje kostru pro stránku s editorem videa. Šablona `finished.html` slouží k zobrazení obrazovky během zpracovávání videa. Stránka se periodicky obnovuje. Pokud si stránky zobrazí uživatel bez zapnutého JavaScriptu, zobrazí se postup řešení problému s načítáním, obrázek 5.3.



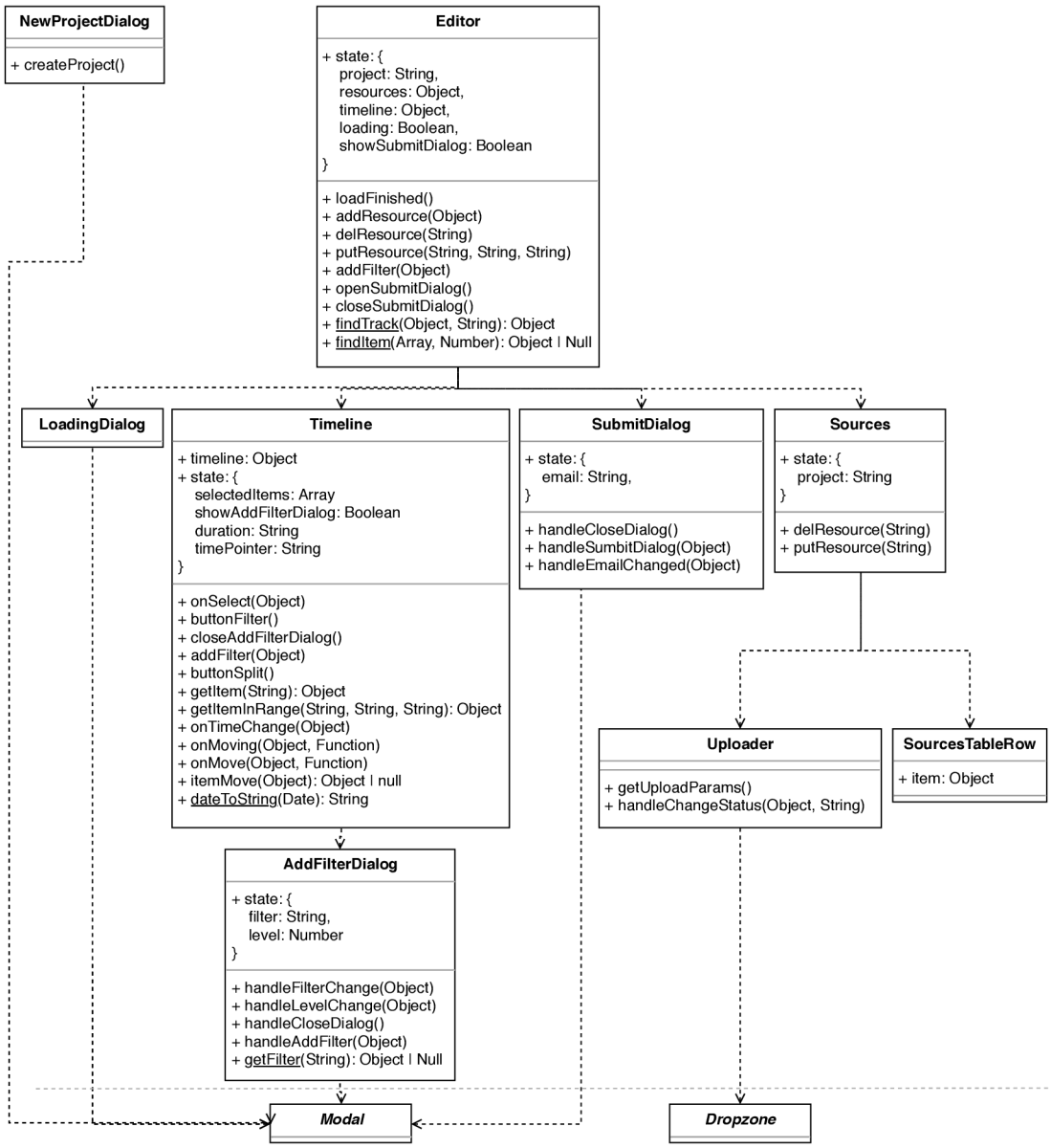
Obrázek 5.3: Šablona zobrazená v prohlížeči s vypnutým JavaScriptem.

5.5.2 React komponenty

React komponenty jsou využívány na úvodní stránce pro vytvoření projektu a na stránce se samotným editorem. Komponenty pro úvodní stránku jsou ve složce `/react/newProject`, komponenty editory pak ve složce `/react/editor`.

Na úvodní stránce je kořenovou komponentou třída `NewProjectDialog`. Třída vykreslí dialog na vytvoření nového projektu. Po stisknutí tlačítka „Vytvořit nový projekt“ se pošle

požadavek `POST /api/project`. Pokud API vrátí ID vytvořeného projektu, dojde k přesměrování na stránku s editorem. Třída `NewProjectDialog` je závislá na komponentě `Modal` z *npm* balíčku *react-modal*[6]. Přehled komponent a jejich vzájemných závislostí nabízí obrázek 5.4.



Obrázek 5.4: Diagram tříd. Potomci třídy Component s vyznačením závislostí typu „používání“.

Struktura komponent editoru je rozsáhlejší:

- `Editor` je kořenovou komponentou. Ve svém stavu drží informace o id projektu, seznamu zdrojů projektu a časové osy projektu. Stav projektu získá po načtení stránky

[6] react-modal – React komponenta pro vytváření dialogových oken, https://github.com/reactjs/react-modal.

z API požadavkem `GET /api/project/:projectID`. Poté, co úspěšně získá stav projektu, skryje překryvnou vrstvu s načítáním. Rovněž řídí viditelnost dialogu pro renderování projektu. Stav projektu propaguje do podřazených komponent `Timeline` a `Sources`. Pomocí callbacků s kořenovou komponentou komunikují podřazené komponenty při požadavcích na manipulaci se zdroji a časovou osou.

- `SubmitDialog` má na starost dialogové okno pro renderování projektu. Obsahuje formulář pro vyplnění emailu. Po vyplnění emailové adresy a potvrzení zašle na API požadavek `PUT /api/project/:projectID`. Pokud se vrátí kladná odpověď, je uživatel přesměrován na stránku `/project/:projectID/finished`.

- `Sources` vykresluje seznam zdrojů projektu a formulář pro nahrávání dalších zdrojů. Nahrávání souborů zajišťuje komponenta `Sources`. Třída `Sources` používá pro položky zdrojových souborů podřízenou komponentu `SourcesTableRow`. Pokud podřízené komponenty požádají o změnu zdrojových souborů / časové osy, předá požadavek nadřízené komponentě `Editor`.

- `SourcesTableRow` vykreslí jeden řádek v seznamu zdrojových souborů. Řádek si jako svůj stav udržuje informace o položce. Položku zdrojů lze přidávat na časovou osu nebo smazat z projektu. Komponenta volá API – pro přidání položky na časovou osu `PUT /api/project/:projectID/file/:fileID` a pro smazání souboru `DELETE /api/project/:projectID/file/:fileID`. Pokud API vrátí odpověď bez chyby, komponenta požádá nadřazenou komponentu `Sources` o změnu zdrojových souborů / časové osy.

- `Uploader` slouží k vykreslení formuláře pro nahrávání souborů. Má za úkol nakonfigurovat komponentu `Dropzone` z *npm* balíčku. Komponentě předá funkce pro zpětná volání při změně stavu (dokončení nahrávání, selhání nahrávání) a tato volání obsluhuje. Při úspěšném nahrání souboru požádá nadřazenou komponentu `Sources` o přidání souboru do seznamu zdrojových souborů.

- `Timeline` má na starost časovou osu. Časová osa není komponenta React. K inicializaci časové osy dochází ve funkci `componentDidMount`, která se volá po umístění komponenty do virtuálního DOM. Při změně položek časové osy dojde k překreslení pomocí funkce `componentDidUpdate`. Komponenta má na starost také přesun položek, rozpůlení položek a přidávání filtrů a zobrazování dialogového okna pro přidání filtru – komponenty `AddFilterDialog`. Komponenta si ve stavu uchovává pole s označenými položkami na časové ose a informaci, zda má být viditelné dialogové okno pro přidání filtru. Pokud dialog požádá o přidání filtru, je požadavek předán nadřazené komponentě `Editor`.

- `AddFilterDialog` vykresluje formulář pro přidání filtru. Ve svém stavu si uchovává hodnoty z formuláře – zvolený filtr a jeho hodnotu. Při potvrzení formuláře, komponenta vytvoří objekt flitru a požádá nadřazenou komponentu `Timeline` o přidání filtru.

5.6 Controller

Controllery mají za úkol přijmout požadavek, zpracovat parametry požadavku, na základě požadavku vykonat s pomocí modelu požadované akce a poté zajistit odpověď. Odpověď

může controller vytvořit sám nebo použije view. Controller je řadič – řídí ostatní součásti. Složka `/controllers` obsahuje 3 controllery – `apiController`, `errorController` a `mainController`.

5.6.1 apiController

Nejrozsáhlejší controller je `apiController`. Ten nese zodpovědnost za celé API. Má na starost přijímaní požadavků, manipulaci s XML i vytváření odpovědí. Tento controller nepoužívá view, neboť odpovědi API jsou ve formátu *JSON*. Na konci souboru `apiController` se nacházejí také 3 pomocné funkce – `fileErr` (vytvoří odpověď s chybou čtení projektu), `isNaturalNumber` (test na nezáporná celá čísla) a `isset` (kontrola existence proměnných). Tyto funkce jsou pomocné a jsou používány pouze v rámci souboru.

Nejjednodušším požadavkem je `GET /api`. Obsluhu zajistí funkce `default`, která pouze vrátí informaci o umístění dokumentace k API. Nejpozději na konci každé hlavní funkce controlleru dojde k vygenerování odpovědi. V případě chyby dojde k vytvoření odpovědi ihned po zaznamenání.

Vytváření odpovědí

Každá odpověď obsahuje atribut `msg`, v případě úspěchu může obsahovat další položky, v případě neúspěchu obsahuje odpověď vždy položku `err` a `msg`. Jak vypadá odpověď ve funkci `projectFilePOST`, pokud nepřiložíme nahrávaný soubor, ukazuje následující ukázka.

```
1 res.status(400);
2 res.json({
3     err: 'Chybi soubor.',
4     msg: 'Telo pozadavku musi obsahovat soubor k nahrani.',
5 });
6 return;
```

V případě úspěchu chybí položka `err`, a naopak odpověď obsahuje informace o nahraném souboru (identifikátor souboru, MIME typ a délku ve formátu „00:00:00,00“).

```
1 res.json({
2     msg: `Upload of "${filename}" OK`,
3     resource_id: fileID,
4     resource_mime: mimeType,
5     length: length,
6 });
```

Zpracování parametrů požadavku

Většina funkcí controlleru potřebuje k činnosti upřesňující parametry HTTP požadavků. Parametry jsou dostupné skrze objekt požadavku (první předávaný parametr). Parametry, které jsou součástí cesty v URL (projectID, fileID) jsou dostupné skrze objekt `req.params`. Tyto parametry jsou vždy definované. Pokud uživatel parametr v URL vynechá, router nezavolá obslužnou funkci a zobrazí uživateli chybu 404. V některých případech by nebylo možné uvádět všechny parametry přímo do URL. Parametry mohou být také v těle požadavku. K těmto parametrům se přistupuje pomocí `req.body`. Parametry mohou v požadavku chybět, controller musí na začátku sám ověřit uvedení povinných parametrů funkcí `isset`.

```
1 // Required parameters: track, item, filter
2 if (!isset(req.body.track, req.body.item, req.body.filter)) {
```

```
    res.status(400);
    res.json({
        err: 'Chybi parametry.',
        msg: 'Chybi povinne parametry: "track", "item", "filter".',
    });
    return;
}
```

Odlišně se přistupuje k požadavkům během nahrávání souboru. Soubory k nahrání jsou rovněž uvnitř požadavků jako parametr. Nejsou ve formátu *JSON*, ale *multipart/form-data*. Během nahrávání není potřeba s daty manipulovat. Při zahájení nahrávání se určí soubor, do kterého se bude zapisovat, a s dokončením nahrávání vrátí server odpověď o úspěšnosti operace. K tomu slouží rozšíření *busboy*. Při přijetí požadavku se otestuje, zda obsahuje požadovaná data a poté se data přesměrují do souboru.

Práce s XML

Serverová část je postavená na práci s XML. Na parsování XML souborů je používán balíček *jsdom*[7]. Node.js neobsahuje nástroje pro práci s DOM, které jsou v prohlížečích k dispozici. Zpracování XML pomocí SAX parseru nepřipadá v úvahu. SAX parsery procházejí soubory sekvenčně od začátku do konce. DOM přístup umožňuje náhodný přístup k elementům v XML dokumentu. Knihovna poskytuje objekt `window` a `document`, skrze který jsou dostupné metody známé z prohlížečů (`document.getElementById`, `document.querySelector` apod.). Soubor lze načíst funkcí `JSDOM.fromFile`. Při ukládání lze získat celé XML pomocí atributu `outerHTML` u kořenového elementu `<mlt>` a poté uložit získaný textový řetězec do souboru.

Práce se souborovým systémem

Aby mohl controller správně pracovat, potřebuje přístup k souborům ve složce `WORKER`. Práci se souborovým systémem umožňuje modul *File System* (`fs`). Používání funkcí modulu `fs` je podobné POSIX funkcím jazyka C. Všechny funkce mají synchronní a asynchronní variantu. U I/O operací je silně doporučeno používat asynchronní práci se souborovým systémem. Asynchronní funkce mají jako poslední parametr callback, který se zavolá po dokončení operace. Callbacku je jako první parametr předán objekt pro uchování chyb a poté volitelně data operace. Callback je volán při úspěchu i neúspěchu, neúspěch je indikován objektem 1. parametru. Dalším modulem spojeným se souborovým systémem je `path`, který poskytuje nástroje pro práci s cestami k souborům a adresářům. Jak vypadá asynchronní čtení souboru ukazuje následující ukázka.

```
fs.open(path.join(projectPath, 'processing')), 'wx', (err, fd) => {
    if (err) throw err;
    // prace se souborem
    fs.close(fd, (err) => {
        if (err) console.error(err.stack);
    };
});
console.log('stale se muze cekat na nacteni souboru);
```

Funkce `fs.open` je zavolána, vytvoří požadavek souborovému systému a namísto aktivního čekání je funkce uspána a pokračuje se dál vykonáváním příkazů pod `fs.open`.

[7] busboy – implementace standardů pro práci s DOM pro Node.js, https://www.npmjs.com/package/jsdom.

V tomto případě by se vypsal do terminálu text „stale se muze cekat na nacteni souboru“ a k práci se souborem by se JavaScriptový engine vrátil až po zpřístupnění požadovaného souboru. Z toho plyne, že I/O operace jednoho požadavku nebrzdí vykonávání souběžných požadavků.

Generování identifikátorů

Při vytváření nového projektu a při nahrávání souborů je potřeba vygenerovat jednoznačný identifikátor. Generování zajišťuje `apiController` pomocí balíčku *nanoid*[8]. Pro nový projekt se vytváří 32znakový identifikátor, pro nahrávaný soubor generuje 21 znaků. Identifikátory obsahují číslice, velká a malá písmena anglické abecedy a znak podtržítka a spojovníku (A-Za-z0-9_-). Teoreticky by neměla nastat kolize identifikátorů. Aby byla pravděpodobnost kolize 32znakových identifikátorů 1 %, museli bychom generovat 1000 identifikátorů za sekundu po dobu větší, než 1 kvadrilion let. V případě 21znakového identifikátoru souborů bychom museli nahrávat 1000 souborů za sekundu po dobu přibližně 41 milionů let [14]. Součástí projektů mohou být soukromé soubory, které budou chtít uživatelé ochránit před náhodným hádáním ID projektu. Teoreticky je možné procházet všechny možné identifikátory hrubou silou. Hrubou silou by ale bylo možné obejít i zabezpečení pomocí uživatelských účtů. Pokud bychom chtěli získat konkrétní projekt hrubou silou, trvalo by nám to opět více než kvadrilion let[9]. Do budoucna se plánuje zkombinovat zabezpečení pomocí ID projektů s přihlašováním pomocí centrálního autentizačního serveru VUT.

5.6.2 errorController

Pokud dojde během řádné obsluhy požadavku k výjimce, nebo pokud je zavolána funkce `next`, předá router obsluhu požadavku controlleru `errorController`. Tento controller obsahuje pouze funkci `default`. Dojde k zalogování chyby a odeslání odpovědi s chybovým kódem 500 a JSON odpovědí „Internal server error occured.“.

```
1 exports.default = (err, req, res, next) => {
2     console.error(err.stack);
3     res.status(500);
4     res.json({
5         msg: 'Internal server error occured.',
6     });
7 };
```

5.6.3 mainController

Controller `mainController` zpracovává požadavky, u kterých se očekává odpověď ve formě HTML pro vykreslení webovým prohlížečem. Používá se pro 3 situace. Uživatel navštíví úvodní stránku s možností vytvořit nový projekt, uživatel zobrazí stránku s editorem, uživatel zobrazí stránku s výsledným videem / s obrazovkou informující o stavu zpracování. Činnost controlleru je jednoduchá. Při zobrazení úvodní stránky nebo stránky s projektem controller pouze předá požadavek na vykreslení view.

```
1 exports.main = (req, res) => res.render('main', {});
```

[8] nanoid – bezpečný generátor identifikátorů, které mohou být použité v URL, https://github.com/ai/nanoid.

[9] K výsledku jsem dospěl na základě kalkulačky na stránkách http://daleswanson.org/things/password.htm a https://random-ize.com/how-long-to-hack-pass/

Při požadavku na výsledné video se controller podívá do adresáře projektu po příznaku probíhajícího zpracování (soubor `processing`). V případě probíhajícího zpracování požádá view o vykreslení čekací obrazovky. Pokud se video nezpracovává, podívá se, jestli existuje výsledný soubor `output.mp4`. Pokud soubor neexistuje, nemá projekt výsledné video, a je vrácena chyba 404. Pokud je soubor k dispozici, zašle se v odpovědi přímo samotný soubor, odpovědí není HTML stránka.

```
1  fs.access(path.join(projectPath, req.params.projectID, 'processing'), fs.constants.R_OK, (err) => {
2      if (err) {
3          const outputFile = path.resolve(path.join(projectPath, req.params.projectID, 'output.mp4'));
4          fs.access(outputFile, fs.constants.R_OK, (err) => {
5              if (err) res.sendStatus(404);
6              else res.sendFile(outputFile);
7          });
8      }
9      else res.render('finished', {});
10 });
```

Kapitola 6

Testování

Testování slouží k odhalení chyb a k vylepšení uživatelského zážitku. Cíl testování může být různý. Nejprve se v této kapitole věnuji vyhodnocení splnění případů užití, testování vykreslování (UI) a testování uživatelského zážitku (UX), ve druhé části se zaměřuji na testování modelu a testování stavu aplikace.

6.1 Vyhodnocení splnění případů užití

Před zahájením testování s uživateli je dobré ověřit, zda implementovaný systém odpovídá potřebám a požadavkům uživatelů. Požadavky na řešení jsou uvedeny v kapitole 3.2.

Práce se zdroji je popsána obrázkem 3.5, tato část je implementována. „Nahrát multimediální soubor“ je možné metodou drag&drop i dialogem. „Odebrat multimediální soubor“ lze, soubor nesmí být použit na časové ose. Dále je možné „Vložit video nebo audio na konec časové osy“ a „Vložit obrázek na konec časové osy“ (což zahrnuje „Nastavit dobu trvání“).

Práce s přehrávačem náhledu je popsána na obrázku 3.6. Tuto část se nepodařilo implementovat. Teorii potřebnou k implementaci popisuji v kapitole 4.2, ukázky demonstrující použití se nacházejí ve složce projektu v `examples`, ale k implementaci ve videoeditoru nedošlo. Realizace by zahrnovala u neznámých souborů získávat zkonvertovanou verzi ze serveru, měnit aktuální klipy a připravovat následující, to vše pro libovolné množství stop. Realizace by byla nad rámec této bakalářské práce.

Práci s časovou osou popisuje obrázek 3.7. Tato část je implementována. Je možné „Přiblížit časovou osu“, „Posunout časovou osu do stran“ a „Nastavit pozici ukazatele“ (uchopením a posunutím). Uživatel nemůže ručně „Přidat nebo odebrat časovou osu“, k dispozici je automaticky jedna volná video stopa a jedna volná audio stopa.

Práce s položkami časové osy je popsána na obrázku 3.8. Položky je možné „Posunout na časové ose“ a „Přesunout na jinou časovou osu“ uchopením a přetažením. Dále je možné položku „Odstranit z časové osy“, „Rozdělit v bodě na 2 části“ a „Přidat filtr“. Dostupné jsou následující filtry: jas, kontrast, roztmívat obraz, zatmívat obraz, postupně zesílit zvuk, postupně zeslabit zvuk. Nebyla implementována možnost „Změnit začátek nebo konec ořezem“, funkci nebyla přiřazena priorita, neboť je možné stejného výsledku dosáhnout rozdělením položky na 2 části a smazáním jedné z částí. Dále nebyly implementovány přechody – „Přidat přechod mezi 2 položkami“. Při tvorbě výukových videí nejsou přechody nezbytné, z přechodů by byl implementován pouze efekt prolnutí a implementace ovládání by byla složitá. S implementací přechodů se počítá v budoucnu. Možnost „Zobrazit vlastnosti“ nebyla implementována. Bez přechodů je možné veškeré informace umístit do dialogu filtrů.

Přechody je možné používat pomocí API, omezení se týká uživatelského rozhraní. Nicméně videa s přechody není možné zatím rozdělit na 2 poloviny.

6.2 Testování uživatelského rozhraní

Už při návrhu editoru a uživatelského rozhraní jsem navrhované řešení konzultoval s neprofesionálními uživateli desktopových editorů. Během průzkumu jsem zjišťoval používané filtry a typy přechodů. Nakonec jsem filtry roztřídil do dvou kategorií – vylepšení a efekty. Zatímco vylepšení mohou zlepšit uživatelský zážitek z výukového videa, efekty odpoutávají pozornost od sdělení videa. Mezi vylepšení se zařadilo například úprava jasu, kontrastu, odstínu, světlosti, sytosti, ostrosti, vyvážení bílé, otočení a velikost, úprava polohy, ořezání, průhlednost. Mezi nepotřebné efekty se zařadil například sépiový a šedotónový filtr, negativ, rozmazání, posun barev, přidávání kazů do videa apod. Bylo důležité s uživateli provést toto rozdělení, aby mohly být implementovány pouze efekty zlepšující zážitek z videa. Omezení výběru filtrů a obecně funkcí vede uživatele k vytváření kvalitních videí (pro výukové účely).

Pro ověření navrženého rozhraní probíhalo uzavřené testování s vybranými uživateli. Vybíral jsem uživatele, kteří mají zkušenost s úpravou videí pomocí desktopových aplikací, ale zároveň tyto editory nepoužívají na profesionální úrovni. Potřeboval jsem uživatele, který zná rámcově práci s videi, ale zároveň může být při používání editoru nejistý. U části uživatelů byla volba vytvářeného videa i zařízení na uživatelích, aby se objevily možné problémy, část uživatelů naopak pracovala podle stanoveného protokolu. Tito uživatelé postupovali dle textové specifikace případů užití pro nejčastější úpravy záznamů přednášek, tabulka 3.1. Uživatel dostal zařízení se splněnými vstupními podmínkami a cílem bylo splnit kroky toku událostí. Výsledné video si nechali zpracovat na serveru a zaslat na emailovou adresu. Pozici ukazatele nastavovali uživatelé ručně (krok 4) z důvodu chybějící implementace náhledu.

Během testování byl uživateli k dispozici tester, testování probíhalo formou pozorování. Z hlediska implementace UI bylo během testování odhaleno odlišné vykreslování rozložení prvků při rozlišením větším než 2K. Dále bylo upraveno umisťování dialogových oken a ztmavení obrazovky při zobrazení okna.

Rozložení uživatelského rozhraní přišlo všem testovaným subjektům logické, věděli, kde se co nachází a k čemu jednotlivé části slouží. Připomínky byly k zadávání hodnoty filtru. Hodnota obrazových filtrů se nyní zadává posuvníkem, zadávání hodnoty do textového vstupu zůstalo pouze pro dobu trvání. Na řešení uživatelé kritizovaly chybějící funkce, které chyběli z důvodu nekompletní implementace. S implementovanými částmi uživatelského rozhraní byli uživatelé spokojení.

6.3 Testování API

Základním stupněm testování je použití analyzátoru kódu *ESLint*. Analyzátor dokáže odhalit syntaktické chyby, časté sémantické chyby v kódu a dále kontroluje jednotný styl kódu. Kontrolu lze provádět z kořenového adresáře projektu příkazem `npx eslint <soubor>`. Kontrolu lze provést nad jedním souborem nebo rekurzivně nad obsahem adresáře. Kontrola má smysl pro složky `/controllers`, `/models` a `/react`. Rozsah kontroly se nastavuje konfiguračním souborem `.eslintrc.js`.

V průběhu implementace byla průběžně testována správnost generovaného XML souboru. Soubor byl testován na syntaktickou správnost a správnost dle externího *DTD* sou-

boru. Soubor *DTD* je součástí projektu *MLT* a určuje, jakého obsahu může XML nabývat. Validní soubor XML je základem pro úspěšně vyexportovaný projekt. Kontrolu provádělo automaticky vývojové prostředí *IntelliJ IDEA* a namátkově jsem soubor ověřoval na portálu *TRUUGO*[1].

Správnost výstupního videa byla kontrolována i vizuálně. Renderování projektu jsem testoval pod 32bitovým Ubuntu 16.04 a pod 64bitovým Ubuntu 18.04. Server *prednasky.com*, na kterém je plánováno nasazení, používá operační systém Ubuntu, proto jsem projekt testoval právě pod tímto operačním systémem.

Při testování jsem s menšími obměnami vytvářel následující výstupní video: 1. položka jednoduchá, 2. položka s přechodem do 3. položky, 3. položka s aplikovaným filtrem a s přechody na předchozí i následující 4. položku a poslední 5. položka je bez přechodu, ale s aplikovaným filtrem, obrázek 6.1.

Obrázek 6.1: Vzorový projekt pro testování API. Zelené položky mají aplikován filtr, fialová značí přechod mezi položkami.

6.4 Rychlost zpracování, maximální délka

Aby mohlo řešení uživatelům nabídnout lepší služby, než stávající řešení, musí být schopné zpracovat video o délce delší než 30 minut, ideálně delší než 60 minut, viz srovnání stávajících řešení v tabulce 2.1.

Testování použitelnosti řešení z hlediska výkonu a maximální možné délky videa probíhalo na zařízení MacBook Air s 1,3GHz procesorem *dual-core Intel Core i5* a 4 GB operační paměti. Komunikace s vlastním řešením probíhala po lokální síti rychlostí až 300 Mb/s, s konkurenčními řešeními jsem komunikoval rychlostí až 50 Mb/s. Testy byly voleny tak, aby rychlost sítě nehrála roli.

Během testování jsem zkoušel zpracování videa o délce 1 hodiny a 40 minut při rozlišení 1280x720 a 30 snímcích za sekundu. Velikost videa byla 333 MB, video bylo kódováno ve formátu H264, audio ve formátu AAC, použitý kontejner byl MPEG-4. Video bylo nahráno na portál, vloženo do časové osy a byl na něj aplikován filtr pro zvýšení jasu. Poté byl projekt vyrenderován.

Nahrávání videa do vlastního řešení proběhlo za necelé 4 minuty. Zpracování videa trvalo 45 minut. Po celou dobu zvládal server odpovídat na požadavky ostatních uživatelů. Druhý testovaný nástroj – *Clipchamp Create* provádí úpravy na straně prohlížeče. Video bylo po přidání do projektu k dispozici po 10 sekundách. Renderování videa ovšem trvalo přibližně 2,5 hodiny.

V porovnání těchto přístupů vyhrává serverové zpracování videí. U kratších videí se náskok snižuje, ale stále převažuje. Implementované řešení zvládá zpracovat videa delší než 60 minut a překonává tak existující řešení, vyjma *WeVideo* s neomezenými parametry výsledných videí.

[1]TRUUGO XML Validator – bezplatný online XML validátor, včetně podpory externího DTD, https://www.truugo.com/xml_validator/.

Kapitola 7

Závěr

V rámci této práce vznikl REST API server, který slouží k úpravám multimediálních souborů. K API je přiložena dokumentace umožňující komukoliv vytvořit vlastní aplikaci nad tímto API. K provozování serveru není zapotřebí databáze ani webový server. K získávání informací o souborech používá *FFmpeg*, k vykreslování výsledného videa framework *MLT*. Server ukládá stav projektů do XML souborů. V případě potřeby může uživatel editoru znovu načíst projekt ve stavu, v jakém jej opustil. Také je možné v libovolném čase vzít složku s projektem ze serveru a nechat jej ručně zpracovat programem *MLT*. Stačí upravit cesty k souborům. Zpracování videa není omezeno velikostí souborů ani délkou.

Dále vznikl jednoduchý videoeditor (bez náhledu a přechodů) demonstrující základní použití API pro úpravu videí, zejména střih a základní filtry nad záznamy přednášek. Uživatelské rozhraní odpovídá zažitým zvyklostem z desktopových aplikací. V současné době umožňuje vkládání videí na časovou osu, střih, přesun, aplikování filtrů, postupného náběhu a postupného mizení. Uživatelské rozhraní tvoří React komponenty, je snadno rozšiřitelné. Celá aplikace je tvořena jako open source řešení a zdrojové kódy jsou umístěny na *GitHub.com*[1] pod licencí Apache-2.0.

Součástí této práce je návod pro zprovoznění řešení pro běžné použití i návod pro sestavení projektu určený budoucím vývojářům. Projekty jsou zabezpečeny unikátním ID. Délka identifikátoru neumožňuje získání konkrétního projektu hrubou silou. Aplikace je dále rozšiřovatelná. Do budoucna se počítá s rozšířením možností editoru o náhled a přechody a s napojením na uživatelské účty externí autentizační služby.

Dále vznikl ucelený přehled možností manipulace s multimediálním obsahem přímo v prohlížeči, včetně ukázek použití pro videoeditor. Efekty nebo přechody je možné simulovat přímo v prohlížeči, ale samotné renderování výsledného videa je lepší přenechat serverové části.

Tato práce má potenciál pro další rozvoj. Nejvíce prostoru vidím v možnostech editoru. Zbývá implementovat přechody, některé filtry a náhled videa. Do budoucna vidím možnost zpeněžení nástroje nabídnutím nástroje koncovým uživatelům nebo například online reklamním agenturám.

Za největší přínos považuji vytvoření REST API, díky kterému mohou vzniknout další zajímavé projekty nezávisle na platformě. Může tak vzniknout například nástroj pro úpravu a okamžité sdílení videí s ostatními uživateli. Se samotným editorem se počítá na portálu *presnasky.com*, kam by měl být nasazen do září 2019.

[1] GitHub repozitář projektu – dostupný z https://github.com/kudlav/videoeditor.

Literatura

[1] Performance Best Practices Using Express in Production. *Express - Node.js web application framework*, 2017, [cit. 2019-03-11].
URL https://expressjs.com/en/advanced/best-practice-performance.html

[2] Filter. *MDN Web Docs*, 2019, [cit. 2019-04-17].
URL https://developer.mozilla.org/en-US/docs/Web/CSS/filter

[3] Introducing JSX. *React – A JavaScript library for building user interfaces*, 2019, [cit. 2019-05-02].
URL https://reactjs.org/docs/introducing-jsx.html

[4] JavaScript Frameworks. *Wappalyzer*, 2019, [cit. 2019-01-17].
URL https://www.wappalyzer.com/categories/javascript-frameworks

[5] Tutorial: Intro to React. *React – A JavaScript library for building user interfaces*, 2019, [cit. 2019-03-11].
URL https://reactjs.org/tutorial/tutorial.html

[6] Adobe Corporate Communications: Flash & The Future of Interactive Content. *Adobe Blog*, 2017, [cit. 2019-01-17].
URL https://theblog.adobe.com/adobe-flash-update

[7] Debill, E.: Modulecounts. [cit. 2019-04-17].
URL http://www.modulecounts.com/

[8] Jackson, C.: The perils of using Internet Explorer as your default browser. *Microsoft Tech Community*, 2019, [cit. 2019-04-17].
URL https://techcommunity.microsoft.com/t5/Windows-IT-Pro-Blog/The-perils-of-using-Internet-Explorer-as-your-default-browser/ba-p/331732

[9] Malý, M.: REST: architektura pro webové API. *Zdroják*, 2009, ISSN 1803-5620, [cit. 2019-02-21].
URL https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[10] Pasquali, S.: *Mastering Node.js: build robust and scalable real-time server-side web applications efficiently*. Birmingham: Packt Publishing, druhé vydání, 2017, ISBN 978-1-78588-896-0.

[11] Pfeiffer, S.: *HTML5 - audio a video: kompletní průvodce*. Encyklopedie webdesignera, Brno: Zoner Press, první vydání, 2011, ISBN 978-80-7413-147-9.

[12] Rozentals, N.: *Mastering TypeScript - Second Edition*. Packt Publishing, druhé vydání, 2017, ISBN 9781786468710.

[13] StatCounter: Desktop Browser Market Share Worldwide. *StatCounter Global Stats*, [cit. 2019-04-17].
URL http://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-201803-20190

[14] Zhuravlev, A.: Nano ID Collision Calculator. [cit. 2019-04-26].
URL https://zelark.github.io/nano-id-cc/

Příloha A

Obsah přiloženého CD

CD obsahuje tyto položky:

- `aplikace` – zdrojové kódy
- `bp-wis.pdf` – PDF s technickou zprávou (verze odevzdaná do WIS)
- `bp-tisk.pdf` – PDF s technickou zprávou (tištěná verze)
- `bp-source` – zdrojový kód zprávy (LaTeX)
- `dokumentace` – vygenerovaná dokumentace REST API
- `video.mp4` – demonstrační video
- `plakatek.pdf` – plakátek formátu A2

Postup zprovoznění aplikace popisuje soubor `README.md` ve složce `aplikace`.